<u>ColdFire Ethernet</u>
By Eric Gregori

1.0: Embedded Ethernet

Ethernet has become a reality for low cost embedded systems. The Ethernet standard (ieee 802.3) was originally designed for networking computers over Local Area Networks. The standard has become so popular that it is currently hard to find a PC or laptop without a Ethernet port. The ieee802.3 specification (Ethernet) defines a mechanical/electrical connection between devices (physical layer), and a multi node addressable communications protocol (MAC layer).

1.1: The Ethernet Physical Layer

The Ethernet physical layer defines the physical connections between nodes. The 802.3 standard defines many physical layers including everything from coaxial cable to fiber optics. An through the years, the common choice has changed drastically from thick multi stranded cables with large connectors (called thicknet) to the small RJ-45, 8 pin connector we are use to today.

The common modern copper physical layer is referred to as 100Base-TX. This copper based twisted pair medium contains 8 wires grouped in 4 twisted pairs. 2 twisted pairs are used for communications in each direction. The cable is referred to as a category 5 (cat 5 for short). The category 5 standard defines a cable consisting of 4 twisted pairs capable of carrying frequencies up to 100mhz .

The modern Ethernet jack

Most modern building and residents are wired using Category 5 cables for their PC networks and broadband. Making this an ideal medium for distributed processing/sensing in a building or residential environment.

1.2: The Ethernet MAC layer

The MAC (Media Access Control) protocol layer defines the communication that occurs over the physical layer. Ethernet is a multi node protocol , so each node has a unique address. This address is defined in the MAC layer. For Ethernet, MAC addresses are 48 bits long (6 bytes or octets). The devices MAC address never changes, usually it is programmed at the factory. The MAC address must be unique, so MAC addresses are managed/distributed by the ieee.

The MAC address, along with various other fields is contained in the Ethernet MAC header. As the name implies, the header sits in front of the Ethernet packet. It contains the MAC address of the source node, along with the MAC address of the destination node, and a type field.

Preamble	Destination Address	Source Address	Frame Type	Frame User Data	FCS Checksum
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46 – 1500 Byte	4 Bytes

Ethernet MAC header

Ethernet can be used directly without any additional layers. It provides a simple point to point communication mechanism, with some error checking (FCS checksum). Ethernet by itself does not provide the higher levels of communications robustness we have become accustomed to. Additional layers are required to add features such as: multiple ports, packet re-transmission, packet timeouts and connections. These additional layers are defined by the 7 layer OSI model.

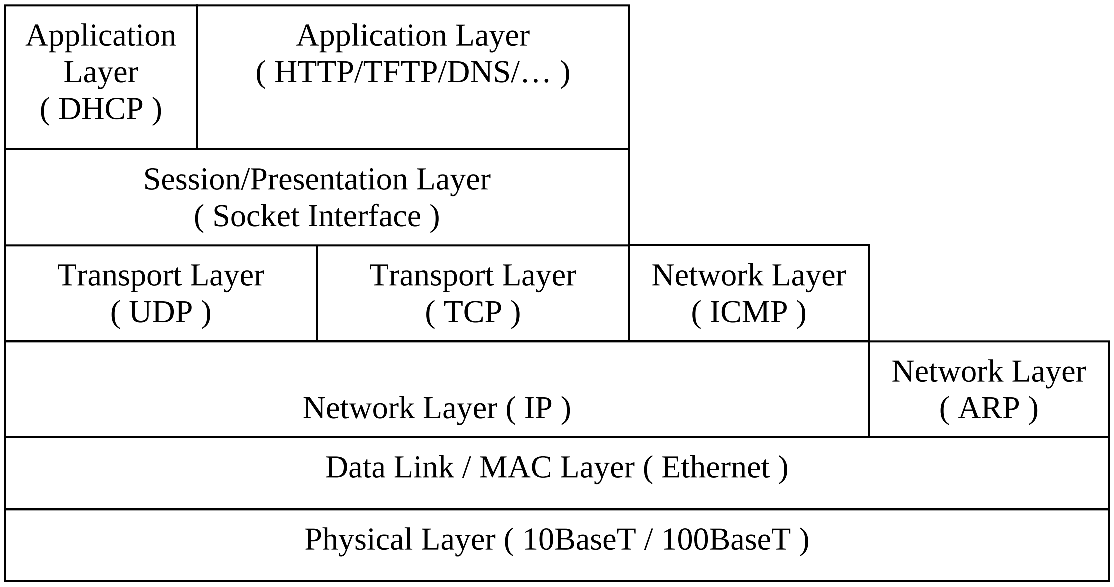
1.3: 7 Layer OSI model

The 7 layer OSI model defines the functions of the various layers in a communication stack. The lowest layers (Physical and MAC / Data Link layers) are traditionally implemented in hardware. The 5 layers above the MAC / DDL are usually implemented in software.

The network or IP layer (for a TCP/IP stack) provides an additional layer of addressing (the IP addresses we are use to xx.xx.xx.xx) and multiplexing. Multiplexing splits a single communications channel into multiple time divided communications channels (ports in TCP/IP lingo).

The transport layer adds the most critical feature to the communications stack. The transport layer is the tcp in tcp/ip. This layer is responsible for creating a virtual connection between 2 logical points (not nodes). The logical points are referred to as sockets. The sockets API is actually defined by the session layer.

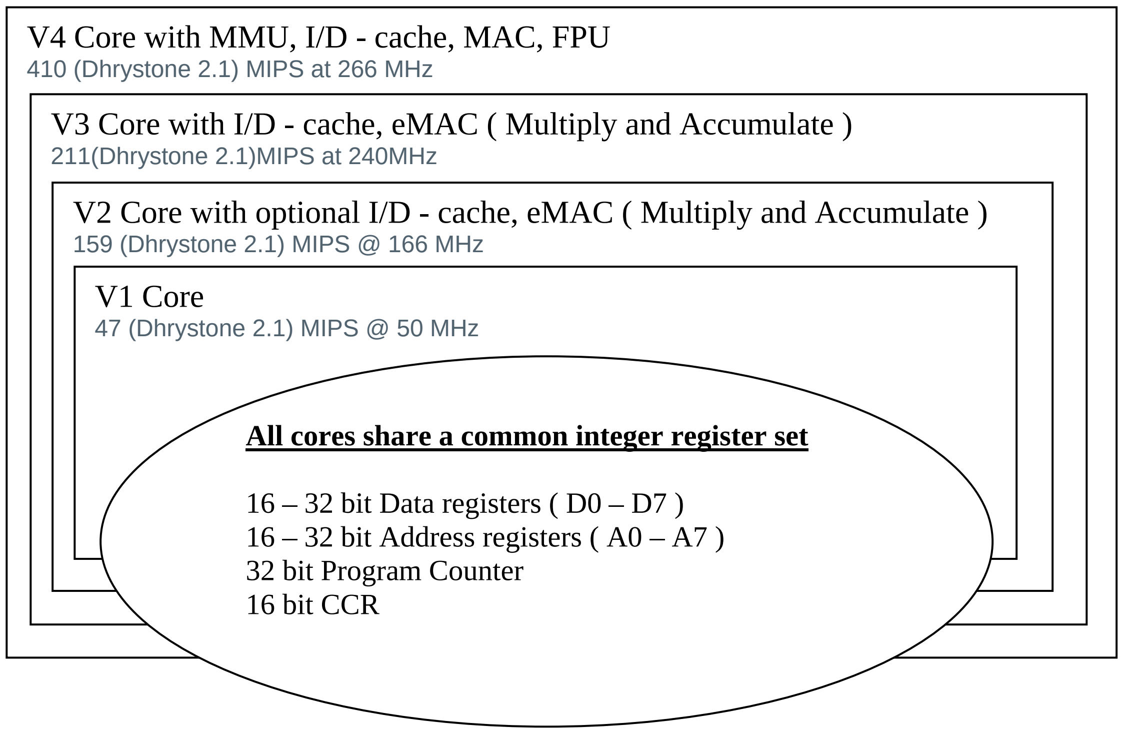
Finally, the application layer. This application defines the common protocols used on the internet; http, smtp, tftp. This layer can also be used for custom protocols.


Application Layer (DHCP)
Application Layer (HTTP/TFTP/DNS/…)
Session/Presentation Layer (Socket Interface)
Transport Layer (UDP)
Transport Layer (TCP)
Network Layer (ICMP)
Network Layer (IP)
Network Layer (ARP)
Data Link / MAC Layer (Ethernet)
Physical Layer (10BaseT / 100BaseT)

7 Layer OSI model

2.0: The ColdFire family of Microcontroller

The ColdFire family of Microcontrollers are based on the 32 bit ColdFire cores. The ColdFire core is available in 4 varieties, each a superset of the core below it. The cores are completely scalable, with the differences being additional instructions or add on modules (MMU for instance). The V1 core contains the base register and instruction set. The V2 core adds to the V1 core additional instructions and addressing modes, along with a optional eMAC (Multiply and Accumulate unit).


