ColdFire® TCP/IP Stack Deep Dive
MCF5223x
ColdFire® Ethernet MCU Family
Overview

Enabling Ethernet Connectivity
The Freescale Controller Continuum is the industry's first and only roadmap for compatible 8-bit and 32-bit architectures.

From the ultra-low-end RS08 to the highest-performance ColdFire V4 devices, the Controller Continuum offers compatibility across our portfolio of consumer and industrial microcontrollers and microprocessors.

Development tools such as CodeWarrior® for ColdFire and Processor Expert™ simplify migration and upward compatibility.
MCF5223x ColdFire® with Embedded Flash & Ethernet MAC&PHY

**MCF5223x ColdFire Family**

**Targeted at Industrial Control Applications**
- Environmental Monitoring
- Remote Data Collection
- Medical Pumps and Monitors
- Power-over-Ethernet
- Security/Access Panels
- Lighting Control Nodes
- Vending Machines

**Key Features**
- 57 MIPS V2 Core with Enhanced Multiply and Accumulate for DSP-like functionality!

- Integrated Connectivity including:
  - 10/100 Ethernet Controller
  - 10/100 Ethernet Physical Layer
  - CAN 2.0B Controller
  - Cryptographic Acceleration Unit

- Additional control features include:
  - Up to 73 General Purpose I/O
  - 4ch. 32-bit timers with DMA support

- Starting from $7.99 suggested resale price
**V2 ColdFire Core**
- Up to 46 Dhrystone 2.1 MIPS @ 50MHz
- EMAC Module and HW Divide
  - No external bus

**Memory**
- 32KBytes SRAM
- Up to 256K bytes flash
  - 100K W/E cycles, 10 years data retention

**Key Features**
- 10/100 Ethernet MAC with PHY
- Three (3) UARTs
- Queued Serial Peripheral Interface (QSPI)
- I²C bus interface modules
- 4 ch. 32-bit timers with DMA support
- 4ch. 16-bit Capture/Compare/PWM timers
- 2 ch. Periodic Interrupt Timer
- 8 ch. PWM timer with enhanced DAC capabilities
- 4 ch. DMA controller
- Watchdog timer
- Real Time Clock (RTC) with 32kHz Oscillator
- 8 ch. 12-bit A-to-D converter with Simultaneous Sampling
- Up to 56 General-Purpose I/O
- MCF52236 Starting from $5.39 at 10k resale with 256K Flash
- MCF52232 Starting from $4.94 at 10k resale with 128K Flash

---

**ColdFire® MCF52232, MCF52236**
**New Ethernet Devices**

<table>
<thead>
<tr>
<th>Optional Additional Modules</th>
<th>BDM</th>
<th>PLL</th>
<th>GPIO</th>
<th>JTAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto</td>
<td>4-Ch, 32-bit Timer</td>
<td>4-Ch, DMA</td>
<td>PHY</td>
<td>10/100</td>
</tr>
<tr>
<td>CAN</td>
<td>4-Ch, 16-bit Timer</td>
<td>I²C</td>
<td>UART</td>
<td></td>
</tr>
<tr>
<td>Memory Options</td>
<td>2-Ch, PIT</td>
<td>QSPI</td>
<td>UART</td>
<td></td>
</tr>
<tr>
<td>128 KB Flash</td>
<td>4-Ch, 8-Ch. PWM</td>
<td>2 x 4-Ch., 12-bit ADC</td>
<td>UART</td>
<td></td>
</tr>
<tr>
<td>256 KB Flash</td>
<td>RTC</td>
<td>ColdFire® V2 Core</td>
<td>System Integration</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Single 3.3V Power Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temp Range: -40°C to +85°C; 0°C to +70°C</td>
</tr>
<tr>
<td>80 LQFP</td>
</tr>
</tbody>
</table>

---

Freescale Confidential Proprietary
MCF5223x – Ethernet Media Access Controller (MAC)

- The Ethernet MAC supports 10/100 Mbps Ethernet/IEEE 802.3 networks
- IEEE 802.3 full duplex flow control
- Support for full-duplex operation (40Mbps throughput) with a minimum system clock rate of 50MHz
- Support for half-duplex operation (20Mbps throughput) with a minimum system clock rate of 25MHz
- The ePHY (embedded PHYsical layer interface) is IEEE 802.3 compliant
- Supports both the media-independent interface (MII) and the MII management interface
- Full-/half-duplex support in all modes
- Requires a 25-MHz crystal for its basic operation
- Supports Loopback modes
MCF5223x - Cryptographic Acceleration Unit (CAU)

- Uses standard ColdFire® coprocessor interface and instructions
- Simple, flexible programming model
- Supports DES, 3DES, AES, MD5 and SHA-1.
- Architecture allows for future enhancements
- Supports all ColdFire® cores
FlexCAN – Controller Area Network

Figure 30-1. FlexCAN Block Diagram and Pinout
M52235EVB Evaluation Board and Development System

- Evaluation board with fully functional Power over Ethernet circuitry. Supports plug-in Zigbee daughter card
- Kit to include CD ROM, Power Supply, P&E BDM Cable, and Ethernet Crossover Cable
- Target Suggested Resale Price: $299

M52235EVB Software Support

- Free ColdFire_TCP/IP_Lite stack
- Free CodeWarrior® SPECIAL EDITION Included in Each Development Kit
- ColdFire Init – Graphical Initialization Tool
M52233DEMO Low cost demo board

M52233DEMO Low Cost Board
- Evaluation board with Plug-in Zigbee daughter card
- Kit to include CD ROM, Power Supply, and Ethernet Crossover Cable
- Target Suggested Resale Price: $99
- Available: May 2006

M52233DEMO Software Support
- Free ColdFire_TCP/IP_Lite stack
- Free CodeWarrior® SPECIAL EDITION Included in Each Development Kit
- ColdFire Init – Graphical Initialization Tool
EVB and DEMO Comparison

- **MCF52235**
  - 32K RAM 256K Flash, Ethernet with PHY, CAN, Crypto
  - 112 LQFP pin
- **Light Sensor**
- **PoE capabilities**
- **3 UARTs**
- **Supports plug-in Zigbee daughter card**

- **MCF52233**
  - 32K RAM 256K Flash, Ethernet with PHY
  - 80 LQFP pin
  - **Accelerometer (3 axis g sensor)**
  - **1 UART**
  - **Supports plug-in Zigbee daughter card**
ColdFire TCP/IP LITE Stack
ColdFire_TCP/IP_Lite Stack

Available from Freescale:
*InterNiche Technologies and Freescale have collaborated to provide an OEM version of InterNiche’s NicheLite™, ColdFire_TCP/IP_Lite*

Features
- Address Resolution Protocol (ARP)
- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Dynamic Host Configuration Protocol
- (DHCP) Client
- Bootstrap Protocol (BOOTP)
- Trivial File Transfer Protocol (TFTP)

Freescale Provided additional free software:
- Web Server with Flash File System
- Serial to Ethernet
- Sample TCP, UDP, clients and servers.
Additional Features from InterNiche
These features can be purchased from Interniche as add-ons

**SLIP**
- Serial Line IP – Used to communicate IP over a modem.

**PPP**
- Point to Point Protocol – Used to establish a connection with a ISP.

**SMTP client**
- Simple Mail Transport Protocol – Used to send a email message.

**SNMP**

**DHCP server**
- Dynamic Host Configuration Protocol – The server is used to assign IP addresses in a network.

**Telnet Server**
- Provides a method of “logging” onto the node from a remote location.

**DNS server**
- Dynamic Name Server – Used to translate URL names to a IP address.

**FTP client**
- File Transfer Protocol client – Used to transfer files to and from the device.

**FTP server**
- File Transfer Protocol server – Used to allow the transfer of files to and from the device.

**SSL**
- Secure Socket Layer – Provides a encrypted connection between to devices.

**Web Server**
- Or HTTP server – Used to send web pages to a web browser client.

<table>
<thead>
<tr>
<th>IPv6</th>
<th>IPv4</th>
<th>v4/v6</th>
<th>Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Auto-IP**
- DDNS
- DHCP Server
- DNS Client
- DNS Server
- Email Alerter
- FTP Server + Client
- IP Multicast
- IPSec/IKE
- NAT
- NAT-PT
- POP3
- PPP
- MS-CHAP
- MultiLink
- PPPoE
- RIP
- SSL/TLS
- SNMPv1
- SNMPv2(c)
- SNMPv3
- Telnet Server
- TFTP Client+Server
- Web Server
- HTML Compiler
- Web Server-SSL
- NicheTool
NicheStack PPP

Supports:
  • LCP
  • IPCP
Hayes dialing code
VJ Header Compression
PAP, CHAP and MS-CHAP security
Multiple Simultaneous Links
Supports DHCP
Compliant with RFC’s 1144, 1332, 1344, 1661, 1662 and 1994
PPPoE - PPP over Ethernet

Allows PPP connections through Ethernet adapters

- Used by broadband service providers to allow PPP authentication
- Maintains the familiar "dial-up experience" when connecting with a broadband modem
- Operates between PPP and Ethernet driver
- RFC 2516
MultiLink PPP

A method for splitting, recombining and sequencing datagrams across multiple logical data links.
Based on an LCP option negotiation.
Originally designed for multiple bearer channels in ISDN
RFC 1990
Secure Sockets Layer Library

Provides Secure Sockets Layer for embedded web services
Layer of security for HTTP web traffic between client and server
Based on public key asymmetric cryptography
Requires upgrade to NicheStack – won’t work with NicheLite/FreescaleLite
NicheStack SSL – Features

Adds SSL to web services
- NicheStack
- WebPort - HTTP Server
- NicheView Browser

RSA key exchange method with 1024 bit key generation and Triple DES encryption

Blocking and non-blocking modes

Supports IETF
- SSL v2.0
- SSL v3.0
- TLS v1.0

Includes
- SSL Library
- API Library for server-side SSL
- Static Library

Based on Open SSL
NicheStack SSL -- Benefits

Authenticated, encrypted communication
NicheStack SSL protects the integrity of the embedded device and its configuration
Agent
Uses UDP/IP
MIB-database
Variables
  • Statistic values about communication
  • Private extensions
  • Access rights configurable
MIB-compiler.
Advantages of InterNiche SNMP

Portable
MIB Compiler
  • An automated tool to help support new MIBs
Sample Code
  • implementation of MIB2, SNMPv3 RFCs
Complies with RFC standards
SMTP Email Alerter

Low cost mechanism for adding EMAIL reporting capability to embedded application.

- Supports Simple Mail Transfer protocol (SMTP)
- Sends predetermined messages from an embedded system to a local or remote email address
- Sockets interface makes porting quick and easy
- Supports multiple target email addresses
- Supports many individual messages and formats
- Compliant with RFC’s 821, 869 & 870
Telnet

Provides network accessibility for remote configuration and monitoring
Compatible with commercial TELNET Clients (Windows, NT UNIX, etc.)
Supports multiple TELNET sessions
Highly portable
Small memory requirement
Compliant with RFC 854
FTP Client/Server

Adds file server capabilities
Supports Passive mode
Multi user and multi session
Two Way Tasking - no special multitasking features are required
Run by polling from a central loop or take advantage of an RTOS suspend/resume feature
Can open sessions as a Server or Client
Compliant with RFC 959
NicheTool (included with NicheLite product)

Advanced Debugging and Tuning Suite
Included with NicheStack and NicheLite
Allows developers to rapidly view, debug and tune their target system
Reduces need for in-depth TCP/IP expertise
NicheTool Features

An expandable menu system with a command line interface (CLI)
Direct visibility into key networking structures
Access RFC 1213 MIB II statistics and approximately 100 extensions
Trace packets as they travel through stack layers
View buffer utilization
NicheTool Benefits

It allows engineers to quickly:
Verify the stack build
Verify hardware / network connections
Find errors and trace network connections
Optimize throughput and memory utilization
Create customized menus for the target system
InterNiche Prices

Have customers contact InterNiche for price quotes

Single product license pricing

- SMTP $3000
- PPP $3000
- SNMP v1/v2c $5000
- FTP $2000
- SSL $10,000 (requires NicheStack which is $8000)
ColdFire TCP/IP Lite Project Overview
The Directory Structure

ColdFire_Lite = Interniche stack and projects
Runtime_loaded_web_page_example = Loadable labs
Directory Details – Runtime Loadable Demos/Labs

Runtime_loaded_web_page_example directory
This directory contains the runtime loadable demos/labs.
Directory Details – ColdFire_Lite

ColdFire_Lite directory

The ColdFire_Lite directory contains the TCP/IP stack and Web Server Firmware.
ColdFire_Lite Project File

The project File is used to open the project in CodeWarrior®.
The NichLite directory

The NichLite directory contains the source to the TCP/IP stack.

