Application Note

Java Thermostat Demo

AN010403-0203

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

List of Figures iv
Introduction 1
ZiLOG's Approach 1
Background 2
The eZ80® Family 2
HTTP Webserver 3
Embedding the Webserver and Internet Appliance 3
Java and Web Browsers 5
Java in Embedded Systems 5
Theory of Operation 6
Reading and Writing to the Embedded Webserver 8
Implementation Details 8
Java Applet Software Development 8
Embedded Thermostat Firmware Development 9
How the Firmware Works 9
Embedded Webserver Connections 10
Interfacing to the Thermostat Hardware 11
Communicating to the Applet Using the Network 14
Additional Demo Features 15
Getting Started 18
Hardware Requirements 19
Software Requirements 19
Converting Java and other Web Files 20
Java in Embedded Applications 21
Program Listings 23
Java Client Program 23
Thermostat Firmware Program 37
Web Files 49
ThermostatDemo.html 49

Acknowledgement

Staff Application Engineer

Rajiv Pujar

List of Figures

Figure 1. eZ80® CPU Block Diagram 3
Figure 2. Thermostat Java Applet Screen 7
Figure 3. Java Control Demonstration 15
Figure 4. Process Control Demo 17
Figure 5. The eZ80190 Evaluation Board 18
Figure 6. eZ80190 Evaluation Board Simple Block Diagram 19

Introduction

ZiLOG's eZ80® solutions offer an easy way of embedding Java elements on an eZ80® microprocessor to develop Internet-enabled process control and monitoring applications. Applications can be developed more quickly by combining real-time tasks, meeting the developer's requirement for outstanding network functionality and extensibility with Java optimized for embedded use.

One method of providing a Graphical User Interface (GUI) for an embedded system is to use Java applets. The Thermostat application described in this Application Note is written in the C language. This document explains how to use both Java and C in an embedded system.

Among other features, Java provides mechanisms for remote execution, which allows for dynamic updating and reconfiguration. These methods use the same underlying mechanism that enables Java applets, which are the primary source of Java applications in the Internet community.

Java applets are programs that are loaded from remote servers onto local machines. These applets are executed within a browser application that contains the Java Virtual Machine. Java provides a mechanism for the browser to execute unknown applets without the risk of compromising the security of local machine resources.

A variety of languages are used in the design and specification of hardware and software for embedded electronic systems. An embedded systems design requires integration of hardware and software components, which are described using dissimilar languages. The Java Thermostat Demo is a Java-enabled process control application that uses the Internet.

With this Java-based demo application, the user can monitor and manipulate the Thermostat-based control system in real time and can display dynamic temperature values. In this demonstration, the thermostat is connected to the Internet; it can be located anywhere in the world.

ZiLOG's Approach

The present architecture and design of the Java Thermostat incorporates a GUI using Java to connect to the embedded environment. The user interface engine is a Java applet. The real-time and device-specific processes are written in C.

The primary advantages of this methodology are:

1. The Java GUI obtains the data from the low-level embedded software code (C & assembly) and presents the data in a client browser.
2. Java aids in building prototypes and simulations after obtaining feedback from target customers or marketing departments, and before building hardware.

The professional implementation of Java in an embedded environment poses considerable challenges. These challenges can stem from the differences between embedded systems and general-purpose workstations, for which Java is designed. Embedded systems generally lack file and Internet naming services, large memories, multithreading capability, and powerful CPU capability. Nevertheless, a Java-based system for embedded applications must not compromise the robustness of an application.

The Java Thermostat application is based on a technique that isolates the real-time control sections of an embedded application from its Java-based sections, while allowing information to flow freely between the two sections.

The interface allows for the remote manipulation of system control settings such as upper and lower setpoints, and displays a system status via a Java-based GUI. Thus, the Java Interface acts both as a monitor of the system, by using a graphical display to depict the system component, as well as an interactive control for changing system-level parameters. As a result, graphic displays, such as interactive buttons, can allow direct manipulation of the underlying hardware.

This control interface is required for next generation Internet-based control systems because it provides a remote way of handling actions involving system configuration, monitoring, and control. Moreover, if the system is essentially a transducer (sensor), a user interface can also be useful for data collection.