V4 Core with MMU, I/D - cache, MAC, FPU
410 (Dhrystone 2.1) MIPS at 266 MHz
V3 Core with I/D - cache, eMAC (Multiply and Accumulate)
211(Dhrystone 2.1)MIPS at 240MHz
V2 Core with optional I/D - cache, eMAC (Multiply and Accumulate)
159 (Dhrystone 2.1) MIPS @ 166 MHz
V1 Core
47 (Dhrystone 2.1) MIPS @ 50 MHz
All cores share a common integer register set
16 – 32 bit Data registers (D0 – D7)
16 – 32 bit Address registers (A0 – A7)
32 bit Program Counter
16 bit CCR

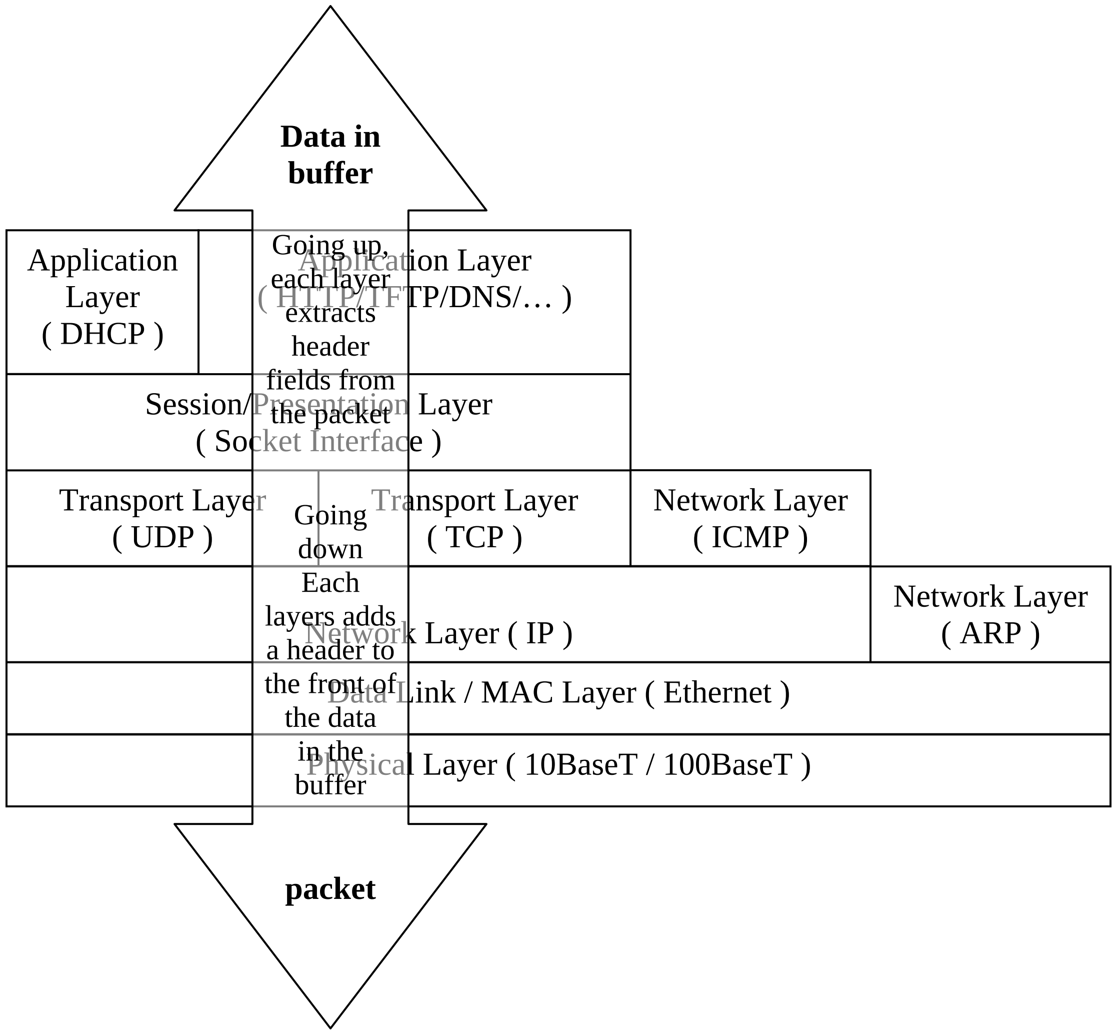
The ColdFire cores have scalable instruction sets, features, and performance

2.1: The advantage of a 32 bit architecture

The true 32 bit architecture of ColdFire Microcontrollers lends itself well to efficient communication stack data movement. In a communication stack such as TCP/IP the packet comes in at the bottom of the stack, and propogates up. Data to send, starts at the top of the stack as a buffer, and works it’s way to the bottom of the stack to be sent out as a packet.

Movement up and down the stack is an area of inheritant inneficinecy in a communications stack.   To improve efficiency higher performance stacks use pointers instead of copying the data multiple times (this is sometimes referred to as zero copy). Pointer arithmetic is significantly more efficient with a 32 bit core using true 32 bit registers.

This is a big advantage for the ColdFire 32 bit architecture. In addition, advanced addressing modes with offset capability can make extracting data from individual fields in a header a single instruction operation.


Data in buffer
Application Layer (DHCP)
Going up, each layer extracts header fields from the packet
Application Layer (HTTP/TFTP/DNS/…)
Session/Presentation Layer (Socket Interface)
Transport Layer (UDP)
Going down Each layers adds a header to the front of the data in the buffer
Transport Layer (TCP)
Network Layer (ICMP)
Network Layer (IP)
Network Layer (ARP)
Data Link / MAC Layer (Ethernet)
Physical Layer (10BaseT / 100BaseT)
packet

3.0: The ColdFire Fast Ethernet Controller (FEC)

The FEC module is the ColdFire interface to the Ethernet world. The FEC module is consistant from the highest performance V4 core based part, all the way down to the V1 core. This consistency means that drivers written for one Ethernet enabled ColdFire processor will work on any Ethernet enabled ColdFire processor (memory allocation would be the biggest difference).

The FEC module is high performance Ethernet engine with a very rich heritage. The FEC module started out in the MPC860T. This high performance Power Architecture based part quickly became a power house in the Ethernet world, going into high performance routers and telephone equipment. The MPC860T was so popular in the Ethernet trade that if you make a call today, chances are that somewhere along the route the voice data of your call will pass through a MPC860T. The MPC860T came out in the mid 1990's. The FEC module has been tested and improved upon for over 10 years in some of the highest performance Ethernet environments. The FEC module from the MPC860 is now in the ColdFire line of processors.

3.1: ColdFire FEC features

- The Ethernet Media Access Controller (MAC) is designed to support 10 and 100 Mbps Ethernet/IEEE 802.3 networks.

- IEEE 802.3 full duplex flow control

- Support for full-duplex operation (200 Mbps throughput)

- Retransmission from transmit FIFO following a collision (no processor bus utilization)

- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)

- Address recognition
  — Frames with broadcast address may be always accepted or always rejected
  — Exact match for single 48-bit individual (unicast) address
  — Hash (64-bit hash) check of individual (unicast) addresses
  — Hash (64-bit hash) check of group (multicast) addresses
  — Promiscuous mode

- Dedicated DMA controller to allow for packet transmission and reception with no processor overhead.

The Fast Ethernet Controller supports both 100Mbps and 10Mbps, allowing the ColdFire to interface to both old (10BaseT networks) and newer (100BaseTX) networks (with the appropriate 10/100 PHY). Full duplex operation means that packets can be sent and received at the same time (remember the PHY uses separate wires for TX and RX). At 100Mbps this translates to a maximum 200Mbps throughoutput.

The hardware performs all the functions of the 802.3 Ethernet MAC layer without software intervention. The software simply initializes the FEC, writes the nodes MAC address into the MAC address register, and initializes the RX and TX buffer rings. The FEC will automatically receive, process, and verify (via CRC) incoming packets, and DMA the packet into a RX buffer. For TX, the FEC is triggers by software, then automatically DMA's the packet from the TX buffer, calculates a CRC, serializes the packet, and sends it out to the PHY. If a collision is detected, the FEC will perform a random backoff, and retry, without processor intervention. After the packet is transmitted, the FEC reports a status.