ColdFire_Lite\src\projects
The Freescale_HTTP_Web_Server directory contains the source code for the Freescale Web Server.

ColdFire_Lite\src\projects\example
Opening ColdFire TCP/IP Lite
Locate and Open the TCP/IP/Web Server Project

Close all open CodeWarrior® Project Windows.
Choose File > Open
Browse to the ColdFire Lite Directory.

• This will be located where you unzipped the ColdFire Lite project, or if
  you are using a Freescale laptop
Directory Details – ColdFire_Lite

ColdFire_Lite directory

The ColdFire_Lite directory contains the TCP/IP stack and Web Server Firmware.
ColdFire_Lite Project File

The project File is used to open the project in CodeWarrior®.
The ColdFireLite project
Project Files
int eng_udp_callback(PACKET pkt, void *param)
{
    struct udp *pup = (struct udp *) pkt;
    pup->data_len = pkt->nb_len;
    pup->host_ip = pkt->host;
    eng_process_udp_packet(pup->data, pup->data_len);
    return 0;
}
```c
int eng_udp_callback(PACKET pkt, void *parm)
{
    struct udp *pup;
    unsigned int data_len;
    void *data;
    ip_addr host_ip;

    /* get pointer to UDP header */
    pup = (struct udp *)pkt->nb->nb_prot;
    pup -= 1;

    data = pkt->nb->nb_prot;
    data_len = pkt->nb->nb_len;
    host_ip = pkt->nb->nhost;
    eng_process_udp_packet( host_ip, data, data_len );

    udp_free( pkt );
    return( 0 );
}
```
LAB: Flashing the board with ColdFire_Lite
Flashing and Booting the board

Connect the board via USB and serial to the PC.
Select the ColdFire_Lite project
Flash the ColdFire_Lite project
Setup Hyperterminal for
  • 115200, 8, n, 1
Run the project
Build the project by clicking on the MAKE icon (circled in RED)
Flash Programming Caution

CodeWarrior® for ColdFire **DOES NOT** behave like the HC08 and HC12 tools when downloading code to internal FLASH. Code **MUST** be downloaded by the Flash programmer to internal Flash of the MCF52235 as described in the previous slides. Once code is programmed in Flash it can be debugged using the procedure in the following slides.
Selecting the XML File

Select the M5223EVB-25MHZ xml file.
Flash Programmer Screen

After Loading the XML file, the Flash Programmer will show following screen. Note the Target Processor, and RAM memory buffers are setup automatically from the XML file.
Erase the Flash by selecting Erase/Blank Check, and clicking the Erase button. Watch the Status window for errors.
Programming the Flash

After the Erase is Complete, go to the Program/Verify window and click on the Program button.
Click on the Run icon, circled in **RED** below. This will execute the code in flash. If you have an external power supply, you could also disconnect the USB from the board and hit reset.
Connecting the Serial Port

Connect the serial port on the demo board to the PC. Then open hyperterminal and configure for 115Kbaud, 8, n, 1, no flow control. Hit enter until you see the ‘INET>’ prompt then type ‘tkstat’.
Serial output at boot (115200, 8, n, 1)

Running ColdFire TCP/IP-Lite stack

Copyright 2006 by Freescale Semiconductor Inc.
Use of this software is controlled by the agreement
found in the project LICENSE.H file.
Built on Sep 27 2006 19:25:56

Heap size = 27136 bytes

IP Address = C0A80163
Gateway = C0A80101
Mask = FFFFFFF00
etheraddr = 00:BA:DB:AD:01:02

Starting ints.
Calling netmain()...
InterNiche ColdFireLite TCP/IP for Coldfire, v3.0

Copyright 1997-2006 by InterNiche Technologies. All rights reserved.
Preparing device for networking
Ethernet started, Iface: 0, IP: 192.168.1.99
IP address of : 192.168.1.99
INET>
INET>
The ColdFire_Lite target

The ColdFire_Lite target is the barbones Interniche stack only target.

Use this project for customers who are looking for a clean port of the Interniche stack to the MCF5223x.

The only features available with this target is the serial console and ping.
Ping the ColdFire from the PC

Open a DOS window
Type: ping 192.168.1.99 (the ColdFire default IP address)
Ping the PC from the ColdFire

At the INET prompt type:

```
INET>
INET>
INET> host 192.168.1.1
INET> ping
ping sent, check icmp for reply
INET> icmp
ICMP layer stats:
icmpInMsgs 5   icmpInErrors 0, echoReqs 4, echoReps 1, unhandledTypes: 0
icmpOutMsgs 5   icmpOutErrors 0,
INET> ping
ping sent, check icmp for reply
INET> icmp
ICMP layer stats:
icmpInMsgs 6   icmpInErrors 0, echoReqs 4, echoReps 2, unhandledTypes: 0
icmpOutMsgs 6   icmpOutErrors 0,
INET>
```
Type: help diag

```
SNMP Station: diagnostic commands:
arps      - display ARP stats and table
buffers   - display free q buffer stats
queues    - dump packet buffer queues
dbytes    - dump block of memory
debug     - set IP stack debug tracing
d trap    - try to hook debugger
iface     - display net interface stats
linkstats - display link layer specific stats
tcp       - display TCP stats
sockets   - display socket list
tbconn    - tcp BSD connection stats
tbsend    - tcp BSD send stats
tbrcv     - tcp BSD receive stats
allocsize - set size for alloc() breakpoint
instat    - display IP layer stats
icmpstat  - display ICMP layer stats
udp       - display UDP layer stats
upcall    - trace received packets
tkstats   - tasking system status
dcstats   - DHCP Client statistics
users     - list all users
adduser   - add a new user

INET>```
To view the running tasks

INET> tkstats

tasking status: task wakeups: D

<table>
<thead>
<tr>
<th>name</th>
<th>state</th>
<th>stack</th>
<th>used</th>
<th>wakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>console</td>
<td>running</td>
<td>2048</td>
<td>556</td>
<td>153035</td>
</tr>
<tr>
<td>clock tick</td>
<td>sleeping</td>
<td>2048</td>
<td>204</td>
<td>117764</td>
</tr>
<tr>
<td>Main</td>
<td>blocked</td>
<td>2048</td>
<td>456</td>
<td>202</td>
</tr>
</tbody>
</table>

INET>

console = The serial console.
clock tick = RTOS/Stack timers.
Main = “Inet main” = The Stack task.
ColdFire Boot Up Sequence
ColdFire_Lite Boot

In file MCF5223_VECTORS.S
/*
 * Exception Vector Table
 */
VECTOR_TABLE:
_VECTOR_TABLE:
INITSP: .long ___SP_INIT/* Initial SP */
INITPC: .long start /* Initial PC */

start:
move.w #0x2700,sr
jmp _asm_startmeup

In file MCF5223_lo.s
_asm_startmeup:

.........
jsr mcf5223_init // mcf5223_sysinit.c
jsr cpu_startup // mcf5223.c

.........
/* Jump to the main process */
jsr main
ColdFire_Lite Boot  main.c

/* hardcode FEC IP address for now. We set it in netstatic, and
 * Ip startup code will initialize net[] from it.
 */
#if 1  // EMG 192.168.1.99
  netstatic[0].n_ipaddr = (0xC0A80163);
  netstatic[0].n_defgw  = (0x00000000);
  netstatic[0].snmask   = (0xffffff00);
#else  //jpw 192.168.2.3
  netstatic[0].n_ipaddr = (0xC0A80203);
  netstatic[0].n_defgw  = (0xC0A80201);
  netstatic[0].snmask   = (0xffffff00);
#endif
netstatic[0].mib.ifDescr = (u_char *)"Fast Ethernet Controller";

/* We set the station's Ethernet physical (MAC) address
 * from the address already in use by dBUG. This prevents
 * ARP problems on the development server. Production systems
 * usually read this from flash or eprom.
 */

#elif USE_FEC
  tmp = 0x00cf5223;
  mac_addr_fec[0] = (u_char)(tmp >> 24);
  mac_addr_fec[1] = (u_char)(tmp >> 16);
  mac_addr_fec[2] = (u_char)(tmp >>  8);
  mac_addr_fec[3] = (u_char)(tmp & 0xff);
  tmp = 0;
  mac_addr_fec[4] = (u_char)(tmp >> 24);
  mac_addr_fec[5] = (u_char)(tmp >> 16);
#endif
#if NPDEBUG
  dprintf("etheraddr = %02X:%02X:%02X:%02X:%02X:%02X

   mac_addr_fec[0], mac_addr_fec[1], mac_addr_fec[2],
   mac_addr_fec[3], mac_addr_fec[4], mac_addr_fec[5]);
#endif
#endif
// EMG - Override default buffer sizes to fit into Kirin2E

bigbufsiz = 1536 + 16;    // EMG
lilbufsiz = 200;          // EMG

/* Heap memory saving trick - reduce the time a TCP socket
 will linger in CLOSE_WAIT state. For systems with limited
 heap space and a busy web server, this makes a big difference.
*/
// EMG was 5     4/5/06 set to 1
TCPTV_MSL = 1;   /* set low max seg lifetime default */

#ifdef NPDEBUG
printf("Starting ints.\n");
#endif

//   mcf5xxx_irq_enable();           /* Let the interrupts fly... */
iniche_net_ready = TRUE;

    while( !uart_flush(0) ){};

#ifdef NPDEBUG
printf("Calling netmain()...\n");
#endif

netmain();    /* Start and run net tasks, no return. */
USE_ARG(err);
return 0;
}
ColdFire_Lite Boot

Main() calls netmain() in netmain.c which starts all the tasks.

```c
int
netmain(void)
{
    int i;
    int e;

    iniche_net_ready = FALSE;
    e = prep_modules(); ---------------------------------------------- in allports.c

    /* Create the threads for net, timer, and apps */
    for (i = 0; i < num_net_tasks; i++)
    {
        e = TK_NEWTASK(&nettasks[i]); ---------------------------------- Walk through the nettask[] array in netmain.c, "Inet main" MUST be first.
        if (e != 0)
        {
            dprintf("task create error\n");
            panic("netmain");
            return -1; /* compiler warnings */
        }
    }

    e = create_apptasks(); -------------------------------------------- Starts the FreeScale task, and console task
    if (e != 0)
    {
        dprintf("task create error\n");
        panic("netmain");
        return -1; /* compiler warnings */
    }

    uart_yield = 1;

    // MAIN_TASK_IS_NET
    tk_netmain(TK_NETMAINPARM); ---------------------------------------- Starts tk_netmain in netmain.c. This is the main network task.
    panic("net task return"); ---------------------------------------- The task never returns.
    return -1;
}
```
Network tasks

struct inet_taskinfo nettasks[] = {
#ifndef NO_INET_STACK

    &to_netmain,   /* netmain should always be first in this array */
    "Inet main",
    tk_netmain,
    NET_PRIORITY,
    NET_STACK_SIZE,
},
#endif /* NO_INET_STACK */
#ifndef NO_INET_TICK

    &to_nettick,
    "clock tick",
    tk_nettick,
    NET_PRIORITY,
    CLOCK_STACK_SIZE,
},
#endif /* NO_INET_TICK */
In allports.c

/* FUNCTION: tk_netmain()
 *
 * Main thread for starting the net. After startup, it settles into
 * a loop handling received packets. This loop sleeps until a packet
 * has been queued in rcvdq; at which time it should be awakened by the
 * driver which queued the packet.
 *
 * PARAM1: n/a
 *
 * RETURNS: n/a
 */

#ifndef NO_INET_STACK
TK_ENTRY(tk_netmain)
{
    netmain_init(); /* initialize all modules */

    iniche_net_ready = TRUE; /* let the other threads spin */

    for (;;) {
        TK_NETRX_BLOCK();
        netmain_wakes++; /* count wakeups */

        /* see if there's newly received network packets */
        if (rcvdq.q_len) {
            pktdemux();
            /* do not kill packet demux on net_system_exit. It may be
             * vital to a clean shutdown
             */
        }

        USE_ARG(parm); /* TK_ENTRY macro defines tk_netmain with 1 arg parm */
        TK_RETURN_UNREACHABLE();
    }
}

ColdFire_Lite Boot, the network task
FreeScale Task

The function `create_apptasks()` in `tk_misc.c` creates the FreeScale task used in all the future labs.

The FreeScale task then starts any additional tasks required for that particular folder.

