ZiLOG and the Embedded Internet – a White Paper

Bill Giovino
http://Microcontroller.com

Introduction – the Evolution of Embedded Systems and the Migration to Embedded Internet

Of all the semiconductor industries, the Embedded Systems marketplace is the most conservative. Engineering decisions in this market are usually conservative, leaning towards established, low-risk solutions. Because of this, the basic infrastructure of embedded systems has only evolved slowly over the past ten years; 8-bit microcontrollers are still the workhorses of the industry, with 16-bits slowly gaining ground. C Compilers, debuggers, and RTOSes gradually took root in this industry over the past ten-years. Emulators have slowly given way to high-speed software debugging techniques. Networking of embedded systems has gradually become more prevalent than stand-alone systems.

The keywords here are ‘slowly’ and ‘gradually’. Embedded developers are a conservative lot and are reluctant to jump onto a new technology where there is no clear advantage in project development. That is why well-publicized technologies such as fuzzy logic and embedded java have seen so little market acceptance – these technologies were not attempting to solve any existing development problems.

While the landscape of embedded systems is vast and encompasses a great deal of application diversity, more and more embedded systems are now interconnected thru serially networked systems. Many high-volume embedded systems applications are physically networked via UARTs, CAN, SPI, or Ethernet. Supporting these various hardware standards are software standards, many of them voluntary, conformity of which is subject to change based upon the whim of the design engineer. Because of this, two UART-based systems, for example, probably will not be able to network between each other even though they are based on the same hardware standard.

In addition, many conventional embedded systems are primarily used in control-oriented applications in sensing and affecting external events. Many of these microcontrollers are managed by human controllers by some sort of man machine interface (MMI) – for example, a cash register, a cell phone, a T.V. screen, or a PC interface.

It is this MMI that often represents the most costly investment in the system’s development in both time and money.

These two issues, one of universal connectivity and MMI, have always been issues in the embedded community. Now, along comes Embedded Internet which solves both – TCP/IP for universal connectivity and a browser to manage a user interface.

The reason for Embedded Internet's revolutionary adoption rate in Embedded Systems marketplace is that it effectively addresses two Embedded Systems industry problems – the disparity of networking standards and the inconsistency of user interfaces.

Embedded Internet is replacing the custom non-standard software protocols with the TCP/IP universal standard. The hardware protocol is most commonly Ethernet, but can also be Bluetooth or any other hardware protocol that is compatible with TCP/IP packets.

The user interface is now the ubiquitous browser.

Standard #1 - TCP/IP

TCP/IP is the name given to a **group of protocols**, or "**stack**", that are the main transport of how data is transferred over the internet.

TCP stands for Transport Control Protocol. Its sub-protocols perform the various data transfer tasks along an internet. On a browser the sub-protocol can often be identified by the prefix to the URL, for example, "HTTP:".

The most well-known types of TCP sub-protocols are:

- **FTP** – transfers files from one computer to another
- **HTTP** – sends a web page from a server to a browser
- **HTTPS** – send a web page in a secure, encrypted format from a server to a browser
- **POP3** – receives an email message
- **SMTP** – sends an email message

A computer that is a web server would start a transfer by putting together a **TCP packet** with the appropriate type of sub-protocol include the data to be transferred. The TCP packet is then encapsulated into another protocol called IP.

IP stands for **Internet Protocol** and it is responsible for getting the TCP data from one IP address to the other. It contains the originating address of the packet and its destination on the Internet. For an example, an IP packet might say "*take this HTTP packet and send it from Yahoo! to a user at IP address 12.45.125.4*".

There is also another type of protocol, called UDP. It is used in some situations instead of TCP. It's function will not be discussed here.

A microcontroller, with a TCP/IP stack and HTTP sub-protocol in software, becomes an **Embedded Web Server** (HTTP TCP). The TCP/IP packets can be sent over an Ethernet connection to a waiting computer (Win9x, Linux, WinCE, SUN) with a browser.

Standard #2 – The Browser

The web browser has become the standard in document presentation. In its most basic form, a web browser's function is to take a text-based document that is formatted by a markup language called "**HTML**" and present the document in a visual display, usually a computer monitor. In the Client/Server model (see below), the HTML pages are stored on the server. They are then served, thru the HTTP TCP/IP protocol, to the browser, which receives the pages and formats the page appropriately for the user's viewing.