The advantage of this high level of functional integration into the FEC is reduced software overhead. The software simply has to create the packet and give it to the FEC for transmission via the TX ring buffer. On the receive side, the software simply has to take the packet from the RX ring buffer. The ring buffers are managed by the FEC hardware.

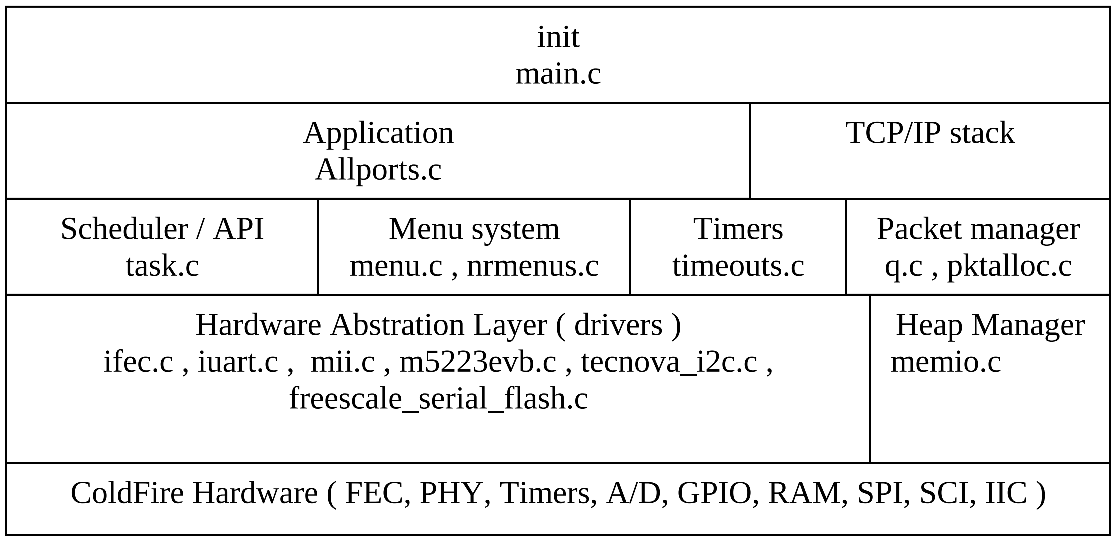
4.0: ColdFire TCP/IP stack

TCP/IP (Transmission Control Protocol / Internet Protocol) is the communication protocol of the internet. The acronym is derived from 2 layers of the communication stack, TCP and IP. The term TCP/IP actually describes multiple protocols within both the stack. Each protocol is defined by a RFC (Request For Comment).

Proper TCP/IP stack operation requires multi-tasking. The ColdFire TCP/IP stack is integrated with a simple operating system. This simple round-robin OS can also be used by the application code. The OC is non-preemptive, but does provide 2 modes of operation (single stack (superloop) and multi-stack). Additional OS features include a interactive real-time upgradeable menu system, user timers, and heap memory management.

4.1: ColdFire TCP/UDP/IP Stack features

HTTP (HyperText Transport Protocol), Serial to Ethernet, TFTP (Trivial File Transfer)
Mini IP Application's Interface
DHCP (Dynamic Host Configuration Protocol) or manual IP configuration, DNS
TCP (Transmission Control Protocl), UDP (User Datagram Protocol)
ICMP (Internet Control Messaging Protocol), BOOTP (BOOTstrap Protocol)
ARP (Address Resolution Protocol), IP (Internet Protocol)

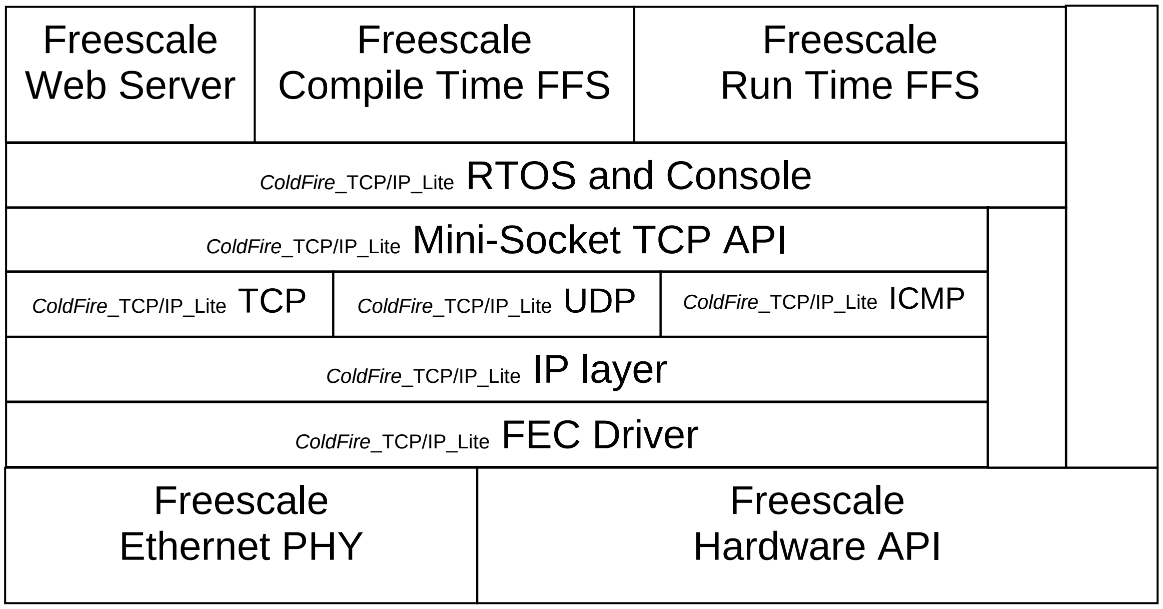

init
main.c
Application
Allports.c
TCP/IP stack
Scheduler / API
task.c
Menu system
menu.c , nrmenus.c
Timers
timeouts.c
Packet manager
q.c , pktalloc.c
Hardware Abstration Layer (drivers)
ifec.c , iuart.c , mii.c , m5223evb.c , tecnova_i2c.c , freescale_serial_flash.c
Heap Manager
memio.c
ColdFire Hardware (FEC, PHY, Timers, A/D, GPIO, RAM, SPI, SCI, IIC)

ColdFire TCP/IP Stack and RTOS

The TCP/IP stack implements the protocols described in the following RFC's (please refer to http://www.rfc-editor.org/rfcxx00.html for details);

RFC791	Internet protocol (IP)
RFC792	Internet Control Message protocol (ICMP)
RFC768	User Datagram Protocol (UDP)
RFC793	Transmission Control Protocol (TCP)
RFC826	Ethernet Address Resolution Protocol (ARP)
RFC1035	Domain Name Server (DNS)
RFC2131	Dynamic Host Configuration Protocol
RFC2132	DHCP options

The Session / Presentation layer is a mini-socket interface similar the familiar BSD socket interface. The stack has been optimized for embedded application using zero-copy functionality for minimum RAM usage.


Freescale Web Server
Freescale Compile Time FFS
Freescale Run Time FFS
ColdFire_TCP/IP_Lite RTOS and Console
ColdFire_TCP/IP_Lite Mini-Socket TCP API
ColdFire_TCP/IP_Lite TCP
ColdFire_TCP/IP_Lite UDP
ColdFire_TCP/IP_Lite ICMP
ColdFire_TCP/IP_Lite IP layer
ColdFire_TCP/IP_Lite FEC Driver
Freescale Ethernet PHY
Freescale Hardware API

FFS = Flash File

4.2: DHCP client

The Dynamic Host Configuration Protocol is used to aquire network parameters at runtime. The DHCP protocol is define in RFC2131 and RFC2132. The stack runs a DHCP client which searches for a DHCP server (this is referred to as discovery).

Packets are transferred using the UDP layer, and BOOTP ports (67 and 68). Since the IP stack does not have a IP address yet, discovery is done using strictly broadcast addresses. Included in the discovery packet is a unique transaction ID (xid).

A listening DHCP server sends a offer message containing the xid sent by the client and the suggested network parmaeters, again using broadcast addressing. Also encoded in the offer is a unique server ID. The client will use this server ID when sending a request packet back to the server indicating that it accepts the network parameters that were offered. Finally the server ACKS the client using it's new IP address.

4.3: DNS Client

The DNS client is used to communicate with the DNS (Domain Name Server). The purpose of the DNS system is to translate Domain Names into IP addresses. The DNS protocol is described in RFC1035. DNS can use UDP or TCP, with port 53. The DNS protocol is stateless, all the information is contained in a single message. This message is fully documented in RFC1035.