```c
int
create_apptasks(void) In tk_misc.c
{
    int e = 0;

    #ifndef TFTP_PROJECT
    // EMG
    create_freescale_task();
    #endif

    #ifdef TK_STDIN_DEVICE
    e = TK_NEWTASK(&keyboardtask);
    if (e != 0)
    {
        dprintf("keyboardtask create error\n");
        panic("create_apptasks");
        return -1; /* compiler warnings */
    }
    #endif

    return 0;
}
```
The InterNiche RTOS
The Interniche stack also contains a simple RTOS.
- Non-preemptive – requires that a task give up control to the next task.
- Each task has its own stack (not superloop).
- A task is either sleeping based on time or an event.
- If a task is not sleeping, it is ready to run.

- You can add your own task via the tk_new() function.
- Your task MUST sleep to give up control to the next task in the list.

- There are no priorities, when task 1 gives up control, the RTOS tries to run task 2, and so on.

- Task 1 uses the system task, and MUST be the network task.
Tk_block() switches from one task to the next task in the ring.

Tk_new() adds a new task into the ring.

Tk_kill() removes the stack from the ring.

If a task is “sleeping”, it is skipped in the ring.
// entry points to tasking system

```c
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
// Init the RTOS
```

```c
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
// add a new task to the list
```
Creating a task

```
TK_OBJECT(to_keyboard); ------------------------------ in tk_ntask.h --------------- task *to_keyboard
TK_ENTRY(tk_keyboard); ----------------------------- in tk_ntask.h --------------- int  tk_keyboard( int parm )

e = TK_NEWTASK(&keyboardtask); ------------------- in osporttk.c ---------------- function adds keyboardtask descriptor
struct inet_taskinfo keyboardtask =
    {
        &to_keyboard,
        "console",
        tk_keyboard,
        NET_PRIORITY - 1,
        IO_STACK_SIZE,
    };

TK_ENTRY(tk_keyboard) ------------------------------ in tk_ntask.h --------------- int tk_keyboard( int parm )
{
    for (;;)
    {
        TK_SLEEP(1); /* make keyboard yield some time */
        kbdio(); /* let Iniche menu routines poll for char */
        keyboard_wakes++; /* count wakeups */

        if (net_system_exit)
            break;
    }
    TK_RETURN_OK();
}
```
Task Stack Protection

When a task is created, the stack space for that task in filled with a pattern ‘STAC’.

This pattern is used to indicate a fault if a stack overrun occurs. This is referred to as a guardband.

The tkstat command uses this pattern to determine the amount of stack used.

The guardband is checked at every task switch.
Task Stack Protection
For More Information on the RTOS

http://www.freertos.com

NicheTask

Welcome to the NicheTask™ open source site. This site is meant to be an embedded development resource center, providing all the information necessary to use the free NicheTask source code in your next embedded application.

Recognizing that many designs require only a clean, small and flexible tasking system, InterNiche Technologies has elected to contribute NicheTask to the embedded development community as open source software. The royalty-free "C" source code is being made freely available to all device developers to speed development of next generation appliances, consumer electronics and Internet connected devices.

Our goal in releasing NicheTask under an open source license is to make available a small, lightweight multitasking system for any embedded device, harness the pool of talented embedded developers, and further enhance NicheTask functionality by leveraging the imagination, development and debugging efforts of the NicheTask user community.

NicheTask is a round-robin tasking system that contains only control logic. The API has been designed so that the application can easily be mapped to more sophisticated embedded operating systems, such as μC/OS-II, without adding any overhead by simply calling the application's tasking calls to the RTOS calls. This means that you can start with NicheTask and upgrade or convert to a different RTOS or more complex architecture at a later time without having to re-work their protocol stack and application code.

Once a task gains control, it runs until it voluntarily blocks. The programmer has control over the length of time that a task is allowed to run so it will not lock the system. Tasks can be dynamically created, with each task having its own stack and control structure.
LAB: RTOS
In this LAB we will load a simple project consisting of multiple tasks blinking LED’s.

We will experiment with altering the sleep times of each task and observing the results.

We will also show the result of a task not giving up real-time.
Flash the ColdFire_Lite_RTOS target
Run
Observe the LED’s
Type tkstat at the inet prompt.
For this lab, 4 tasks are created. Each task blinks a LED at a different rate. The rate is controlled by the TK_SLEEP() in 5ms units.

Code in: freescaleRTOS.c

TK_ENTRY(tk_freescale4)
{
    int i;

    // Wait for TCP/IP stack to init
    while (!iniche_net_ready)
    TK_SLEEP(1);

    // Task's must not return, Infinite loops
    for (;;)
    {
        #if 1
            // Good
            TK_SLEEP(1400);
            printf( "\nTask4" );
        
        #else
            // Bad
            for( i=0; i<0xFFFF; i++ );
        #endif

        LED3_TOGGLE;

        if(net_system_exit)
        break;
    }

    TK_RETURN_OK();
}
What happens if a task does NOT sleep

1) Comment out a TK_SLEEP() in one of the tasks.
   Rebuild, flash, and run.

   Are all 4 tasks still running?
   Can you type anything at the console prompt?

2) Add a printf() to the task ( still no sleep ).
   Rebuild, flash, and run.

   What happens?
CodeWarrior 6.3 features
Issue when upgrading from 6.3 preview edition

Many customer had problems upgrading from the preview edition to the special edition of CodeWarrior 6.3.

This problem appeared primarily when customers also had CodeWarrior for S08 installed.

The issue is with the P&E drivers for the USB debugger.

Installing the new P&E drivers from the P&E site was the simplest method of resolving this issue.
Downloads

Find more Downloads

<table>
<thead>
<tr>
<th>Type</th>
<th>Name / Size / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Update</td>
<td><strong>Cyclone Max Stand Alone Software v7.20</strong> (18438 KB)</td>
</tr>
<tr>
<td></td>
<td>This application installs the latest Cyclone Max Software and Firmware upgrades. Simply</td>
</tr>
<tr>
<td></td>
<td>download and install this package to get the latest released software products for the</td>
</tr>
<tr>
<td></td>
<td>Cyclone Max.</td>
</tr>
<tr>
<td>Software Update</td>
<td><strong>P&amp;E Hardware Interface Drivers</strong> (8418 KB)</td>
</tr>
<tr>
<td></td>
<td>Installs P&amp;E drivers to allow P&amp;E applications to communicate with P&amp;E hardware via the</td>
</tr>
<tr>
<td></td>
<td>parallel port, PCI bus, Ethernet, Serial, and USB. This does not include the application</td>
</tr>
<tr>
<td></td>
<td>level support which comes with the different products.</td>
</tr>
</tbody>
</table>

© 2006 P&E Microcomputer Systems

Website Terms of Use and Sales Agreement
Issue building TCP/IP stack with 6.3 preview edition

Something changed with the release of CodeWarrior 6.3 causing many errors when building.

This issue was traced to common.h.

It has been fixed in this rev of the stack. For customers using the older version of the stack, make the following simple change.
Open common.h
Modify common.h

Set __CFM68K__ to 0
And __MC68K__ to 0

```c
/*
 * Include any toolchain specific header files
 */
#if (defined(__MWERKS__))
#include "build/mwerks/mwerks.h"
#define __CFM68K__ 0
#define __MC68K__ 0
#elif (defined(__DCC__))
#include "build/wrs/diab.h"
#elif (defined(__ghs__))
#include "build/ghs/ghs.h"
#endif
```
CodeWarrior 6.3 handles Interrupts differently

The ColdFire supports up to 8 interrupt levels, 0 to 7.

Normally this can be used to support prioritized nested interrupts.

CodeWarrior 6.3 disable this by default.
The normal method of writing an ISR

; 816: __declspec(interrupt)
; 817: void
; 818: fec_isr(void)
; 819: {
; 820:   BD * bdp;
; 821:   int i;
; 822: 

0x00000000                    _fec_isr:
;                             fec_isr:
0x00000000  0x40E746FC2700           strldsr #0x2700 <- Disables interrupts
0x00000006  0x4E560000               link     a6,#0
0x0000000A  0x4FEFFFC4               lea      -60(a7),a7
0x0000001E  0x48EF33FF000C           movem.l d0-d7/a0-a1/a4-a5,12(a7)
;
Overriding this “feature”

; 816: _declspec(interrupt:0)
; 817: void
; 818: fec_isr(void)
; 819: {
; 820: BD * bdp;
; 821: int i;
; 822:

0x00000000  _fec_isr:
;    fec_isr:
0x00000000  0x4E560000  link  a6,#0
0x00000004  0x4FEFFFC4  lea   -60(a7),a7
0x00000008  0x48EF33FF000C  movem.l  d0-d7/a0-a1/a4-a5,12(a7)
Forcing the interrupt level to another value

; 816: __declspec(interrupt:0x2200)
; 817: void
; 818: fec_isr(void)
; 819: {
; 820:   BD * bdp;
; 821:   int i;
; 822:
;
0x00000000                    _fec_isr:
;                         fec_isr:
0x00000000 0x40E746FC2200       strldsr #0x2200 <- SR = 0x2200
Why bother

Many customers require a low latency on servicing a specific interrupt.

Traditionally you would assign your “must do now” interrupt to the highest maskable level (6).

By disabling interrupts upon entering ANY isr (even a lower priority ISR) you get what’s referred to as a priority inversion. The lower priority interrupt holds off the higher priority interrupt.

Example: 802.15.4 MAC port

The 802.15.4 MAC interrupt MUST happen within 8us of the external IRQ signal.
The FEC ISR can run for over 100us.
The FEC ISR is set to a lower level then the MAC IRQ.
The Serial Port Driver
The serial driver

The serial driver is initialized in the function mcf5223_init() in the file mcf5223_sysinit.c.
Uart_init() is in the file iuart.c
The driver supports all three of the serial ports.

The define POLLED_UART controls the mode of the driver.

```c
#ifdef POLLED_UART
    • Puts the UART driver in polled mode.
#endif
```

```c
#ifndef POLLED_UART
    • Puts the UART driver in interrupt mode.
#endif
```
The serial driver parameters

In the file iuart.c

Here the serial RX buf is limited to 32 bytes, the TX buf is 256 bytes.

```c
#ifndef UART_RXBUFSIZE
#define UART_RXBUFSIZE 32 // EMG
#endif

#ifndef UART_TXBUFSIZE
#define UART_TXBUFSIZE 256 // EMG - I decreased this
#endif