The Client-Server Model

TCP/IP networks follow the client-server model, detailed in the following diagram:

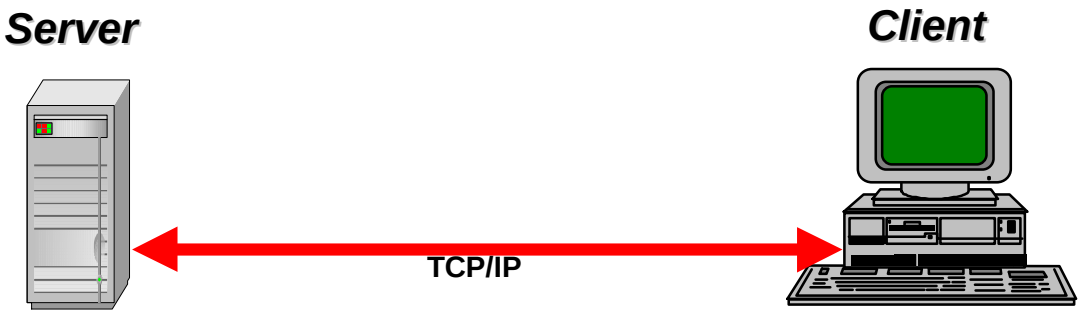

Figure 1

The ***Server*** stores data that is "served" to a **client**. In the case of the World Wide Web, a server stores web pages which can be sent, on demand, to a client. The server typically consists of the data to be sent to a client, an implementation of a TCP/IP protocol stack, and a physical connection to a network that is connected to the vast backbone that is the Internet.

A ***Client*** is simply any type of computer that has a physical connection to the Internet backbone with a TCP/IP protocol stack. The Client also contains an important piece of software called a **Browser**.

While today's browsers can perform a myriad of tasks, the core task is receiving the data in an HTTP protocol packet and rendering the data, via appropriate formatting, in the browser area so that it can be read and understood by the user.

While Figure 1 appears to apply to a conventional World Wide Web server & PC client, the exact same model and description can apply to an Embedded Internet model:

Embedded Web Server (EWS) **Figure 2** **Client**

In Figure 2 the client/server model is the same as in Figure 1; the Web Server in this case is a processor, usually a microcontroller, with an embedded TCP/IP stack and a physical interface to a TCP/IP network. The **Embedded Web Server** also contains the web pages, stored in Flash or EEPROM, that are transferred over the TCP/IP connection to a destination browser.

For simplicity's sake these examples show only one client and one server, but of course the TCP/IP networks that they are connected to can have **multiple clients and servers** on the same network.

Embedded Internet Explained

A microcontroller with an embedded TCP/IP stack (called an **"Embedded Web Server"**, or **EWS**) sends pages over a physical layer connection (ex: DSL, Cable, or more commonly Ethernet) to a remote computer with a browser. The computer with the browser could be in the same room or on the opposite side of the globe. The browser reads the HTML formatting and displays a page in the browser window. The user can send data to the EWS stack thru a **form** on the page, in the exact same way one fills out a form on the World Wide Web – data is entered or buttons selected and then the user presses the "**Submit**" button.

Once "Submit" is pressed, the form data is sent over the physical connection and is received by the TCP/IP stack in the remote microcontroller. The EWS's stack then passes the submitted data to user code in program memory in exactly the same way that parameters are passed to a subroutine. The EWS's application code then takes over and can activate a motor or request a reading from an I/O pin.

The Embedded Internet Challenge

TCP/IP is a protocol that is easy to understand, but difficult to implement. This is because of the many sub-protocols involved and the options each sub-protocol can support. In theory each embedded internet node should be able to interact with every other but in practice many of the present solutions involve a series of compromises.

At the extreme low end is the low-cost small footprint stack. This stack requires less than 4K of program memory and less than 2K of RAM. While these stacks are compliant with TCP/IP standards, in reality they are often compliant with just a **limited subset**. For instance, they may only be capable of supporting one browser connection at a time, making simultaneous connections on a moderately busy network impossible. In addition, these tiny stacks may not be capable of handling some of the more sophisticated HTML pages, reducing them to being capable of handling only simple control tasks with limited I/O such as home water heaters.

At the opposite end is the full-featured 32-bit+ stack with all of the sub-protocols and options found in a full World Wide Web server. These solutions meet all the TCP/IP standards and are specifically targeted for high-demand control tasks on a busy network. In order to accomplish this these solutions sacrifice cost and parts count for no-compromise performance.

But the most important part of an Embedded Internet application is **interactivity**. The web browser needs to able to control and monitor the embedded system. If the Embedded Internet's only function is to serve up static web pages to a browser then it is little more than a document delivery system.

An effective Embedded Internet system needs to be able to have full interactivity with a browser. This translates to:

1. Being able to send data to the Embedded Internet system from the browser thru a **Form**, and
2. Having the Embedded Web Server send data to the browser that reflects a present condition

In the first case, this translates to the web server being able to send form data, in the exact same way one would fill out a name/address form on the web, to an embedded web server. The most effective way of doing this is using a TCP sub-protocol called an **HTTP POST**. This allows the user, via the browser interface, to send data to the embedded web server. This data is then passed, as parameters, to the application subroutines in program memory so that they can take a reading from a sensor or control an actuator.