5.0: Available examples and Appnotes

Appnotes are available at www.freescale.com

5.1: HTTP web server and flash file system

The HTTP web server and flash file system is described in detail in appnote:

AN3455 – ColdFire Lite HTTP Server

- HTTP1.0 compliant server with connection persistance and multiple sessions (HTTP1.1 will be available in future revisions).
- Supports multiple HTTP connections.
- Includes a Flash File System supporting both ColdFire internal flash and external SPI flash.
- Web pages can be updated in flash over Ethernet, or built in at compile time.
- Supports the HTTP GET method, with a simple mechanism for adding other methods.
- 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.

5.2: UDP/TCP clients and servers example source code

The ColdFire Lite stack project includes almost a dozen built in usage examples. These examples are designed to highlite various features in the stack, and demonstrate how to use them. The TCP/IP stack and RTOS, along with all the sample applications listed below are discussed in appnote AN3470.

Code examples include;
ColdFire_Lite
Barebones TCP/IP stack

ColdFire_Lite_RTOS
How to use the RTOS application

ColdFire_Lite_TFTP
TFTP server application

ColdFire_Lite_UDP_client
UDP client application for UDP performance testing

ColdFire_Lite_UDP_server
UDP server application for UDP performance testing

ColdFire_Lite_TCP_client
TCP client application for TCP performance testing

ColdFire_Lite_TCP_server
TCP server application for TCP performance testing

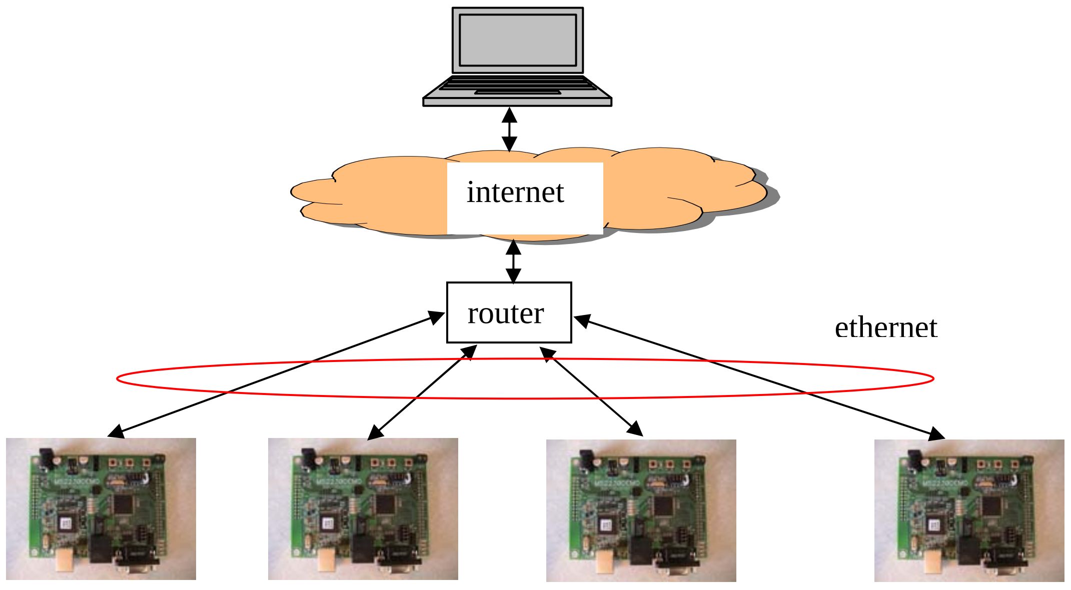
ColdFire_Lite_TCP_serial_client
TCP to serial / serial to TCP client

ColdFire_Lite_TCP_serial_server
TCP to serial / serial to TCP server

ColdFire_Lite_TCP_with_Web_Server
Web (HTTP) server with dynamic HTML

5.3: ColdFire_Lite_TCP_alarm

The alarm demo application includes both PC side and ColdFire side firmware. This code is an example of a remote sensor application. Where a remote sensor periodically sends data over TCP to a host server.



Remote Sensors connect with host over the internet to report and receive data.

5.4: HTTP Client Firmware

The HTTP client provides the ability to read web pages and XML data from the internet using a ColdFire processor. The HTTP client uses the DHCP client to automatically aquire a IP address and other TCP/IP information includeing the IP addresses of any DNS server. Then the HTTP client uses the DNS client to translate any user provided URL's to real IP addresses.

The HTTP client uses the GET method to request a page from the server. Along with the GET request is the HTTP header. The HTTP header is hardcoded in the HTTP client via constant strings declared in the file emg_http_client.c.

5.5: The WGET command – An example of using the HTTP Client

The wget command is a command often found in Linux distributions that transfers files using the HTTP protocol. The wget command is a console based HTTP client. Using the menuing system provided by the ColdFire TCP/IP stack (and explained in the AN3470 appnote) and the HTTP client, wget functionality can be added to the ColdFire TCP/IP stack.

5.6: RSS/XML character data filter

To extract the character data (the information you actually want to read) from the RSS stream, all the Meta-Text must be filtered out. Any HTML that can be processed must be translated and processed. For instance, a <br> is HTML for a new line. This would appear as &lt;br&gt; in the RSS stream. The filter must correctly translate &lt;br&gt; into a carriage return and line feed. Other HTML tags that are routinely embedded in character data include paragraph tags <p> and image tags <IMG … >. In the stream these tags appear as &lt;p&gt; and &lt;IMG … &gt; repectively. The paragraph tab can be translated to a carriage return and line feed, but the image tag must be ignored unless the embedded system can process images.

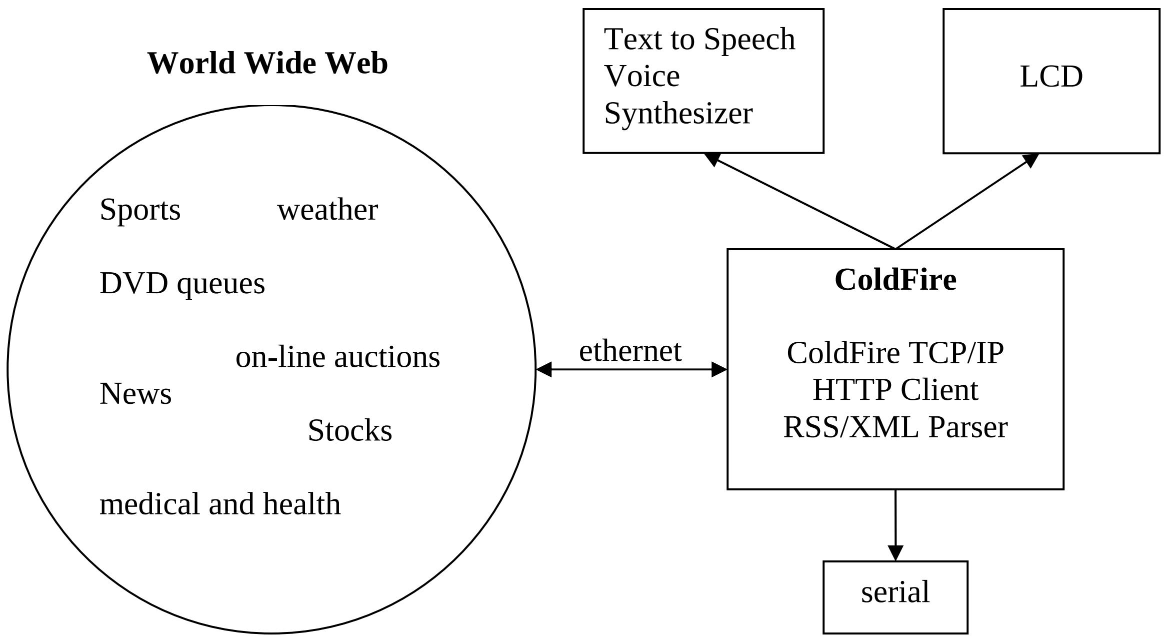
The filter takes in a XML or RSS data stream, and a list of tags. It outputs the character data only from the selected tags. The tag list is a array of pointer to the tag strings the filter should be filtering character data from.

Normally the filter returns 0. When the filter processes the '>' in a tag that is in the list, the filter returns the index into the filter array for the tag that it found plus 1. Example: after detecting a <title> tag in a RSS or XML stream the filter will return 1. After detecting a <description> tag in a stream the filter will return 2. Normally the filter returns 0.