#ifndef UART0_SPEED
#define UART0_SPEED 115200
#endif

#ifndef UART1_SPEED
#define UART1_SPEED 19200
#endif
```
The Serial Console Interface – type help at the INET> prompt

INET> help
SNMP Station: general commands:
  help    - help with menus
  state   - show current station setup
  delay   - set milliseconds to wait between pings
  host    - set default active IP host
  length  - set default ping packet length
  quit    - quit station program
  ping    - send a ping
  baud    - set serial console BAUD
  setup   - set interface IP address
  version - display version information
  !command - pass command to OS shell

Also try 'help [general|diagnostic|EMG HTTP]'
INET>
Type help diag at the INET> prompt

INET> help diag
SNMP Station: diagnostic commands:
arps    - display ARP stats and table
buffers  - display free q buffer stats
queues   - display packet buffer queues
dbytes   - display block of memory
debug   - set IP stack debug tracing
dtrap   - try to hook debugger
iface   - display net interface stats
linkstats - display link layer specific stats
tcp     - display TCP stats
sockets  - display socket list
tbconn   - tcp BSD connection stats
tbsend   - tcp BSD send stats
tbrcv    - tcp BSD receive stats
allocsize - set size for alloc() breakpoint
ipstat   - display IP layer stats
icmpstat - display ICMP layer stats
udp      - display UDP layer stats
upcall   - trace received packets
tkstats  - tasking system status
users    - list all users
adduser  - add a new user
INET>
HTTP Server Info

INET> help EMG
SNMP Station: EMG HTTP commands:
  - dir - Dir of EMG FFS
  - flash_erase - Erase the dynamic FLASH area
  - var - Dynamic HTML variable
  - http - Dump HTTP sessions array
INET> http

HTTP sessions array Dump

<table>
<thead>
<tr>
<th>STATE</th>
<th>VALID</th>
<th>KEEP_ALIVE</th>
<th>FILE_POINTER</th>
<th>SOCKET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for header</td>
<td>Not Valid</td>
<td>0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>Wait for header</td>
<td>Not Valid</td>
<td>0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>Wait for header</td>
<td>Not Valid</td>
<td>0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>Wait for header</td>
<td>Not Valid</td>
<td>0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

INET>
### Insight into the RTOS

**INET> tkstats**

<table>
<thead>
<tr>
<th>name</th>
<th>state</th>
<th>stack</th>
<th>used</th>
<th>wakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>console</td>
<td>running</td>
<td>2048</td>
<td>536</td>
<td>1216676</td>
</tr>
<tr>
<td>EMG HTTP server</td>
<td>ready</td>
<td>2048</td>
<td>192</td>
<td>51859563</td>
</tr>
<tr>
<td>clock tick</td>
<td>sleeping</td>
<td>2048</td>
<td>104</td>
<td>42047</td>
</tr>
<tr>
<td>Main</td>
<td>blocked</td>
<td>4096</td>
<td>392</td>
<td>0</td>
</tr>
</tbody>
</table>

**INET>**
Ethernet info – the iface command

INET> iface
Interface - Fast Ethernet
Status; Admin: up Oper: up for: 8 minutes, 45 sec.
rccd: errors: 0 dropped: 0 station: 0 bcast: 0 bytes: 0
sent: errors: 0 dropped: 0 station: 0 bcast: 0 bytes: 0
MAC address: 00 CF 52 00 00 ..R#. ..

Control Register = 3000

DATARATE = 100Mbps
ANE = Autonegotiation Enabled
DPLX = Half Duplexe

This register advertises the capabilities of the port to the MII
Status Register = 7849

Indicates the PHY supports 100BASE-TX full-duplex mode
Indicates the PHY supports 100BASE-TX half-duplex mode
Indicates the PHY supports 10BASE-T full-duplex mode
Indicates the PHY supports 10BASE-T half-duplex mode
No fault detected
PHY has auto-negotiation ability
valid link has NOT been established
AutoNegotiation NOT complete - Data is NOT Valid

Auto-Neg. Advertisement Register = 81E1

100BASE-TX full-duplex capable
100BASE-TX half-duplex capable
10BASE-T full-duplex capable
10BASE-T half-duplex capable

INET>
Adding a Your Own Command

//*****************************************************************************/
// Fill out structure for EMG FFS DIREctory menu command
//*****************************************************************************/
struct menu_op emg_ffs_dir_menu[] = {
    "EMG HTTP",    stooges, "EMG HTTP menu",
    "dir",    emg_ffs_dir, "Dir of EMG FFS",
    "flash_erase",    flash_erase, "Erase the dynamic FLASH area",
    "var",    emg_http_var, "Dynamic HTML variable",
    "http",    emg_http_sessions, "Dump HTTP sessions array",
    NULL,
};

// Install Menu item 'DIR' for EMG FFS
if( install_menu( emg_ffs_dir_menu ) )
    printf( "\nCould not install DIR menu item for EMG FFS" );
Commands are passed arguments

//*****************************************************************************
// int SoftEthernetNegotiation( int seconds ) Written By Eric Gregori
//
// Work-around for bug in hardware autonegotiation.
// Attempt to connect at 100Mbps - Half Duplexe
// Wait for seconds
// Attempt to connect at 10Mbps - Half Duplexe
//
// Returns 10, or 100 on success, 0 on failure
//*****************************************************************************

int set_baud(void * pio)
{
    char *cp;

    cp = nextarg(((GEN_IO)pio)->inbuf);
    iuart_set_baud( 0, atoi(cp) );

    return(0);
}
Printf is supported with formatting

/************************************************************
// Print Directory of Static and Dynamic Flash File Systems.
//
// Author: Eric Gregori (847) 651 - 1971
/************************************************************
int emg_ffs_dir(void *pio)
{
    int file_count, total_file_size, k, j;
    volatile unsigned long *fat_file_sys;
    volatile unsigned char *fat_file_names;

    ns_printf(pio, "%nStatic FFS ");
    ns_printf(pio, "\n\n%-32s %-6s %-8s",
        "FILENAME",
        "LENGTH",
        "POINTER" );

    total_file_size = 0;

    // Loop through each file printing the info
    for( file_count=0; file_count<emg_static_ffs_nof; file_count++ )
    {
        ns_printf(pio, "%n\n%-33s", emg_static_ffs_filenames[file_count] );
        ns_printf(pio, "%-9d", emg_static_ffs_len[file_count] );
        ns_printf(pio, "0x%-8x", (unsigned long)emg_static_ffs_ptrs[file_count] );
        total_file_size += emg_static_ffs_len[file_count];
    }

    ns_printf(pio,"\n\nTotal Size = %d",total_file_size);
    ns_printf(pio,"\nTotal static files = %d\n",file_count);

    ns_printf(pio, "\nDynamic FFS ");
    ns_printf(pio, "\n%-32s %-6s %-8s",
        "FILENAME",
        "LENGTH",
        "POINTER" );
}
This package is ideal for remote testing

Imagine this, you need a method to instrument a device you are testing.
Just write your own command, or better yet put your data in a VAR, and you can access that data from anywhere in the world.
This is a ideal platform for engineers to write small test programs, or build quick prototypes.
The MCF5223 has:

- 2 independent 4 channel 12 bit A/D converters
- 8 PWM modules
- 4 24 bit timers (can be used as pulse accumulators)
- 1 16 bit timer
- IIC, SPI, 3 UARTS, …..
TFTP
The Interniche stack includes a TFTP server and client.

TFTP requires a file system. Interniche also provides a RAM based virtual file system.

Since the 5223 only has 32K of RAM, the TFTP client and server can only be used for demo purposes.

The TFTP client/server have not yet been linked to the Flash File System.
TFTP project files

The TFTP project files are in the TFTP folder.

Tftpcli.c
  • TFTP client

Tftpsrv.c
  • TFTP server

Tftpudp.c
  • This module contains the low-level UDP routines.
The module m_udp.c contains the UDP API.
LAB: TFTP
In this lab we will build a project with the TFTP client.

We will use the client to connect to the PC

First you must disable 2 services on your computer.

Blackd – BlackIce

And

DefWatch

These processes interfere with the UDP traffic that TFTP uses.
Stop the DefWatch and Blackd processes on your machine. Double click on the TFTPDA32.exe

1) This opens the TFTP server on the PC.
2) I have included some small files.
   1) Test_file1.txt
   2) Test_file2.txt

At the INET> prompt type:
INET> vfsfilelist
total files = 0
dynamically allocated files = 0, buffer space = 0x0
INET>

Notice there are no files in the RAMdrive.
Additional commands with TFTP and VFS enabled

help - help with menus
state - show current station setup
delay - set milliseconds to wait between pings
host - set default active IP host
length - set default ping packet length
quit - quit station program
ping - send a ping
baud - set serial console BAUD
setup - set interface IP address
version - display version information
!command - pass command to OS shell

Also try 'help [general|diagnostic|vfs|tftpcl]

INET> help vfs

SNMP Station: vfs commands:
vfsfilelist - display vfs_file structure info
vfssetflag - set bit in vfs_file flags field
vfsclearflag - clear bit in vfs_file flags field
vfsopenlist - list currently open VFS files

INET> help tftp

SNMP Station: tftp commands:
tfget - tftp GET a file
tfput - tftp PUT a file
tfstate - Display tftp stats

INET>
Heap size = 26880 bytes
IP Address = C0A80163
Gateway = C0A80101
Mask = FFFFE00
etheraddr = 00:BA:DB:AD:01:02

Starting ints.
Calling netmain()...

InterNiche ColdFireLite TCP/IP for Coldfire, v3.0

Copyright 1997-2006 by InterNiche Technologies. All rights reserved.
Preparing device for networking
Ethernet started, Iface: 0, IP: 192.168.1.99
IP address of : 192.168.1.99
INET>
INET>
INET> tftpget 192.168.1.1 test_file1.txt
INET> tftp from 192.168.1.1 done; msg: Transferred 11 bytes in 0.0 second
us:ok(0)
After the file is uploaded

INET> vfsfilelist

```
test_file1.txt   -----WIDNS- 2000643C   B   B   100
total files = 1
dynamically allocated files = 1, buffer space = 0x100
INET>
```
DHCP
When DHCP is enabled, the TCP/IP stack cannot complete its initialization until after the DHCP transaction is complete.

The function netmain_init() in the module allports.c calls the function dhc_setup() in dhcsetup.c.

dhc_setup() runs the DHCP protocol which will contact the DHCP server to acquire a IP address and other network related data.
Enabling the DHCP client

In the file ipport.h you will find the following.

```c
#define INCLUDE_ARP     1  /* use Ethernet ARP */
#define FULL_ICMP       1  /* use all ICMP || ping only */
#define OMIT_IPV4       1  /* not IPV4, use with MINI_IP */
#define MINI_IP         1  /* Use Nichelite mini-IP layer */
#define MINI_TCP        1   /* Use Nichelite mini-TCP layer */
#define MINI_PING       1   /* Build Light Weight Ping App for Niche Lite */
#define BSDISH_RECV     1   /* Include a BSD recv()-like routine with mini_tcp */
#define BSDISH_SEND     1   /* Include a BSD send()-like routine with mini_tcp */
#define NB_CONNECT      1  /* support Non-Blocking connects (TCP, PPP, et all) */
#define MUTE_WARNS      1  /* gen extra code to suppress compiler warnings */
#define IN_MENUS        1  /* support for InterNiche menu system */
#define NET_STATS       1  /* include statistics printfs */
#define QUEUE_CHECKING  1  /* include code to check critical queues */
#define INICHE_TASKS    1  /* InterNiche multitasking system */
#define MEM_BLOCKS      1  /* list memory heap stats */
// EMG #define TFTP_CLIENT     1  /* include TFTP client code */
// EMG #define TFTP_SERVER     1  /* include TFTP server code */
// EMG #define DNS_CLIENT      1  /* include DNS client code */
#define INICHE_TIMERS   1  /* Provide Interval timers */

// EMG - To enable DHCP, uncomment the line below
//#define DHCP_CLIENT     1  /* include DHCP client code */

// EMG #define INCLUDE_NVPARMS 1  /* non-volatile (NV) parameters logic */
#define NPDEBUG         1  /* turn on debugging dprintf()s */
// EMG #define VFS_FILES       1  /* include Virtual File System */
// EMG #define USE_MEMDEV      1  /* Psuedo VFS files mem and null */
#define NATIVE_PRINTF   1   /* use target build environment's printf function */
#define NATIVE_SPRINTF  1   /* use target build environment's printf function */
#define PRINTF_STDARG   1   /* build ...printf() using stdarg.h */
#define TK_STDIN_DEVICE 1   /* Include stdin (uart) console code */
#define BLOCKING_APPS   1   /* applications block rather than poll */
#define INCLUDE_TCP     1  /* this link will include NetPort TCP w/MIB */