In the second case this is the embedded web server's ability to receive information from the user's application code, pass it to the TCP/IP stack, and send it to another embedded web server or a browser.

Enter the eZ80

While both the low-end and high-end solutions have their place, the reality is that in the embedded world most applications have the following requirements:

- **Low-cost**
- **Single-chip solution** (memory on chip)
- **8-bit or 16-bit performance** (74% of microcontrollers sold today are either 8-bit or 16-bit)

As the need for more processor speed develops, many applications are becoming even more increasingly cost-sensitive. In an embedded application, a significant amount of the cost is in the on-chip memory. So, the requirement here is for a high-performance chip that is very code efficient.

The solution here is the eZ80. The eZ80 has **8-bit instructions for code efficiency** and **16-bit registers for high application performance** (the Enhanced mode extends all registers to 24-bits). To help facilitate Embedded Internet applications, special instructions help to bulk-move TCP/IP packet data as it is being processed.

A 16Mbyte linear addressing range is one of the largest addressing ranges available for an 8-bit processor, providing protection against any engineer's code expanding beyond capacity as additional features are added. Most importantly, backwards compatibility with the classic Z80 protects the code investment of ZiLOG's present Z80 user community.

More importantly, the eZ80 TCP/IP stack is a no-compromise stack, with all the sub-protocols and options required to function on a busy network. On-chip features and functionality are such that the eZ80 can act as a full-featured embedded web server while maintaining full real-time control over an embedded system.

Applications

Microcontrollers and microprocessors have been networked using TCP/IP in Industrial Automation and Facilities Management since 1994. The following examples will best illustrate the benefits of the eZ80 in Embedded Internet systems against these examples.