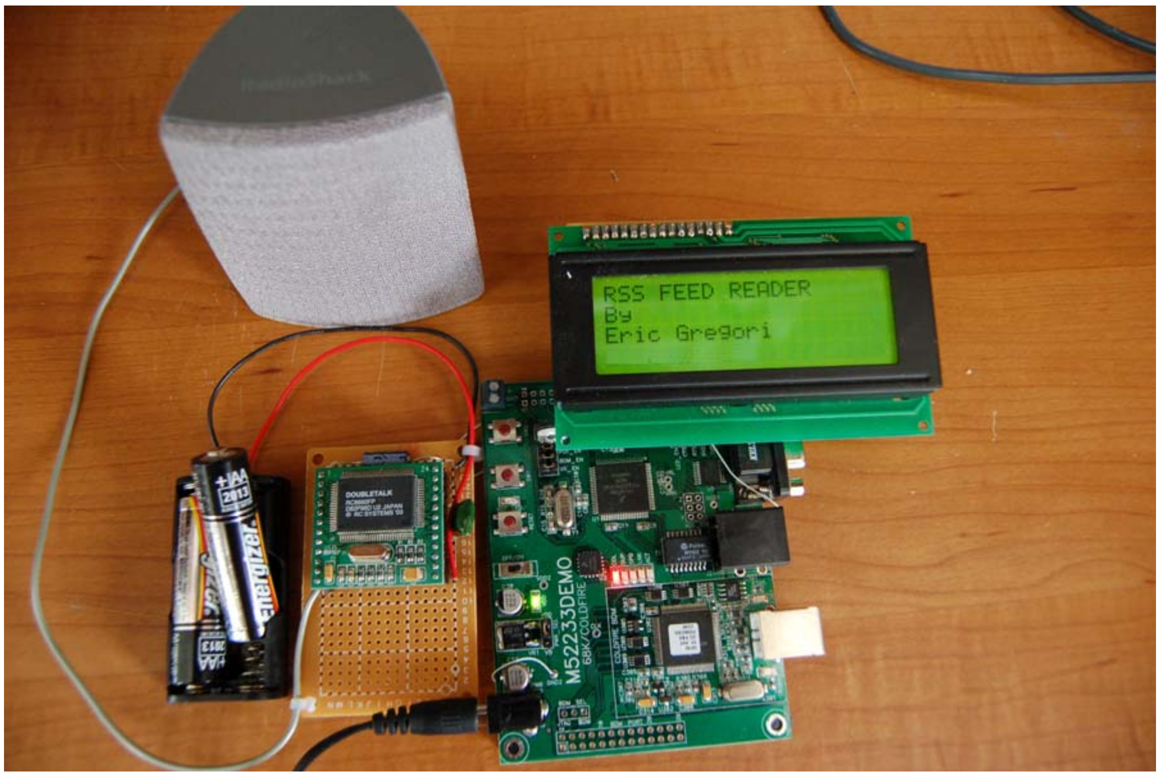
6.0: Example Embedded Appliance - RSS/XML Feed Reader

The RSS/XML Feed reader is an embedded appliance that allows users to display and "hear" real-time content from the World Wide Web. The purpose for the embedded appliance is to provide instant real-time information without booting a PC. There are many types of real-time data available on the web; weather (current and forecast) data, on-line DVD queue data, on-line auction data, sports score data, real-time news data, real-time stock data, medical and health data, and much more. All this data is available on the web as either XML or RSS feeds. This appliance connects to the web, get's the desired feed, and parses the text information or character data from the feed. That data is displayed on the LCD, the serial port, and spoken through the text to speech processor.

For complete details on the RSS feed reader please reference AN3518.



World Wide Web
Sports
weather
DVD queues
on-line auctions
News
Stocks
medical and health
ethernet
Text to Speech Voice Synthesizer
LCD
ColdFire
ColdFire TCP/IP
HTTP Client
RSS/XML Parser
serial



RSS FEED READER
By
Eric Gregori
M52233DEMO

6.1: M52233DEMO board from FreeScale Semiconductor

The M52233DEMO board is a reference board used to evaluate the ColdFire MC52233 processor from FreeScale Semiconductor. The DEMO board includes a serial port, USB BDM debug port, and a Ethernet port. The board along with the free (up to 128K of flash) CodeWarrior tools are all you need to get up and running on your ethernet projects. FreeScale provides a free public source TCP/IP stack on there website. This TCP/IP stack is what the application in this article run on. The ColdFire TCP/IP stack is documented thoroughly in the AN3470 application note from FreeScale.

The DEMO board includes a 40 pin header connector giving the user access to most of the signals from the ColdFire microcontroller. The demo board includes a 3-axis accelerometer connected to 3 of the ColdFire's analog ports. The board also includes a potentiometer, and 2 user buttons.

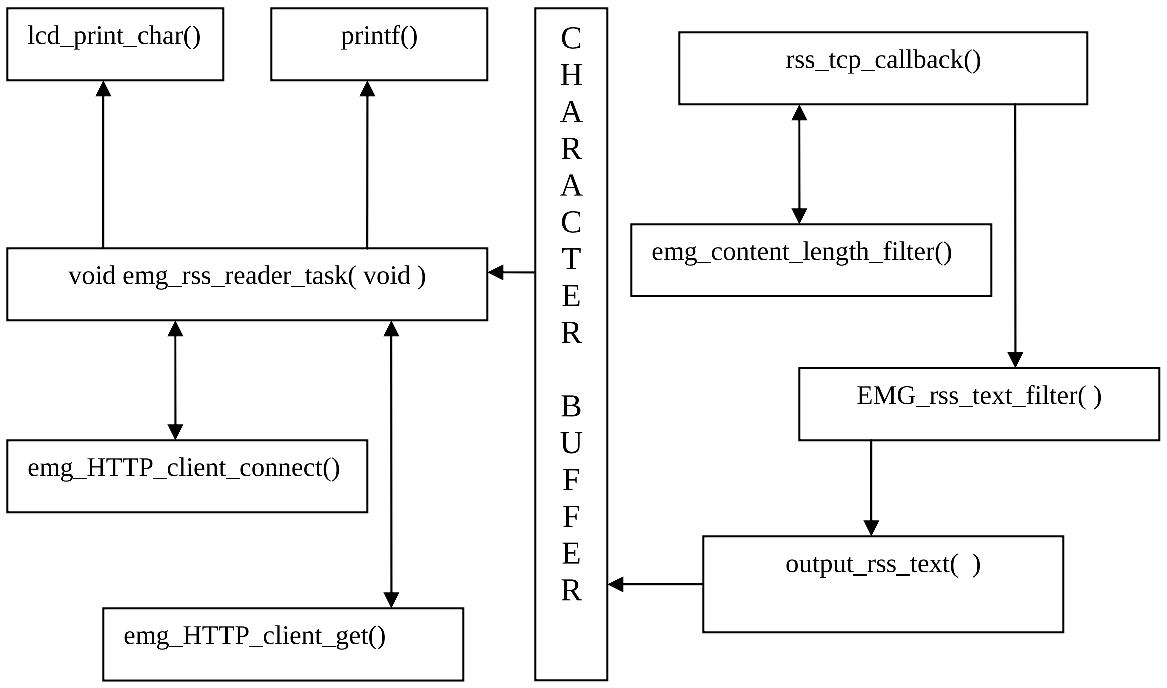
6.2: LCD

The parallel LCD is a 4 X 20 character display that uses the standard Hitachi instruction set. The LCD is used in it's 4 bit mode, requiring only 6 connections to the micro. The 4 bit data bus, a clock signal (E), and a register select line (RS). The firmware includes a library to drive the LCD.

6.3: Voice Synthesizer

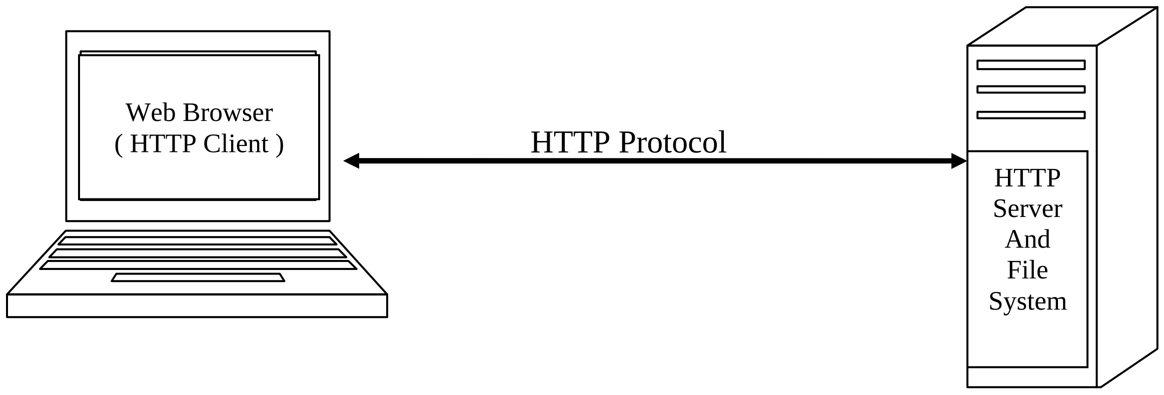
The RCSystems V-Stamp Voice Synthesizer is a easy to use text to speech processor. The V-Stamp is a fully self contained module, requiring only power, a speaker, a resistor, 2 capacitor, and a serial connection to an embedded system. The V-Stamp communicates with the embedded system using a UART. The module automatically sets it's baudrate to that of the embedded systems. From both a hardware and firmware point of view, there is very little work required to add the V-Stamp module to the RSS Feed Reader.