/**** end of option list ***/
```
Pushing SW1 at power-up will enable DHCP.
UDP stands for User Datagram Protocol
It is a layer under the TCP layer in the TCP/IP stack.

UDP does not include acknowledgements or connections.

UDP does support 0xFFF (65533) ports.

UDP is used whenever high speed data transfer is required.
UDP with ColdFire_Lite

Freescale_UDP_client.c contains an example of a UDP client.

This client sends packets as fast as possible to the PC.
LAB: UDP client
LAB: UDP client

1) Select the ColdFire_Lite_UDP_client target
2) Build, flash, run
3) Execute the UDP server on the PC by clicking on the BAT file.
Adjusting the packet size

Turning the POT on the demo board changes the packet size that the UDP client sends.

Notice the effect packet size has on data throughput.
LAB: UDP server
The freescale_UDP_server.c file contains a working and tested UDP server.

Unfortunately, the UDP client on the PC side is not working correctly.
LAB: TCP client
LAB: TCP client

Flash the ColdFire_Lite_TCP_client target
Double click the start_TCP_server.bat
What happens when the POT is adjusted?
LAB: TCP server
LAB: TCP server

With this LAB we will measure the maximum data rate that the ColdFireLite stack can accept a TCP data stream.

Flash the ColdFire_Lite_TCP_server target
Run
Using hyperterminal transfer a test file to the board.
Hyperterminal Configuration

![Hyperterminal Configuration Screen](image)

- **Host address**: 192.168.1.99
- **Port number**: 1234
- **Connect using**: TCP/IP (Winsock)
File transfer with hyperterminal

- Select "Send Text File..."
- Sends a text file to the remote system
Select any file (even binary) and transfer

Copyright 1997-2006 by InterNiche Technologies. All rights reserved.
Preparing device for networking
Ethernet started, Iface: 0, IP: 192.168.1.99
IP address of : 192.168.1.99
INET>
data rate 0 kbps
data rate 81 kbps
data rate 98 kbps
data rate 100 kbps
data rate 101 kbps
data rate 103 kbps
data rate 98 kbps
data rate 103 kbps
data rate 107 kbps
data rate 75 kbps
data rate 97 kbps
data rate 98 kbps
data rate 109 kbps
data rate 101 kbps
data rate 101 kbps
data rate 103 kbps
data rate 103 kbps
data rate 86 kbps
-data rate 104 kbps

Connected 1:124:51  ANSIW  115200 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
LAB: TCP serial server
Serial to Ethernet LAB

In this lab we will build a project with the Serial to Ethernet Firmware.

Flash the ColdFire_Lite_TCP_serial_server target
Run

Open 2 hyperterminal windows
  COM1
  TCP/IP  192.168.1.99  port 1234

Connect to the coldfire.

Anything typed into the TCP/IP hyperterminal appears in the serial hyperterminal, and vice-versa.
LAB:
TCP serial client
LAB: TCP serial client

For this lab, one person loads the ColdFire_Lite_TCP_serial_server target, and the other person loads the ColdFire_Lite_TCP_serial_client.

Connect the two boards via a crossover serial cable.

Each board is connected serially to a PC running hyperterminal.

When the client board is powered up, push and hold SW2.
Serial Flash Support
Serial Flash adds support for upto 4 meg of web pages

The serial flash is a SPI based device. It adds the ability to store up-to 4 meg worth of web pages. It also frees up the other 128K of internal flash for user code.

Do serial flash demo
ZigBee/802.15.4 + ColdFire®
Ethernet = A Winning Combination
Zigbee/802.15.4 Networking


IEEE 802.15.4 Topologies

- **PAN Coordinator**
- **Full Function Device (FFD)**
  - Any topology
  - Network coordinator capable
  - Talks to any other device
- **Reduced Function Device (RFD)**
  - Limited to being leaf devices
  - Cannot become a network coordinator
  - Talks only to a network coordinator
  - Very simple implementation
Network Pieces – PAN Coordinator

PAN Coordinator
- “owns” the network
  - Starts it
  - Allows other devices to join it
  - Provides binding and address-table services
  - Saves messages until they can be delivered
  - And more… could also have i/o capability
- A “full-function device” – FFD
- Mains powered
Network Pieces – End Device

End Device
- Communicates with a single device
- Does not own or start network
  - Scans to find a network to join
- Can be an FFD or RFD (reduced function device)
- Usually battery powered
TCP/IP stack merged with 802.15.4

MC13192 Radio

SPI

ATTN
IRQ
RESET
RXTXEN

802.15.4 MAC

MCF5223x

RTOS

HTTP

TCP/IP Stack

On-Chip Ethernet w/ PHY

PHY

Magnetics

RJ45

Cable

RJ45

Network
System Diagram

MC1321X Radio/Micro

802.15.4

MC13192 Radio

MC1321X Radio/Micro

The AJAX version of a Dial Guage

802.15.4 MAC

RTOS

HTTP

TCP/IP Stack

On-Chip Ethernet w/ PHY

PHY

SPI

ATTN
IRQ
RESET
RXTXEN
The sensors spend most of their time in hybernate mode. In hybernate mode, each sensor only draws 4µA. Each sensor wakes up every 5 seconds as a heartbeat, using the RTI. If the sensor detects a trigger, it wakes up immediately to send its data. Assuming less than one trigger every 5 seconds, each sensor should get a battery life of over 3 years using 2 AA’s. The coordinator is always powered up.
The web server provides an easy method of connecting external embedded systems over serial.
The external embedded system can send data to the web server using the VAR command.
The web server can send data over serial to the embedded system using forms.
This provides a simple mechanism for getting your embedded system on the web.
The ColdFire Lite Folder
The ColdFireLite folder contains the deliverables from Interniche. The Interniche stack supports:

- A simple RTOS
- IP protocol with ICMP and ARP
- UDP protocol
- TCP protocol
- A simple mini-socket API for TCP
- A DHCP client
- A DNS client
- A PING client
- A serial console with configurable menus
- A TFTP server and TFTP client
- A RAM based file system
Allports folder

- Allports.c - netmain_init()
- Timeouts.c- inet_timers() and check_interval_timers()
- Tk_misc.c - Contains the “console” task

These files are used in the boot process.
headers folder

- There are many files in this folder.
- We will concentrate on ipport.h and osport.h

osport.h

- Contains defines associated with the RTOS.
- Contains the standard stack sizes for application tasks.

Ipport.h

- Contains defines associated with the TCP/IP stack.
- Contains switches to enable/disable features of the TCP/IP stack.
Mcf_specific folder

Cksum.s
  • RFC1071 assembly language checksum routine.

Ifec.c
  • Fast Ethernet Controller driver.

Iutils.c
  • Low level serial routines

Tk_utils.s
  • Contains defines associated with the TCP/IP stack.
  • Contains switches to enable/disable features of the TCP/IP stack.
/*
 * Option macros to trade off features for size. Do not enable options
 * for modules you don't have or your link will get unresolved
 * externals.
 */

#define INCLUDE_ARP     1  /* use Ethernet ARP */
#define FULL_ICMP       1  /* use all ICMP || ping only */
#define OMIT_IPV4       1  /* not IPV4, use with MINI_IP */
#define MINI_IP         1   /* Use Nichelite mini-IP layer */
#define MINI_TCP        1   /* Use Nichelite mini-TCP layer */
#define MINI_PING       1   /* Build Light Weight Ping App for Niche Lite */
#define BSDISH_RECV     1   /* Include a BSD recv()-like routine with mini_tcp */
#define BSDISH_SEND     1   /* Include a BSD send()-like routine with mini_tcp */
#define NB_CONNECT      1  /* support Non-Blocking connects (TCP, PPP, et al) */
#define MUTE_WARNS      1  /* gen extra code to suppress compiler warnings */
#define IN_MENUS        1  /* support for InterNiche menu system */
#define NET_STATS       1  /* include statistics printfs */
#define QUEUE_CHECKING  1  /* include code to check critical queues */
#define INICHE_TASKS    1  /* InterNiche multitasking system */
#define MEM_BLOCKS      1  /* list memory heap stats */
#define DHCP_CLIENT     1  /* include DHCP client code */
#define INCLUDE_TCP     1  /* this link will include NetPort TCP w/MIB */
#define TFTP_CLIENT     1  /* Include TFTP client code */
#define TFTP_SERVER     1  /* Include TFTP server code */
#define DNS_CLIENT      1  /* Include DNS client code */
#define INICHE_TIMERS   1  /* Provide Interval timers */
#define VFS_FILES       1  /* include Virtual File System */
#define USE_MEMDEV      1  /* Psuedo VFS files mem and null */
#define NPDEBUG         1  /* turn on debugging dprintf()s */
#define NPDEBUG        1  /* turn on debugging dprintf()s */
#define NATIVE_PRINTF   1  /* use target build environment's printf function */
#define NATIVE_SPRINTF  1  /* use target build environment's printf function */
#define PRINTF_STDARG   1  /* build ...printf() using stdarg.h */
#define TK_STDIN_DEVICE 1  /* Include stdin (uart) console code */
#define BLOCKING_APPS   1  /* applications block rather than poll */
#define INCLUDE_NVPARMS 1  /* non-volatile (NV) parameters logic */

/* **** end of option list ****/
Open The directory Containing MAIN.C
Open main.c
HTTP/HTML/AJAX Overview

(And ColdFire TCP/IP Lite)
The **ColdFire_TCP/IP_Lite** stack includes:

A Mini-Sockets TCP API.

A TFTP (Trivial File Transfer protocol) server.

A DHCP (Dynamic Host Configuration protocol) client.

Zero-copy sockets for performance.

Less then 40K of program space.
Mini-Sockets TCP API

The mini-Sockets API is designed to be as close as possible to the BSD Sockets API and still allow a small footprint. The primary differences are that passive connections are accomplished with a single call, `m_listen()`, rather than the BSD `bind()-listen()-accept()` sequence, and the BSD `select()` call is replaced with a callback mechanism.

BSD = Berkeley Software Distribution
# Mini-Socket Interface Compared to BSD Sockets

<table>
<thead>
<tr>
<th>Mini-Sockets</th>
<th>BSD Sockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_socket()</td>
<td>socket()</td>
</tr>
<tr>
<td>m_connect()</td>
<td>connect()</td>
</tr>
<tr>
<td>m_recv() and/or m_send()</td>
<td>recv() and/or send()</td>
</tr>
<tr>
<td>- or -</td>
<td></td>
</tr>
<tr>
<td>tcp_send() and/or tcp_recv() - (zero-copy I/O)</td>
<td></td>
</tr>
<tr>
<td>m_close()</td>
<td>close();</td>
</tr>
</tbody>
</table>

For server applications:

<table>
<thead>
<tr>
<th>Mini-Sockets</th>
<th>BSD Sockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n/a - merged with listen)</td>
<td>socket()</td>
</tr>
<tr>
<td>(n/a - merged with listen)</td>
<td>bind()</td>
</tr>
<tr>
<td>m_listen()</td>
<td>listen()</td>
</tr>
<tr>
<td>(n/a - handled via callback)</td>
<td>accept()</td>
</tr>
<tr>
<td>m_recv() and/or m_send()</td>
<td>recv() and/or send()</td>
</tr>
<tr>
<td>- or -</td>
<td></td>
</tr>
<tr>
<td>tcp_send() and/or tcp_recv() - (zero-copy I/O)</td>
<td></td>
</tr>
<tr>
<td>m_close()</td>
<td>close();</td>
</tr>
</tbody>
</table>
A Simple Server Using Mini-Sockets

Creating a Listening Socket

// Init a socket structure with our Port Number
emg_http_sin.sin_addr.s_addr = (INADDR_ANY);
emg_http_sin.sin_port = (PORT_NUMBER);

emg_http_server_socket = m_listen(&emg_http_sin, freescale_http_cmdcb, &e);

Accepting a Connection

switch(code)
{
    // socket open complete
    case M_OPENOK:
        msring_add(&emg_http_msring, so);
        break;
}

Receiving TCP data

length = m_recv( freescale_http_sessions[session].socket, (char *)buf

Sending TCP data

bytes_sent = m_send( freescale_http_sessions[session].socket, data, length );

Closing the Socket

j = m_close( so );
A Simple Client Using Mini-Sockets

Creating a Socket
M_SOCK Socket = m_socket();

Connecting to a Server
int m_connect(M_SOCK socket, struct sockaddr_in * sin, M_CALLBACK(name));
// m_connect is blocking until a connection completes.
// If the socket is configured for non-blocking, then the callback function is used to indicate when the connection is established.

Receiving TCP data
length = m_recv( freescale_http_sessions[session].socket, (char *)buffer, RECV_BUFFER_SIZE );

Sending TCP data
bytes_sent = m_send( freescale_http_sessions[session].socket, data, length );

Closing the Socket
j = m_close( so );
HTTP1.0 compliant server with connection persistence and multiple sessions
(HTTP1.1 will be available in future revisions).
GET and POST elements supported.
Dynamic HTML support with replace and conditional tokens.
Serial interface support for Dynamic HTML variables.
Provides run time and compile time flash file systems.
Long file name support with subdirectories.
‘DIR’ command supported on serial interface.
PC utilities for compressing compile time and run time downloadable images of
multi-page web pages.
PC utility for downloading run time downloadable web page image through port 80
(to get through firewalls).
32 byte ascii key for web page download security.
It’s Free for use on ColdFire® processors!!!
### Software Model

<table>
<thead>
<tr>
<th>Freescale Web Server</th>
<th>Freescale Compile Time FFS</th>
<th>Freescale Run Time FFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ColdFire_TCP/IP_Lite</strong> RTOS and Console</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ColdFire_TCP/IP_Lite</strong> Mini-Socket TCP API</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ColdFire_TCP/IP_Lite</strong> TCP</td>
<td><strong>ColdFire_TCP/IP_Lite</strong> UDP</td>
<td><strong>ColdFire_TCP/IP_Lite</strong> ICMP</td>
</tr>
<tr>
<td><strong>ColdFire_TCP/IP_Lite</strong> IP layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ColdFire_TCP/IP_Lite</strong> FEC Driver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freescale Ethernet PHY</td>
<td>Freescale Hardware API</td>
<td></td>
</tr>
</tbody>
</table>

**FFS = Flash File System**
Web Servers

Web Servers implement the HyperText Transfer Protocol (HTTP) to send web pages from a server to a client. The Web Server contains the content, the Web Browser Displays the content. For these labs, the Web Browser used will be the Internet Explorer.
HTTP - An Overview

HTTP – HyperText Transport Protocol.
HTTP – Is used to transfer HTML/Web Pages on the web.

From RFC1945:

The HTTP protocol is based on a request/response paradigm. A client establishes a connection with a server and sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line, including the messages protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible body content.

Generally HTTP uses TCP/IP port 80.

There are two versions of HTTP, 1.0 and 1.1.

HTTP1.0 is defined by RFC1945.
HTTP Protocol Exchange

The client starts an exchange using one of two Methods:

GET method – Request the server to send a file
POST method – Sends a file to the server
  • The method is followed by a list of Request Header Fields

The server responds with a response message:
The first line of the message is the status line.
  • Sample Status line HTTP/1.0 200 OK
    ▪ Status code 2xx means success
    ▪ Status code 4xx means error

The status line is followed by a series of entity header fields separated by
        carriage return/line feeds.
HTTP Request / Response

HTTP Request

GET /filename.htm HTTP/1.1

HTTP Response

HTTP/1.1 200 OK
The Client (Browser) HTTP Request

GET /filename.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword
Accept-language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozzilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: www.msn.com
Connection: Keep-Alive

The above text is sent to the server on TCP/IP port 80
It asks the server to respond with the contents of filename.htm
It tells the server that it supports the HTTP1.1 standard
It tells the server that the client supports: gif, x-xbitmaps, jpeg, and pjpeg images
It tells the server that it supports msword documents