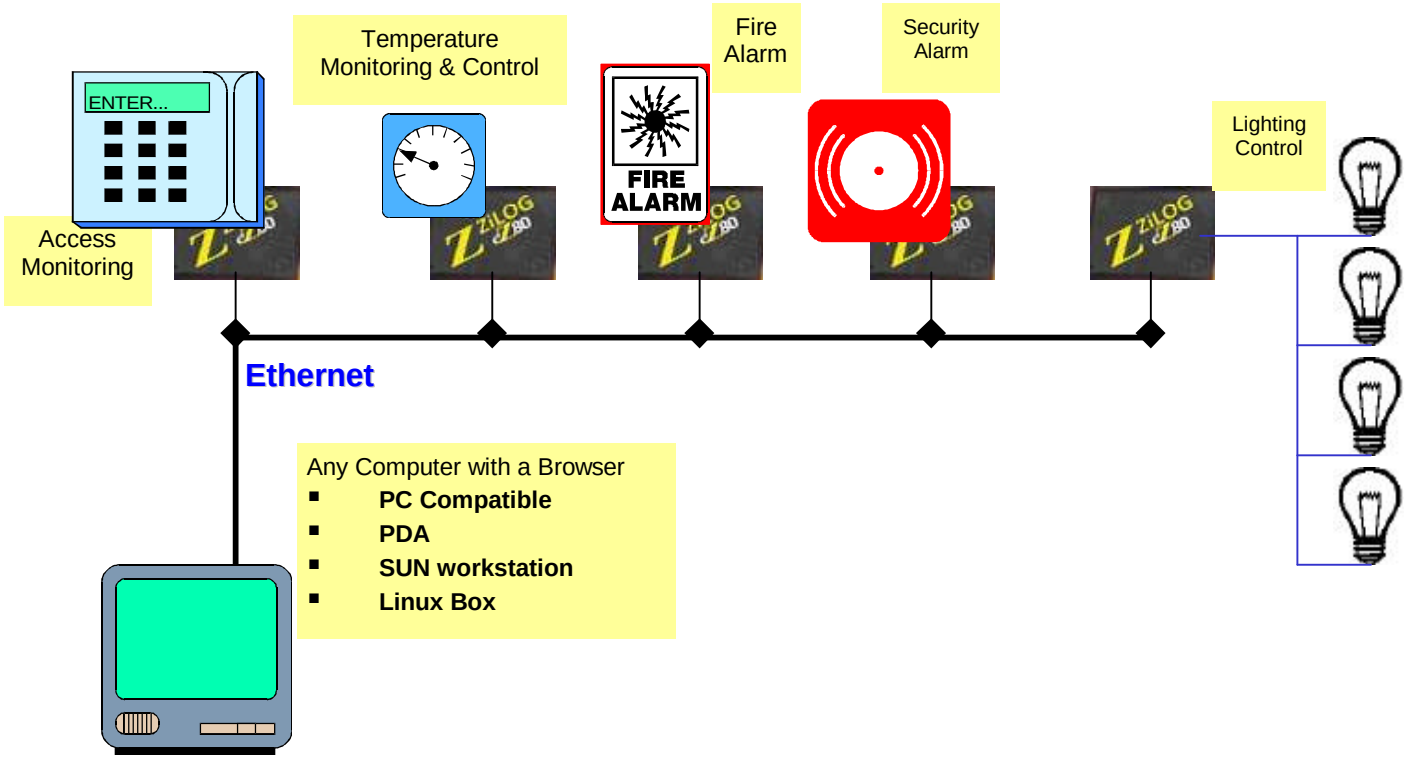

Figure 3 – Facilities Management

The above example shows a network of eZ80s in a facilities management application. Facilities management involves monitoring and controlling all of the complexities of a modern office building – lights, fire alarm, security, elevators, temperature monitoring and control.

It is unrealistic to expect all of these systems in a large facility to be managed manually by flipping switches or turning dials on the wall; today, modern office buildings and skyscrapers are controlled by vast networks that run through the building. In the past, this network has been UART based or more recently via controller area network (CAN), but the sophistication has reached a level where a high speed network with more sophistication is required. Ethernet is used in many new office buildings, and TCP/IP is the underlying protocol. The universal connectivity offered by the standards-based TCP/IP and Ethernet allows nodes from different vendors to interact with each other – the thermostats can communicate directly with the central heating unit, and can report the adjustments in temperature to any computer with a browser that is hooked up to the network. Security control points can interact with lighting, alarms, and door switches. Lights can be sequenced from a browser. Special equipment and training is minimal, as most operators today know how to operate a web browser.

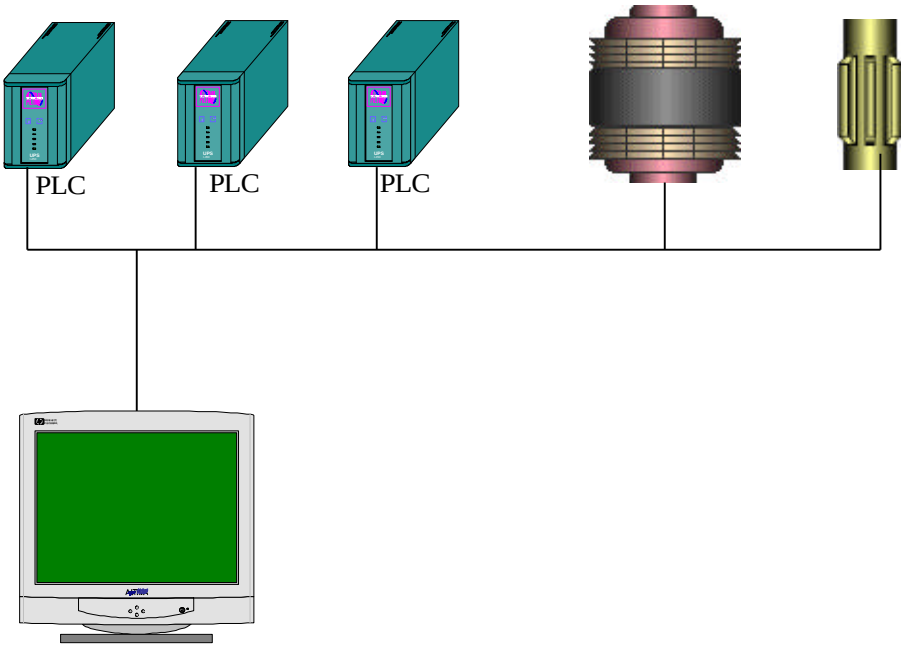

Figure 4 – Industrial Automation

The last example is an Industrial Automation application. In a factory production environment, the manufacturing processes must be tightly controlled. At stake here is the continued smooth operation of the factory; if the system is out of sync, time and dollars can be lost.

The eZ80 can serve as an embedded web server in the various nodes found in an industrial automation system – programmable logic controllers (PLCs), motor control systems, valve controllers, etc. A browser connection from anywhere in the world allows interaction with any system on the factory floor. Because of the popularity of modern browsers, this results in reduced user training since most operators will be familiar with a browser's functions.

From a maintenance side, any authorized browser can be used to install new software modules into the eZ80 embedded web servers. In addition, added value services can be ordered by external access to the vendor's web site, placing an order, and FTP'ing the new code directly into the target eZ80.

Advantages of the eZ80 in an industrial automation application include:

- Increased reliability – over TCP/IP the eZ80 can be monitored, and history data gathered to predict a failure in a controlled production machine.
- Ease of Maintenance – any remote machine that requires calibration or uses consumables is easier to monitor, service, and maintain thru a TCP/IP connection.

- Cost savings – all aspects of the production process can be monitored, including machine load, output production, and production bottlenecks – to further refine the manufacturing process and optimize plant usage.