6.4: Firmware



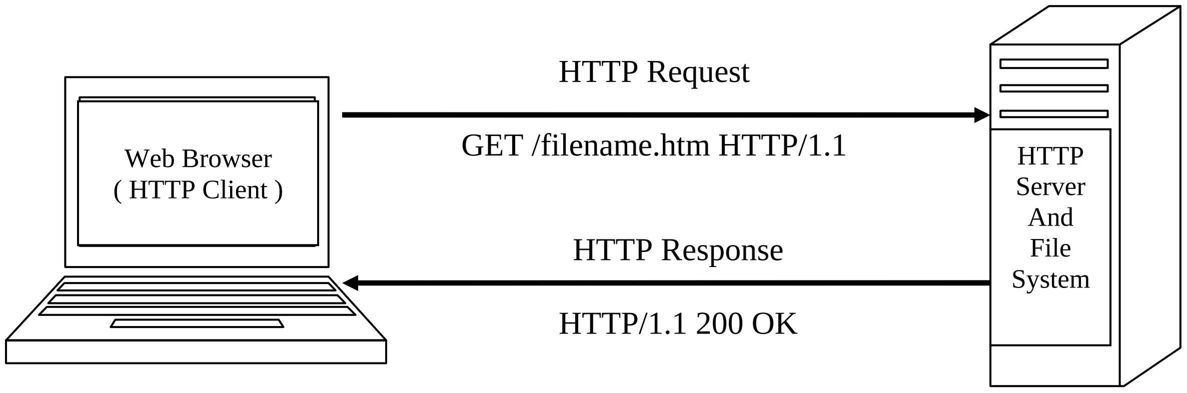
6.5: HTTP Protocol

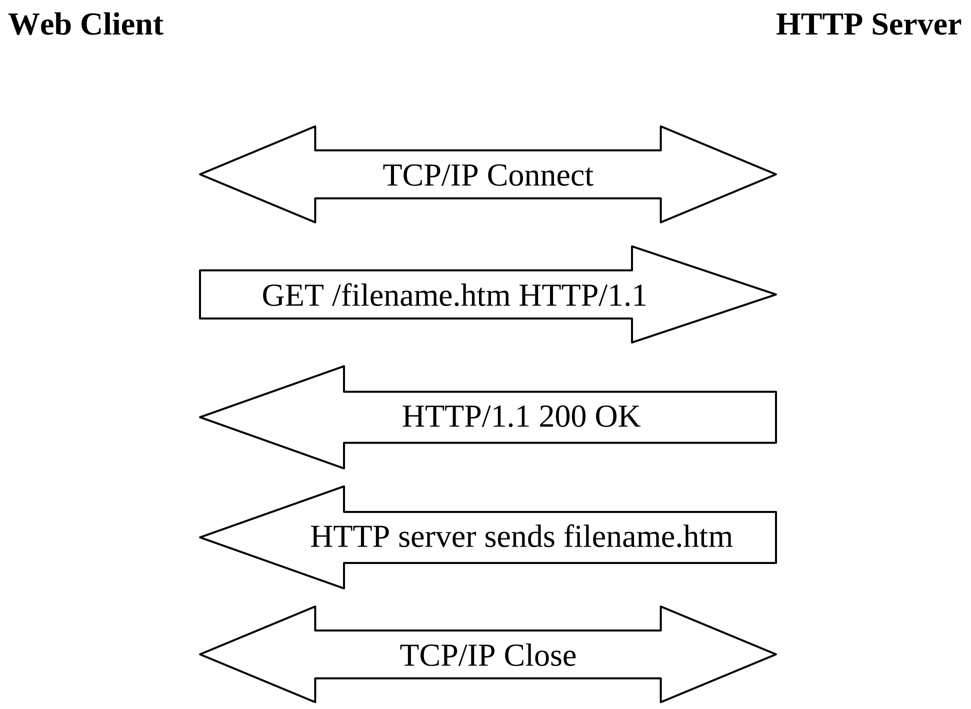
HTTP (Hyper-Text Transport Protocol) is the communication protocol of the World Wide Web. HTTP is used to transfer web pages (Hyper-Text documents) across the internet. A HTTP connection has 2 parts, the HTTP client (web browser) and the HTTP server. The HTTP client is used to receive and view the web page. The HTTP server is used to store, organize, and transfer the web pages.



HTTP is defined by RFC2616, and RFC1945. RFC (Requests For Comments) are technical specifications that define the internet. RFC’s are published by the Internet Engineering Task Force (IETF). See - http://www.rfc-editor.org/ - for more info on RFC’s. RFC2616 defines HTTP version 1.1, the latest version. RFC1945 defines HTTP verion 1.0.

HTTP is a request response protocol. The client requests a web page from the server, the server responds with the web page contents (HTML (Hyper-Text Markup Language)). HTTP can be used to send any type of data, including binary data. The client requests a file using the GET method (HTTP is a ascii protocol). The server responds with a HTTP header followed by the file contents. The client can also send a file using a POST method. Within the request the HTTP version is also embedded in ascii. This tells the server the limitations of the client.


HTTP Request
GET /filename.htm HTTP/1.1
Web Browser (HTTP Client)
HTTP Server And File System
HTTP Response
HTTP/1.1 200 OK



6.6: Really Simple Syndication (RSS)

RSS feeds are available everywhere on the internet. They are used to convey information like local weather, search engine results, DVD queue information, DVD new releases, headline news, and even sports news. The idea behind the RSS feed is to convey textual, dynamic information in a standard simple format.

RSS originated in 1999, with the idea to provide content in a simple easy to understand format. Instead of describing a document like HTML (HyperText Markup Language) RSS feeds use XML to describe data. A RSS feed is simply a XML document containing data. The methods used to convey the data within the XML document are described in the RSS 2.0 specification. All RSS files must conform to the XML 1.0 specification. RSS feeds generally use HTTP as the transport mechanism.

6.7: Extensible Markup Language (XML)

The XML 1.0 specification can be found at www.w3.org/TR/REC-xml/

XML is a language used to describe and parse information. XML is very similar to structures in C. Data is organized into elements, with each element assigned to a tag. The data in the element is surrounded by a start and end tag. End tag names must be the same as the start tag names. End tags are identified by the addition of a '/' before the tag name. The name in the start and end tags gives the element's type.

6.7.1: Tags

```
<TITLE>Advanced ColdFire TCPIP Clients</TITLE>
```

TITLE is the type, <TITLE> is the start tag, and </TITLE> is the end tag. The data is between the tags. Just like a C data structure, the data is associated with the type. The data between the start and end tags is referred to as the element's content.

Elements can contain other elements, providing a method of grouping data of different type under a single name. Just like a C structure, the particular piece of data is referenced by specifying a path to the data.

```
<APPNOTES>
        <BYEG>
                <AN3455>ColdFire HTTP server</AN3455>
                <AN3470>ColdFire TCP/IP Stack</AN3470>
                <AN3492>ColdFire USB Stack</AN3492>
        </BYEG>
</APPNOTES>
```

6.7.2: Special Characters and escape sequences

The ‘&’ , ‘<’, and ‘>’ characters are special XML characters. These characters are used as indicated above to define XML tags, and define escape sequences. Escapes sequences are a method of specifiying a character via a code as opposed to a single symbol. Escape sequences start with a ‘&’ ampersand, and end with a ‘;’ semicolon.

‘<’ is indicated using the escape sequence &lt;.
‘>’ is indicated using the escape sequence &gt;
‘&’ is indicated using the escape sequence &amp;

6.7.3: CDATA sections, the exception to the rule

All rules have exceptions, and in XML it’s the CDATA section. Any text specified in a CDATA section is ignored by the XML parser, allowing the use of special characters without being escaped. A CDATA section starts with a ‘<!CDATA[‘ string, and is terminated with a ‘]]>’ string. Anything between the brackets is ignored by XML.