It tells the server that the language is English, and that the gzip and deflate decompression algorithm’s are available
It tells the server that the browser is running IE6.0 on a Windows machine
Finally it tells the server NOT to close the connection after the file is sent
Keep-Alive

By default, after the server sends the file to the client, it closes the TCP/IP connection.

The Keep-Alive request header field tells the server NOT to close the TCP/IP connection after the file contents are sent. This decreases the packet overhead for future connections.
The Server Response Header

HTTP/1.1 200 OK
Server: Microsoft-IIS/6.0
Cache-Control: no-cache
Content-Type: text/html
Content-Encoding: gzip
Content-Length: 9062

Followed by data from file, in this case encoded using gzip

The above data is returned by the server, to the client:
The HTTP/1.1 200 OK line tells the client/browser that HTTP1.1 is supported, and the 200 tells the client that the file was found
The Server line informs the client of the Web Server type and version
The Cache-Control line tells the client to disable cache
The Content-Type line tells the client the type of data that will follow
The Content-Encoding line tells the client that the following data is encrypted using gzip
The Content-Length line tells the client how many bytes are to follow
HTTP 1.1 is defined by RFC2616

Additions to HTTP 1.1:
Faster response, by allowing multiple transactions to take place over a single *persistent connection*.
Faster response and great bandwidth savings, by adding cache support.
Faster response for dynamically-generated pages, by supporting *chunked encoding*, which allows a response to be sent before its total length is known.
Efficient use of IP addresses, by allowing multiple domains to be served from a single IP address.
Ethereal HTTP demo
### Ethereal HTTP demo

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000000</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP 1.0 [SYN] Seq=0 Ack=1 Win=26144 Len=0</td>
</tr>
<tr>
<td>2</td>
<td>0.0006658</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>TCP</td>
<td>HTTP 1.0 [SYN, ACK] Seq=0 Ack=1 Win=65536 Len=0</td>
</tr>
<tr>
<td>3</td>
<td>0.006866</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=1 Ack=2 Win=5520 Len=0</td>
</tr>
<tr>
<td>4</td>
<td>0.007322</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=1 Ack=2 Win=5520 Len=0</td>
</tr>
<tr>
<td>5</td>
<td>0.007974</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=1 Ack=2 Win=5520 Len=0</td>
</tr>
<tr>
<td>6</td>
<td>0.09901</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>HTTP</td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>7</td>
<td>0.22860</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=12 Ack=1 Len=0</td>
</tr>
<tr>
<td>8</td>
<td>1.007491</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>GET /index.html HTTP/1.1</td>
</tr>
<tr>
<td>9</td>
<td>1.700772</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=12 Ack=957 Len=0</td>
</tr>
<tr>
<td>10</td>
<td>1.788965</td>
<td>192.168.1.99</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td>11</td>
<td>1.939498</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=957 Ack=1423 Len=0</td>
</tr>
<tr>
<td>12</td>
<td>5.72907</td>
<td>192.168.1.99</td>
<td>192.168.1.98</td>
<td>TCP</td>
<td>HTTP 1.0 [FIN, ACK] Seq=1423 Ack=957 Len=0</td>
</tr>
<tr>
<td>13</td>
<td>5.73000</td>
<td>192.168.1.98</td>
<td>192.168.1.99</td>
<td>TCP</td>
<td>HTTP 1.0 [ACK] Seq=1424 Ack=957 Len=0</td>
</tr>
</tbody>
</table>

Frame 6 (765 bytes on wire, 765 bytes captured)
- Ethernet II, Src: 00:cf:52:23:00:00 (00:0c:29:00:00:00), Dst: 00:0c:29:00:00:00 (00:0c:29:00:00:00)
- Transmission Control Protocol, Src Port: 80, Dst Port: 1600 (1600), Seq: 1, Ack: 479, Len: 711

HTTP Transfer Protocol
- HTTP/1.1 200 OK
- Server: EMG/1.1.0
- Cache-control: no-cache
- Data (765 bytes)
Customer Web Server Labs
Loading the Default Static Web Page

The purpose of this lab is to use CodeWarrior® to build and load the stack and default web page. The static file system utility will be used to change the default static web page. We will also learn how to configure the static IP address in both the demo board and the PC.
Using CodeWarrior® to Build the Default Web Page

Follow the instructions from the CodeWarrior lab to configure CodeWarrior and the flash programmer for the MCF5223x.
Load the MCP file
Set up PC Network Connection

Follow one of the following two methods:

- From Control Panel install new connect.
- Use existing connection.
Double click Network Connections
If available, Double click icon.

Otherwise, Double click New Connection icon And follow setup Wizard To create a LAN connection.
Now that a LAN connection is available

Let’s set it up for our needs

Click on Properties Tab
Set up PC Network Connection

The following properties dialog will open

Double Click on the Internet Protocol (TCP/IP) Icon

Checking this will aid config changes later
Set up PC Network Connection

Select
Use the following IP address
Set up PC Network Connection

Enter 192.168.1.1 for the IP address

Click in the Subnet Mask Field and it will auto-fill with 255.255.255.0

Click OK on all LAN setup dialog boxes and close them
Set up PC Network Connection

To return your LAN setting for normal Operation reopen the Internet Properties Dialog box and select Auto IP address.
The Taskbar

If there was a connection previously, an icon may be on the taskbar.

Configured LAN connections

If a connection bubble like shown at left is not shown (100bT is OK too)

And you have a 1gbit card reconfigure your card as shown in the next slides.
Setting Speed to 100Mb, Half for 1Gbit Cards

*****Only needed if communications issues with 1 Gbit card
From the Start menu select RUN
Enter “CMD”, click OK
A DOS window should open.
Enter -> ping 192.168.1.99

At the DOS prompt type ping 192.168.1.99 then hit enter
If Ping Does not work

Go to your hyperterminal window, hit enter a few times. Verify a INET> prompt appears.

Verify that you have a cross connect cable.

Verify that you have disabled VPN (on your personal machine)

Type `iface soft` at the INET> prompt.

Try Ping again after 2 seconds.
The Default Web Page

Open Internet Explorer, and type 192.168.1.99 (the IP address of the demo board) into the address bar. This is the default compile time web page you just loaded with the TCP/IP stack and Web Server.

Dynamic HTTP server with simple Flash File System

Written By Eric Gregori

The dynamic HTTP server with simple flash file system is a application written for the Interniche lite TCP/IP stack. This version also takes advantage of the Interniche Real Time Operating System. The server executes as a task, and supports multiple sessions. The server supports two separate file systems: the static file system uses a custom PC utility to convert web pages and binaries (jpeg, wav, swf, ... ) files into a C array for being compiled into the build. The dynamic file system uses a custom PC utility to convert the web pages and binaries into a compressed image. A custom PC download utility is then used to download the compressed image into the flash of the target system.
The Static/Compile Time Flash File System

The Static/Compile Time Flash File System allows the user to embed web pages consisting of one or multiple files into a target build.

The system has two parts: The firmware running in the ColdFire® processor as part of the Web Server, and the compression utility which is executed on the PC.

The Compression utility takes a list of files, and compresses them into a single ‘C’ file. The ‘C’ file is then compiled and linked into the final target build with the TCP/IP stack and the Web Server.
The compression utility: `emg_static_ffs.exe` is a DOS command utility that can be executed from Windows using a BATCH file.
Compression Utility Command Line Arguments

Emg_static_ffs filelist.txt output_file.c

Where:
Filelist.txt is a text file containing the list of files to compress. Each file should be on its own line, and the first file is the default. Comments can be added using a ‘*’ as the first character in a line.

Output_file.c is the file generated containing all the files in the filelist compressed together, along with data structures used to reference the files from the Web Server.
Sample filelist.txt

* emg static web page description file
* The files listed below will be concatenated into a
  * single C compatible file.

readme.htm
CFCORESEMBLEM.gif

The last line must be a blank line with just a CRLF
(just hit enter in the last blank line).
The output file contains the contents of each file stored as a ‘C’ array. The files inserted are from the filelist.txt file (see previous slides).

Array containing list of filenames

Array containing list of pointers to files.

Array containing file sizes

Array containing file type

Number of files
Other Uses for the Static/Compile Time Flash File System

User Data can also be stored in the static system. The data can be binary or text, but name the file *.txt. The utility actually treats all files as binary files.

The user can access the data from the firmware using examples in the firmware.

This feature can be useful in the static/Compile Time System, but is considerably more useful in the run time loadable system.
We are going to edit a HTML file.
Build a Compressed ‘C’ image.
Copy the Image to our project.
Re-build the project.
Load the new image in flash.
ColdFire_Lite Compile_Time_Loaded_Web_Page_Example

The Compile_Time_Loaded_Web_Page_Example

This is the directory for the static web page demo/lab.

ColdFire_Lite\src\projects\example
Opening the HTML File
HTML or HyperText Markup Language is the language used to describe web pages. HTML is a ascii text based language that defines how text and images are placed on a page. HTML is a ascii text based language that uses “tags” to instruct a web browser how text and images are placed on a page.
Tags start with a ‘<‘ and end with a ‘>‘.
Most tags have a open and close form.
The open form <HTML>
The close form </HTML>
Tag form: <TAG ATTRIBUTE=value>
Tags/attributes are used to define placement, color, style, and fonts for text.
Tags are also used to define position and size for a image.
<HTML>
<HEAD>
<TITLE>This text will appear at the top of the web browser, the navigation bar</TITLE>
</HEAD>
<BODY>
<CENTER>Hello World</CENTER>
</BODY>
</HTML>

The HTML element is used to tell the web browser that we are using HTML instead of JavaScript, or some other language.

The HEAD element contains meta-information. Meta-information is not part of the body of the document but defines the document in a general sense. The Title of the web page is a good example. It is not displayed in the body of the web page, but on the navigation bar of the web browser.

The BODY element defines the displayed portion of the web page.
Some Interesting HTML Tags

<CENTER> Centers the object on the page.

<Hx> Heading Size x, where x is from 1-6.

<P> Start or paragraph.

<FONT COLOR=RED> Sets font color to red.

<FONT SIZE=x> Sets font size, where x is from 1-?.

<A HREF="freescale.com"> Makes text a URL pointing to freescale.com.

<IMG SRC="filename.jpg"> Puts the image filename.jpg into the web page.

<IMG SRC=filename.jpg" ALIGN=center> Loads the image filename.jpg and centers it in the page.

<TABLE> Creates a table with the help of <TR> table row and <TD> table data.
Creating Web Pages

Using notepad, you can start writing HTML immediately, and build your own Web Page.

Or, you can use an HTML generator.

- These programs allow you to design a web page, and generate the HTML for you.
- Just search for “HTML generator” on the web.
- There are dozens of them, some free.
Using Microsoft Word

Microsoft Word can also be used to generate a Web Page

By saving a document as *.htm in Microsoft Word, Word will create a web page.

• The web pages created by Word tend to be very large.
• Also, Word creates a subdirectory for images.
• Be sure to change the image reference paths to remove the directories.

The web page for this lab (readme.htm) was generated in Word.
To Edit the HTML open the readme.HTM file in Notepad

The first few lines of the readme.htm file

<html>
<head>
<title>Dynamic HTTP server with simple Flash File System</title>

Modify the Dynamic HTTP server … String with something else

<html>
<head>
<title>This is really cool</title>

Save the new file
Build a New Output File

First Double click the batch file make.bat to build the image.
Build the project by clicking on the MAKE icon (circled in RED)
Start the Flash Programmer

Start the Flash Programmer by selecting the tools Flash Programmer Pull Down
Erase Flash by selecting Erase/Blank Check, and clicking the Erase button. Watch the Status window for errors.
Program

After the Erase is Complete, go to the Program/Verify window and click on the Program button.
Click on the Run icon, circled in RED below. This will execute the code in flash. If you have an external power supply, you could also disconnect the USB from the board and hit reset.
Open Internet Explorer, and type 192.168.1.99 (the IP address of the demo board) into the address bar. This is the default compile time web page you just loaded with the TCP/IP stack and Web Server.
Run Time Loadable Web Pages

Web Pages can be uploaded via Ethernet at run time. Web Pages can be loaded over and over again. # of re-loads only limited by # of writes to flash.

Loaded Web Pages take priority over default or Compile Time Web Pages.

Loaded Web Pages are protected with a 32 character password string.
Serial Flash Support

The firmware supports a SPI based external serial flash. Serial flash parts are available in 1 Mbyte and 4 Mbyte sizes. When serial flash is enabled, all 256K of on-board flash is available for user firmware.
Build and Loading a Run Time Loadable Web Page

A single Batch file is used to both build and load the Web Page. Within the Batch file are calls to two executable.

The first executable: emg_dynamic_ffs.exe

Compresses the Web Pages into a binary, and adds a File Allocation Table (FAT) to the top of the file. The firmware in the Web Server uses the FAT to reference the data in the file from within the binary image.
Emg_dynamic_FFS.exe

Emg_dynamic_ffs filelist.txt output_file.ffs

Where:
Filelist.txt is a text file containing the list of files to compress. Each file should be on its own line, and the first file is the default. Comments can be added using a ‘*’ as the first character in a line.

Output_file.ffs is the file generated containing all the files in the filelist compressed together, along with File Allocation Table used to reference the files from the Web Server.
Emg_web_uploader ip_address filename.ffs key_string

Where:
Ip_address is the ip address of the hardware (192.168.1.99) in examples.
Filename.ffs is the file generated by the emg_dynamic_ffs utility.
Key_string is the 32 character key used to unlock the flash file system (joshua) in examples.
The filelist.txt file lists the files that will be included in the FFS. Dynamic.ffs is the binary image containing all the files and the FAT. Pause is a DOS command to prompt the user to hit any key. 192.168.1.99 is the IP address of the hardware for these examples. Joshua is the key string for these examples.
Let take a look at the contents in the directory of the demo board.

Notice, the static file system (compile time) still contains files.

When the dynamic (run time) file system is loaded with a binary image, it takes priority over the static file system.

Other files in the static FFS are still available.