<![CDATA[character data here is ignored by XML parser]]>

<![CDATA[<THIS_IS_NOT_A_TAG> &lt;]]>

Since the text between the brackets is ignored by the XML parser, both the tag and the escape sequence are ignored by the XML parser, and are interpreted as text or character data.

6.7.4: Finding the text or character data in a XML document

From the XML 1.0 specification, “All text that is not markup constitutes the character data of the document”. That would include all the text between the brackets in a CDATA section, and any text not between ‘<’, ‘>’ brackets in the main body. Escape sequences in the main body represent a single piece of character data.

To filter out the character data from a XML document simply remove all the tags and ‘<’, ‘>’ brackets. Then translate the escape sequences into actual characters.

6.7.5: Sample XML file

In the sample XML document below, notice how data is encapsulated by tags. Where the tag names describe the data. It even looks like a C structure. To find the desired data in a XML document simply look for the start and end tags with the name of the type of data you are looking for. The actual data associated with the desired type is between the tags.

http://www.weather.gov/data/current_obs/KUGN.xml

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <current_observation version="1.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xsi:noNamespaceSchemaLocation="http://www.weather.gov/data/current_ob
   s/current_observation.xsd">
<credit>NOAA's National Weather Service</credit>
<credit_URL>http://weather.gov/</credit_URL>
- <image>
<url>http://weather.gov/images/xml_logo.gif</url>
<title>NOAA's National Weather Service</title>
<link>http://weather.gov</link>
  </image>
<suggested_pickup>15 minutes after the hour</suggested_pickup>
<suggested_pickup_period>60</suggested_pickup_period>
<location>Chicago / Waukegan, Waukegan Regional Airport, IL</location>
<station_id>KUGN</station_id>
<latitude>42.420</latitude>
<longitude>-87.870</longitude>
<observation_time>Last Updated on Jul 28, 10:52 am CDT</observation_time>
<observation_time_rfc822>Sat, 28 Jul 2007 10:52:00 -0500
   CDT</observation_time_rfc822>
<weather>Overcast</weather>
<temperature_string>73 F (23 C)</temperature_string>
<temp_f>73</temp_f>
<temp_c>23</temp_c>
<relative_humidity>81</relative_humidity>
<wind_string>From the Northeast at 13 Gusting to 21 MPH</wind_string>
<wind_dir>Northeast</wind_dir>
<wind_degrees>30</wind_degrees>
<wind_mph>12.65</wind_mph>
<wind_gust_mph>21</wind_gust_mph>
<pressure_string>29.98" (1014.1 mb)</pressure_string>
<pressure_mb>1014.1</pressure_mb>
<pressure_in>29.98</pressure_in>
<dewpoint_string>67 F (19 C)</dewpoint_string>
<dewpoint_f>67</dewpoint_f>
<dewpoint_c>19</dewpoint_c>
<heat_index_string>73 F (23 C)</heat_index_string>
<heat_index_f>73</heat_index_f>
<heat_index_c>23</heat_index_c>
<windchill_string>NA</windchill_string>
<windchill_f>NA</windchill_f>
<windchill_c>NA</windchill_c>
<visibility_mi>10.00</visibility_mi>
<icon_url_base>http://weather.gov/weather/images/fcicons/</icon_url_base>
<icon_url_name>ovc.jpg</icon_url_name>

   <two_day_history_url>http://www.weather.gov/data/obhistory/KUGN.html
   </two_day_history_url>
<ob_url>http://www.nws.noaa.gov/data/METAR/KUGN.1.txt</ob_url>
<disclaimer_url>http://weather.gov/disclaimer.html</disclaimer_url>
<copyright_url>http://weather.gov/disclaimer.html</copyright_url>
<privacy_policy_url>http://weather.gov/notice.html</privacy_policy_url>
```

```
</current_observation>
```

6.7.6: The problem with XML documents

The problem with XML documents is the flexibility of the tag names. The creator of the XML document can use any name to describe the data. A standard is required to “standardize” the tag names, and how data is associated with type. That standard is the RSS standard.

6.8: The RSS Specification

RSS is a dialect of XML. RSS embeds HTML constructs into the XML architecture. RSS also defines a set group of elements, and a general template using those elements. There are many elements defined in the specification, this appnote will concentrate on the elements of interest for the RSS appliance.

6.8.1: A typical RSS file:

```
<channel>
        <title>The name of the channel</title>
        <link>URL to the HTML website corresponding to this channel</link>
        <description>Text describing the channel</description>
        <item>
                < title>Item1 Title</title>
                <link>URL of item 1</link>
                <description>Text for item 1</description>
        </item>
        <item>
                <title>Item2 Title</title>
                <link>URL of item 2</link>
                <description>Text for item 2</description>
        </item>
</channel>
```

The organization of the RSS file can be compared to a newspaper. The channel is the name of the paper, the items are the articles in the paper, the titles are the titles of the articles, and the descriptions are the text of the articles.

6.8.2: Sample RSS file

The sample RSS file is the same data as the sample XML file above. Notice the structure and tag names. The RSS standard defines the title and description tag names. Those tags contain the data we need. All RSS 2.0 compliant feeds will contain title and description tags.

http://www.weather.gov/data/current_obs/KUGN.rss

```
 <?xml version="1.0" encoding="ISO-8859-1" ?>
- <rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
- <channel>
  <title>Weather at Chicago / Waukegan, Waukegan Regional Airport, IL - via
     NOAA's National Weather Service</title>
  <link>http://www.weather.gov/data/current_obs/</link>
  <lastBuildDate>Sat, 28 Jul 2007 16:32:11 UT</lastBuildDate>
  <ttl>60</ttl>
  <description>Weather conditions from NOAA's National Weather
     Service.</description>
  <language>en-us</language>
  <managingEditor>robert.bunge@noaa.gov</managingEditor>
  <webMaster>w-nws.webmaster@noaa.gov</webMaster>
- <image>
  <url>http://www.weather.gov/images/xml_logo.gif</url>
  <title>NOAA - National Weather Service</title>
  <link>http://www.weather.gov/data/current_obs/</link>
    </image>
- <item>
  <title>Overcast and 73 degrees F at Chicago / Waukegan, Waukegan Regional
     Airport, IL</title>
  <link>http://weather.noaa.gov/weather/current/KUGN.html</link>
  - <description>
  - <![CDATA[
  <img src="http://weather.gov/weather/images/fcicons/ovc.jpg"
     class="noaaWeatherIcon"  width="55" height="58" alt="Overcast"
     style="float:left;" /><br />
    ]]>
    Winds are Northeast at 13 Gusting to 21 MPH. The pressure is 29.98"
     (1014.1 mb) and the humidity is 81%. The heat index is 73. Last Updated
     on Jul 28, 10:52 am CDT.
    </description>
  <guid isPermaLink="false">Sat, 28 Jul 2007 10:52:00 -0500 CDT</guid>
    </item>
    </channel>
    </rss>
```

6.9: RSS/XML character data filter

To extract the character data (the information you actually want to read) from the RSS stream, all the Meta-Text must be filtered out. Any HTML that can be processed must be translated and processed. For instance, a <br> is HTML for a new line. This would appear as &lt;br&gt; in the RSS stream. The filter must correctly translate &lt;br&gt; into a carriage return and line feed. Other HTML tags that are routinely embedded in character data include paragraph tags <p> and image tags <IMG … >. In the stream these tags appear as &lt;p&gt; and &lt;IMG … &gt; repectively. The paragraph tab can be translated to a carriage return and line feed, but the image tag must be ignored unless the embedded system can process images.

6.9.1: RSS/XML character data filter state machine

The character data filter is implemented as a finite state machine. The state machine uses only 2 global variables, a state variable of-course, and a FILO. The purpose of the FILO is to store previous characters. Each byte entered into the filter is shifted left into the FILO. So the most recent character is at location filo_buff[FILO_BUFF_SIZE-1]. The FILO buffer size must be larger then the largest tag name expected. The FILO buffer size is set with the FILO_BUFF_SIZE macro.