```
INET> dir
Static FFS

FILENAME                              LENGTH   POINTER
readme.htm                            22129     0x1465A
CFCOREENSEMBLEM.gif                   12919     0x19CCC
vardump.htm                           1279      0x1CF44

Total static files = 3
Total Size = 36327

Dynamic FFS

FILENAME                              LENGTH   POINTER
readme.htm                            34541     0x20028

Total dynamic files = 1
Total Size = 34541
```

Web Server Defaults

Notice what we entered at the address bar. No filename is specified. When no filename is specified the Web Server defaults to the first file listed in the file system.

* emg dynamic web page description file
* The files listed below will be concatenated into a single compressed downloadable image.
* The first file in the list is the default file

Readme.htm.htm ← This is the file that is loaded by default.
CFCORESEMBLEM.gif
vardump.htm
Going Direct to a File Using the Browser

To go directly to a file in the FFS from the browser, just include the name of the file after the ‘/’ in the IP address.

Notice Vardump.htm is in the static file system, but is still available after loading a dynamic FFS.
The WEB Server Firmware - Processing a FORM Submit

The Web Server detects the form by the ‘?’ in the filename. The FORM is then parsed into the two parts, the NAME and the VALUE.

The NAME is on the left of the ‘=’ sign, the VALUE on the right. The Name is used to call the function “LED”, and pass it the VALUE.*
Dynamic HTML Tokens

Dynamic HTML Tokens allow variable content like sensor data to be inserted into web pages, no programming required. Just insert the token ~IIF; into your HTML, and the token will be replaced with the data referenced by II. Conditional tokens take the idea one step further, by allowing whole HTML strings to be replaced based a data comparison to a constant.
The REPLACE Token ~IIF;

Where:

II = The decimal variable index to read the data.

The variable array contains 32 longwords (can be as high as 99).

F = The format to display the data (D = Decimal, H = Hex).

Example:

```html
<HTML>
<HEAD>
<TITLE>This text will appear at the top of the web browser, the navigation bar</TITLE>
</HEAD>
<BODY>
<CENTER>You have opened this page ~02D;

The Variable index 02 is the web page hit counter.
</CENTER>
</BODY>
</HTML>
```
The CONDITIONAL Token `^II>C|true|false`; 

Where:

II = The decimal variable index to read the data.
    The variable array contains 32 longwords (can be as high as 99)
C = Hex value for comparison.
> = Variable value greater than C
= = Variable value equal to C
& = Variable value and C
! = Variable not equal to C

“true” = ascii string to replace if condition is true
“false” = ascii string to replace if condition is false
<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>On Board Switch</td>
</tr>
<tr>
<td>Web Page Hit</td>
</tr>
<tr>
<td>Analog Channel 0</td>
</tr>
<tr>
<td>Analog Channel 1</td>
</tr>
<tr>
<td>Analog Channel 2</td>
</tr>
<tr>
<td>Analog Channel 3</td>
</tr>
<tr>
<td>Analog Channel 4</td>
</tr>
<tr>
<td>Analog Channel 5</td>
</tr>
<tr>
<td>Analog Channel 6</td>
</tr>
<tr>
<td>Analog Channel 7</td>
</tr>
<tr>
<td>RTC - Hour</td>
</tr>
<tr>
<td>RTC - Min</td>
</tr>
<tr>
<td>RTC - Sec</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
<tr>
<td>Available to user</td>
</tr>
</tbody>
</table>
Notice the “Available To User” entries in the variable array. You can modify the ‘C’ code for the Web Server to assign any 32 bit value you want to a available position in the variable array. Or, you can use the serial interface to modify the variable in the array.

The serial interface method is designed for interfacing to other embedded systems. The serial port supports autobaud, so it will automatically sync to the baud of your embedded device.
Using the Serial Interface- The ‘VAR’ command

INET> var

Dynamic HTML variable dump
Variable 0 = 12345678   BC614E
Variable 1 = 0   0
Variable 2 = 1035   40B
Variable 3 = 2202   89A
Variable 4 = 2205   89D
Variable 5 = 0   0
Variable 6 = 0   0
Variable 7 = 2435   983
Variable 8 = 387   183
Variable 9 = 3125   C35
Variable 10 = 0   0
Variable 11 = 23   17
Variable 12 = 26   1A
Variable 13 = 56   38
Variable 14 = 99   63

INET>
VAR Command Parameters

var – Dumps the contents of the array to the serial port.
Var 14 – Dumps the contents of variable index 14.
Var 14, 12345678 – Assigns 12345678 decimal to variable index 14.
The ‘var x’ Command

INET> var 0
Variable 0 = 12345678   BC614E

INET> var 2
Variable 2 = 1195   4AB

INET> var 3
Variable 3 = 2202   89A

INET> var 4
Variable 4 = 2275   8E3

INET>
Assigning a Variable with the ‘var’ Command var ll, decimal_data

INET> var 14, 100

INET> var 14
Variable 14 = 100  64

INET> var 14, 250

INET> var 14
Variable 14 = 250  FA

INET> var 14, 900

INET> var 14
Variable 14 = 900  384

INET>
How to Use the VAR Command

The Zigbee Coordinator collects data from its sensors, then converts it into ‘VAR’ commands. Each sensor is given a separate variable index.

The ‘VAR’ command is terminated with a CR, the INET> prompt provides software handshaking.
### The HTML Code

```html
<html>
<head>
<meta http-equiv="refresh" content="1">
<title>All Variables</title>
</head>
<body>
<table>
<tbody>
<tr><td>Variable</td><td>HEX</td><td>DECIMAL</td><td></td></tr>
<tr><td>00</td><td>~00H</td><td>~00D</td><td>Not Used</td></tr>
<tr><td>01</td><td>~01H</td><td>~01D</td><td>On Board Switch Status ^01&0001|SW1||^01&0008|SW2||</td></tr>
<tr><td>02</td><td>~02H</td><td>~02D</td><td>Web Page Hit Counter</td></tr>
<tr><td>03</td><td>~03H</td><td>~03D</td><td><font color="^03">0800|"RED"|"BLUE"|</font> Analog Channel 0 (pot)</td></tr>
<tr><td>04</td><td>~04H</td><td>~04D</td><td>Analog Channel 1 (lite)</td></tr>
<tr><td>05</td><td>~05H</td><td>~05D</td><td>Analog Channel 2 (NU)</td></tr>
<tr><td>06</td><td>~06H</td><td>~06D</td><td>Analog Channel 3 (NU)</td></tr>
<tr><td>07</td><td>~07H</td><td>~07D</td><td>Analog Channel 4 (acc-x)</td></tr>
<tr><td>08</td><td>~08H</td><td>~08D</td><td>Analog Channel 5 (acc-y)</td></tr>
<tr><td>09</td><td>~09H</td><td>~09D</td><td>Analog Channel 6 (acc-z)</td></tr>
<tr><td>10</td><td>~10H</td><td>~10D</td><td>Analog Channel 7 (NU)</td></tr>
<tr><td>11</td><td>~11H</td><td>~11D</td><td>RTC - Hour</td></tr>
<tr><td>12</td><td>~12H</td><td>~12D</td><td>RTC - Min</td></tr>
<tr><td>13</td><td>~13H</td><td>~13D</td><td>RTC - Sec</td></tr>
<tr><td>14</td><td>~14H</td><td>~14D</td><td></td></tr>
<tr><td>15</td><td>~15H</td><td>~15D</td><td></td></tr>
<tr><td>16</td><td>~16H</td><td>~16D</td><td></td></tr>
<tr><td>17</td><td>~17H</td><td>~17D</td><td></td></tr>
<tr><td>18</td><td>~18H</td><td>~18D</td><td></td></tr>
<tr><td>19</td><td>~19H</td><td>~19D</td><td></td></tr>
<tr><td>20</td><td>~20H</td><td>~20D</td><td></td></tr>
<tr><td>21</td><td>~21H</td><td>~21D</td><td></td></tr>
<tr><td>22</td><td>~22H</td><td>~22D</td><td></td></tr>
<tr><td>23</td><td>~23H</td><td>~23D</td><td></td></tr>
<tr><td>24</td><td>~24H</td><td>~24D</td><td></td></tr>
<tr><td>25</td><td>~25H</td><td>~25D</td><td></td></tr>
<tr><td>26</td><td>~26H</td><td>~26D</td><td></td></tr>
<tr><td>27</td><td>~27H</td><td>~27D</td><td></td></tr>
<tr><td>28</td><td>~28H</td><td>~28D</td><td></td></tr>
<tr><td>29</td><td>~29H</td><td>~29D</td><td></td></tr>
<tr><td>30</td><td>~30H</td><td>~30D</td><td></td></tr>
<tr><td>31</td><td>~31H</td><td>~31D</td><td></td></tr>
</tbody>
</table>
</body>
</html>
```
POT > 0800 = false

<table>
<thead>
<tr>
<th>Variable</th>
<th>HEX</th>
<th>DECIMAL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>BC61E</td>
<td>12345678</td>
<td>Not Used</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>On Board Switch Status</td>
</tr>
<tr>
<td>02</td>
<td>679</td>
<td>1657</td>
<td>Web Page Hit Counter</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
<td>Analog Channel 0 (pot)</td>
</tr>
<tr>
<td>04</td>
<td>45D</td>
<td>1117</td>
<td>Analog Channel 1 (lite)</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>Analog Channel 2 (NU)</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
<td>Analog Channel 3 (NU)</td>
</tr>
<tr>
<td>07</td>
<td>95F</td>
<td>2399</td>
<td>Analog Channel 4 (acc-x)</td>
</tr>
<tr>
<td>08</td>
<td>18A</td>
<td>394</td>
<td>Analog Channel 5 (acc-y)</td>
</tr>
<tr>
<td>09</td>
<td>C33</td>
<td>3123</td>
<td>Analog Channel 6 (acc-z)</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>Analog Channel 7 (NU)</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>23</td>
<td>RTC - Hour</td>
</tr>
<tr>
<td>12</td>
<td>2F</td>
<td>47</td>
<td>RTC - Min</td>
</tr>
<tr>
<td>13</td>
<td>24</td>
<td>36</td>
<td>RTC - Sec</td>
</tr>
<tr>
<td>14</td>
<td>384</td>
<td>900</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Image of a computer screen showing variable information]
POT > 0800 = true
SW1 Pushed

- Variable HEX: BC614E 12345678
- Variable Decimal: Not Used

- Variable HEX: 01
- Variable Decimal: On Board Switch Status SW1

- Variable HEX: 02
- Variable Decimal: Web Page Hit Counter

- Variable HEX: 03
- Variable Decimal: Analog Channel 0 (pot)

- Variable HEX: 04
- Variable Decimal: Analog Channel 1 (mA)

- Variable HEX: 05
- Variable Decimal: Analog Channel 2 (NLI)

- Variable HEX: 06
- Variable Decimal: Analog Channel 3 (NLI)

- Variable HEX: 07
- Variable Decimal: Analog Channel 4 (acc-x)

- Variable HEX: 08
- Variable Decimal: Analog Channel 5 (acc-y)

- Variable HEX: 09
- Variable Decimal: Analog Channel 6 (acc-z)

- Variable HEX: 10
- Variable Decimal: Analog Channel 7 (NLI)

- Variable HEX: 11
- Variable Decimal: RTC - Hour

- Variable HEX: 12
- Variable Decimal: RTC - Min

- Variable HEX: 13
- Variable Decimal: RTC - Sec
Browser Update

Notice how the last lab updated itself in the browser

The `<meta http-equiv="refresh" content="1">` HTML tag causes the page to automatically reload.
The “1” is the number of seconds to wait before reloading the page.

This is the old method of automatically updating a web page. Notice its not very efficient, the whole page is reloaded even though only a few values change.
Notice the page flickers.

These limitations are addressed in WEB2.0.
Web 2.0 generally refers to a second generation of services available on the World Wide Web that gives users an experience closer to a desktop application than the traditional static web pages. The traditional world wide web was designed to present static information. Web 2.0 is designed to be interactive.
AJAX - A Key Component of Web 2.0

AJAX – Asynchronous Javascript And XML
AJAX is not a technology in itself, but a term that refers to the use of a group of technologies together.
AJAX is a Web development technique for creating interactive web applications.
AJAX uses Javascript, the Document Object Model (DOM), and the XMLHttpRequest object to exchange data asynchronously with the web server and display dynamic data in a smooth manner.
Javascript is a prototype-based scripting language with a syntax loosely based on ‘C’.

Javascript is embedded as ascii source in web pages. The web browser interprets the Javascript within the <HTML> tags. Since the browser actually runs the Javascript, all the web server has to do is serve it up.

Including Javascript in your web pages is easy.
Simple “Hello World” in Javascript