```
#define FILO_BUFF_SIZE          32
```

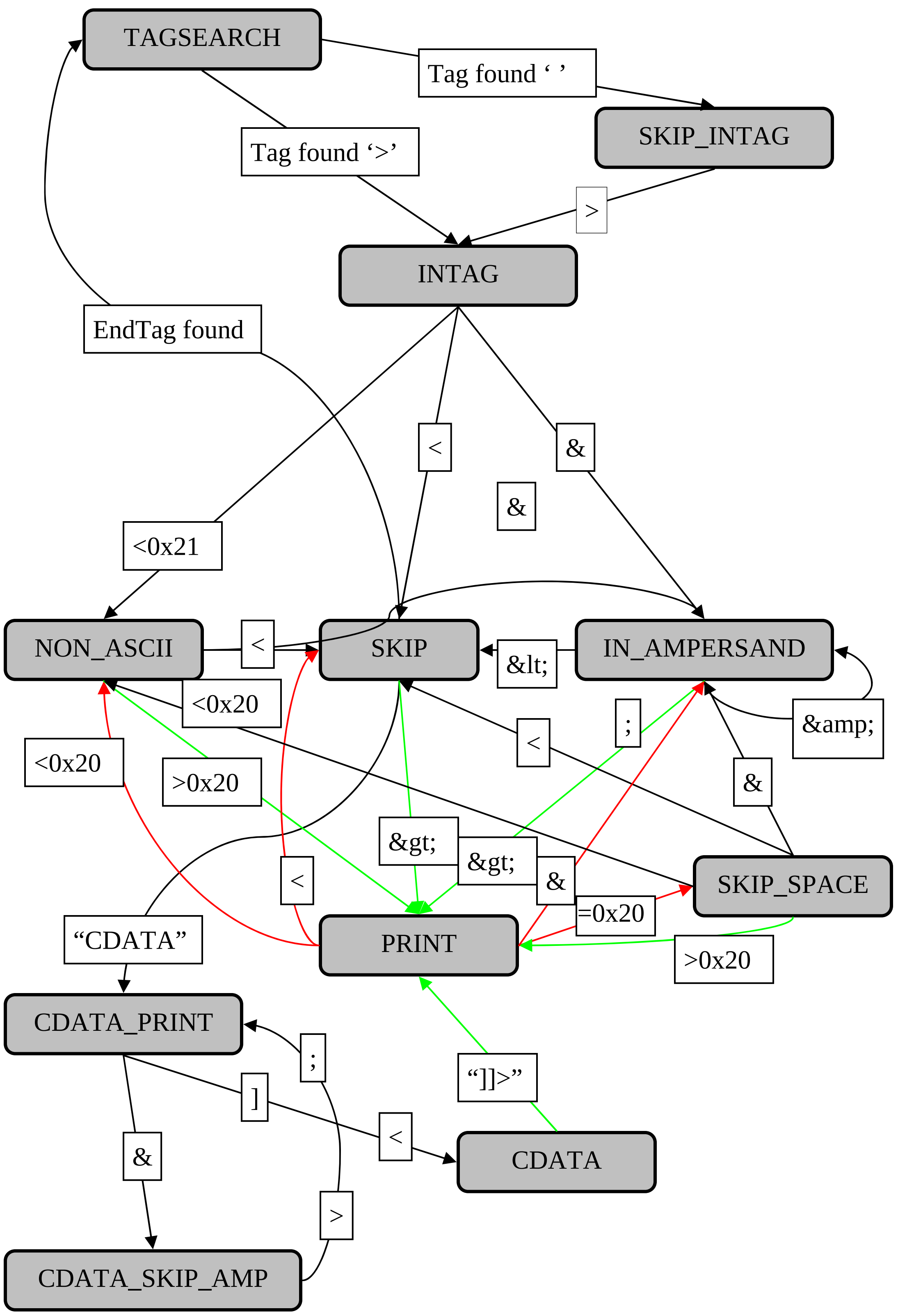
States:

```
        STATE_ZERO
        STATE_TAGSEARCH
        STATE_INTAG
        STATE_PRINT
        STATE_SKIP
        STATE_SKIP_NON_ASCII
        STATE_SKIP_SPACE
        STATE_CDATA
        STATE_CDATA_PRINT
        STATE_CDATA_SKIP_AMP
        STATE_SKIP_INTAG
        STATE_IN_AMPERSAND
```

Global Variables:

```
        unsigned char filo_buff[FILO_BUFF_SIZE];
        unsigned char state;
```

```
unsigned char EMG_rss_text_filter( unsigned char data, unsigned char **tag_filter )
```



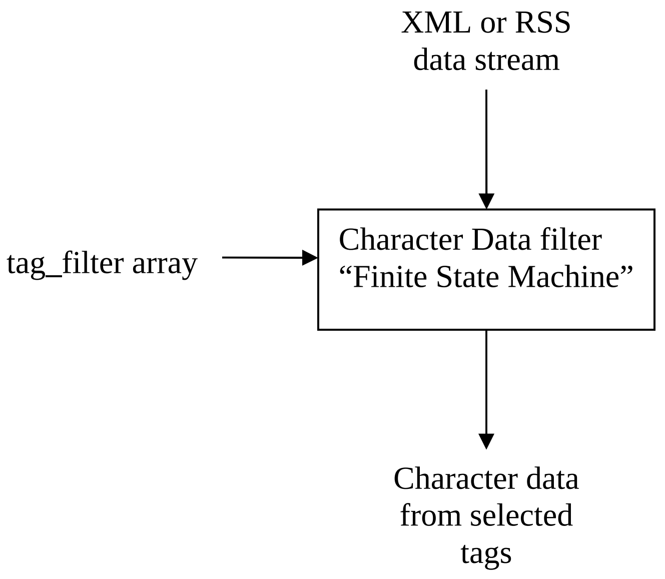
TAGSEARCH
Tag found ‘ ’
SKIP_INTAG
Tag found ‘>’
>
INTAG
EndTag found
<
&
&
<0x21
NON_ASCII
<
SKIP
&lt;
IN_AMPERSAND
<0x20
<
;
&amp;
<0x20
>0x20
&
&gt;
&gt;
&
SKIP_SPACE
<
=0x20
“CDATA”
PRINT
>0x20
CDATA_PRINT
;
“]]>”
]
<
CDATA
&
>
CDATA_SKIP_AMP

The filter takes in a XML or RSS data stream, and a list of tags. It outputs the character data only from the selected tags. The tag list is a array of pointer to the tag strings the filter should be filtering character data from.

Normally the filter returns 0. When the filter processes the '>' in a tag that is in the list, the filter returns the index into the filter array for the tag that it found plus 1. Example: after detecting a <title> tag in a RSS or XML stream the filter will return 1. After detecting a <description> tag in a stream the filter will return 2. Normally the filter returns 0.

```
Const unsigned char *tag_filter[] =
{
                                        {(const unsigned char *)"title"},
                                        {(const unsigned char *)"description"},
                                        {(const unsigned char *)""}
};
```

Sample tag_filter pointer array


XML or RSS data stream
tag_filter array
Character Data filter "Finite State Machine"
Character data from selected tags

Character Data Filter usage

6.10: Using the RSS/XML feed reader

Using the RSS/XML feed reader firmware is easy:

1) Set the url variable to the URL or the desired RSS or XML server.

```
static const unsigned char url[] = "http://www.weather.gov/data/current_obs/KUGN.rss";
```

2) Set the tag_filter[] to the type of data you want to display.

For RSS feeds, <title> and <description> would be a first choice.
For XML feeds, this could be any tag name depending on the desired information.

```
const unsigned char *tag_filter[] =
{
        {(const unsigned char *)"title"},
        {(const unsigned char *)"description"},
        {(const unsigned char *)""}
};
```

3) Set the character buffer size big enough to collect your filtered data.

```
#define         RSS_CHARACTER_BUFF_SIZE         2048
```

4) Compile the project and flash it to the board.

After the TCP/IP stack comes up it will use the DHCP client to automatically aquire a IP address and DNS IP addresses.

Then it will display and speak a title screen.

And connect to the server specified in the URL.
The status of the connection is displayed and spoken.

The file is downloaded from the server using the HTTP client.

And finally displayed and spoken after the connection closes.

Then the RSS/XML feed reader sleeps waiting for eather SW1 or SW2 to be pushed. This will initiate a connection to the server specified in the URL and start the download/display/speak process all over again.

6.11: XML streams

For XML streams the EMG_rss_text_filter() return value is used to determine which tag the FOLLOWING data will be from. This allows another const array containing more descriptive names to be used to describe the data from the tag. Simply use the EMG_rss_text_filter() return value – 1 to index into a descriptive name array and send the indexed string into the character buffer before leaving the rss callback function. The XML filter will then place the data from the tag in the character filter directly after the descriptive name.

7.0: Conclusion

Adding Ethernet to embedded products provides a level of connectivity never before available in the embedded space. Embedded devices can be networked together using the same cables and hardware used by the industry to connect PC together. The embedded device can become part of the PC network. Ethernet and TCP/IP enable the embedded device to connect to the ultimate network, the internet. Once the embedded device is connected to the internet it available to the world (and controllable or viewable from anywhere in the world). The possibilities are endless, from remote monitoring and control, to distributed control, and even real-time worldwide data presentation (RSS feed reader).