```html
<html>
<head>
<title>Simple Javascript</title>
</head>
<script language="JavaScript">
document.write(“Hello World”);
</script>
</html>
```
Document Object Model (DOM)

Javascript would be relatively useless if it could not alter the web page.
Of course, Javascript can alter the web page using the DOM.
The DOM makes everything on a web page a object accessible by Javascript.
Javascript accesses the object using the object ID.
Remember the marquee in the web page from the last lab

\(<\text{marquee width="800" scrollamount=8}>\text{Time Since Last Reset: } \sim 11\text{D};\sim 12\text{D};\sim 13\text{D};\</\text{marquee}>\)

We modify it slightly by adding the id element

\(<\text{marquee id="scroller" width="800" scrollamount=8}>\text{Time Since Last Reset: } \sim 11\text{D};\sim 12\text{D};\sim 13\text{D};\</\text{marquee}>\)

Now, we can alter the marquee from Javascript.
Javascript Runs in the Background

The time in the web page automatically updates.
The time is actually being read from the **ColdFire®** evaluation board Real Time Clock.
Javascript uses the `XMLHttpRequest` function to request data from the web server, without effecting the viewable page.
Internet Explorer has an issue terminating Javascript.

Between the Javascript labs, you should close and re-open Internet Explorer.
LAB 9: Accelerameter Example

• Goto the LAB9_??????? Directory.
• Double Click the make.bat to load the LAB into the ColdFire.

The 52233DEMO board has a 3-axis accelerameter. This device outputs 3 analog voltages representing the x, y, and z planes.

The ColdFire has 2 separate 4 channel 12 bit A/D converters.

3 channels are used here to read the X, y, and z planes, then the A/D values are stored in VAR array locations 7, 8, and 9.
LAB 9: Accelerometer Example

Move your board in free sp
LAB 10: Monitoring Analog Data

• Goto the LAB10_??????? Directory.
• Double Click the make.bat to load the LAB into the ColdFire.
LAB 10: Monitoring Analog Data

Real-Time Analog Data Presented Bargraph Style Using AJAX

108.3 205.5 237.1 302.7
HTML and Javascript

Notice the image has been given an id of bargraph

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Freescale MCF5223x</title>
</head>

<body>

<IMG SRC="avtlogo.gif" id="bargraph" BORDER=0 WIDTH=549 HEIGHT=470 >

</body>
```
The Javascript assigns the height of the bargraph object to the pot_value/10

<script language="JavaScript">
// Javascript for 5223EVB demo Written by Eric Gregori
// The script communicates with the board using a AJAX technique.

// Variables global to script
var pot_value

// Parse input file
function parse_vars( data )
{
    var parsed = data.split( "\n" );
    pot_value = parsed[0]
    bargraph.height = pot_value/10
}
HTML and Javascript

The Javascript request the data from the server using `http_request.open('GET', url, true);`

```javascript
// Request input file
function makeRequest(url)
{
    var http_request = false;
    if (window.XMLHttpRequest)
    { // Mozilla, Safari, ...
        http_request = new XMLHttpRequest();
        if (http_request.overrideMimeType)
        {
            http_request.overrideMimeType('text/xml');
        }
    }
    else if (window.ActiveXObject)
    { // IE
        try
        {
            http_request = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
            http_request = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
    if (!http_request)
    {
        alert("Giving up :( Cannot create an XMLHttpRequest");
        return false;
    }
    http_request.onreadystatechange = function() { alertContents(http_request); }
    http_request.open('GET', url, true);
    http_request.send(null);
}
```
The javascript request the data from the server by requesting the file pot_data.txt. This request is done every 200ms (setTimeout).

```
function alertContents(http_request)
{
    if (http_request.readyState == 4)
    {
        if (http_request.status == 200)
        {
            parse_vars(http_request.responseText);
        }
        else
        {
            // alert('There was a problem with the request.');
            alert( http_request.status );
        }
    }
}

function loop()
{
    makeRequest("pot_data.txt");
    setTimeout("loop()",200);
}

window.onload=loop;
```
LAB 11: Back to Real Work

AJAX can be used for more than fun and games.
In an embedded environment sometimes it would be nice to present real-time changing data in a graphic manner.
Go to the ajax_graph_demo directory.
Close the web browser (internet explorer).
Double click the make.bat file.
Open Internet Explorer, and type 192.168.1.99 in the address bar.
Build and Load ajax_graph_demo
AJAX In Action on a MCF5223x
LAB 12: Monitoring Analog Data with a dial guage

• Goto the LAB12_??????? Directory.
• Double Click the make.bat to load the LAB into the ColdFire.
Turn the POT, and move the board around.

The AJAX version of a Dial Guage
LAB 13: Accessing files in the FFS

• Goto the LAB13_??????? Directory.
• Double Click the make.bat to load the LAB into the ColdFire.
• Go through the presentation

The Powerpoint presentation has been converted to HTML and Javascript. The presentation is being served up by the ColdFire.
The FFS has a User API for user applications to access the flash file system.
The FFS can be used to store any type of data, binary or ascii. The user can store accel tables, nv parameters, configuration info, … The information can be accessed by the firmware with a simple open call. The user can update the information by doing a runtime file load.
Emg_open

//*****************************************************************************
// int emg_open( char *filename, uint32 *data_pointer, uint32 *file_size )
//
// User API to dynamic flash file system
//
// Finds the file descriptor in the FAT.
// Sets data_pointer to start of data.
// Sets file_size to size of file in bytes.
// returns a < 0 if error, 0 = success
//
// for an example of using emg_open(), see cat command in menulib.c
//
// Author: Eric Gregori  (847) 651 - 1971
// eric.gregori@freescale.com
//*****************************************************************************/
The CAT command

The CAT command is an example of how to use the emg_open() function.

The CAT command will dump the contents of a file to the console.
The CAT command code

```c
int cat(void *pio)
{
    char *cp;
    char *data;
    uint32 bytes;
    uint32 index;
    uint32 i, bad_char;

    ns_printf(pio, "\n\n");
    cp = nextarg(((GEN_IO*)pio)->sbuf);
    if (each_open(cp, &index, &bytes) == 0)
    {
        date = (char *)index,
        index = 0;
        bad_char = 0;
        while((index < bytes) && !bad_char)
        {
            for (i = 0; i < 10; i++)
            {
                if (data[index] < 0 || (data[index] > 127))
                {
                    bad_char = 1;
                    break;
                }
                ns_printf(pio, "%c", data[index]);
                index++;
                if (index == bytes)
                    break;
            }
            tk_sleep(2);
        }
        else
            ns_printf(pio, "File Not Found");
        if (bad_char)
```
LAB 14: Try to load a image > 128K

• Goto the LAB14_?????? Directory.
• Double Click the make.bat to load the LAB into the ColdFire.

The load will fail, because the image is too big.
Verify that the original dynamic FFS contents have not been corrupted.
FAQs

How many web pages can be loaded into the Run Time or Compile Time FFS?
- 255 files in each for a total of 510

What is the MAX size of a Run Time Web Page image?
- 128K, Limited only by the size of a flash logical block.

What is the MAX size of a Compile Time Web Page Image?
- Whatever FLASH is left over from the TCP/IP stack and Web Server Firmware minus the Run Time FFS area(128K) = Currently about 64K.

Is the Run Time Loadable Web Page verified after downloading?
- Yes and no. Handshaking is used to verify that all the pakets were transferred correctly. No, because there currently is no verify that flash got written correctly. There are hooks already in the code to do this, and I plan on releasing a update with these changes soon.

How quickly can AJAX poll the server for information?
- That depends on the connection, and the web browser. With a small closed network, and Internet Explorer 6.0, the update rate can be as high as 100ms.
Reference Material
Firmware Overview
NicheLite Documentation can be found in the project.
The HTTP Server Task

//*****************************************************************************
// Declare Task Object
//*****************************************************************************
TK_OBJECT(to_emghttpsrv);
TK_ENTRY(tk_emghttpsrv);
struct inet_taskinfo emg_http_task = {
    &to_emghttpsrv,
    "EMG HTTP server",
    tk_emghttpsrv,
    NET_PRIORITY,
    APP_STACK_SIZE
};
long emghttpsrv_wakes = 0;

TK_ENTRY(tk_emghttpsrv)
{
    int err;
    while (!iniche_net_ready)
        TK_SLEEP(1);
    err = freescale_http_init();
    if (err == SUCCESS)
    {
        exit_hook(freescale_http_cleanup);
    }
    else
    {
        dtrap();    // emghttp_init() shouldn't ever fail
    }
    for (;;)
    {
        freescale_http_check();
        tk_yield();    // will block on select
        // give up CPU in case it didn't block
        emghttpsrv_wakes++;
        if (net_system_exit)
            break;
    }
    TK_RETURN_OK();
}
Questions, Answers and Consultations