The technology of the World Wide Web provides a way to perform all of these actions and more while using any remote computer equipped with a web browser. The web browser window becomes, in effect, the embedded computer's display, and the remote computer's keyboard and mouse become its input devices. By using a remote computer with the web browser as the user interface, the embedded system can support functionality that would be difficult or expensive to accomplish in more proprietary ways.

Background

The eZ80® Family

The eZ80® is ZiLOG's next-generation Z80 processor core. It is a high-speed, optimized pipeline-architecture microprocessor that can operate at frequencies up to 50MHz. The eZ80® offers an 8-bit CPU capable of executing code four times faster than a standard Z80 operating at the same clock speed. The increased processing efficiency of the eZ80® can improve available bandwidth and decrease power consumption. The eZ80®'s 8-bit processing power rivals the performance of competitors' 16-bit microprocessors. The eZ80® is also the first 8-bit microprocessor to support 16MB linear addressing. Each software module, or each task, under a real-time executive or operating system can operate in Z80-compatible

(64KB) mode or full 24-bit (16MB) address mode. Figure 1 shows the eZ80® CPU block diagram.

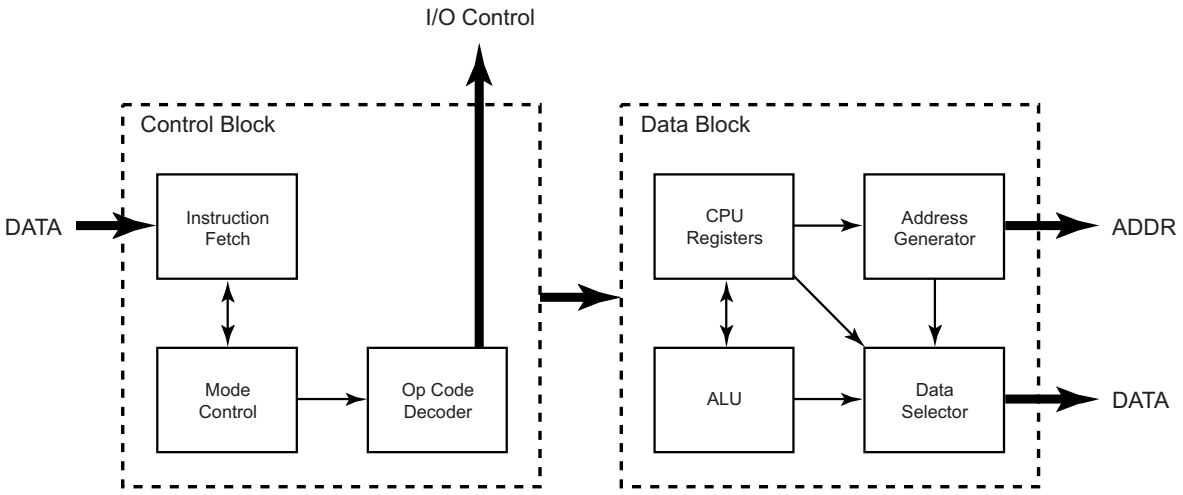


Figure 1. eZ80® CPU Block Diagram

The eZ80190 microprocessor is the first in a line of eZ80®-based standard products targeted toward embedded Internet applications. The Java Thermostat application is tailored to the eZ80190 device.

HTTP Webserver

HTTP webservers use a standard synchronous request/response design over TCP/IP, identical to classical client/server architecture. When a client makes a request to an HTTP server, it sends an HTTP request message. The HTTP request message includes the client request, as well as information about the client's capabilities. The HTTP response is similar to the request, except that it can be composed of two parts—the response header and the response body. The response body represents the result of the initial request. A single blank line in this HTTP response file separates the response header from the response body.

Embedding the Webserver and Internet Appliance

The Difference Between an Embedded Webserver and a Standard Webserver

Most webserver software does not work well in embedded systems. A webserver that is designed to run on a workstation faces a different set of requirements than one designed to run in an embedded system. For example, most nonembedded webservers include features such as log files.

On the other hand, the implementation of embedded webserver software must be concerned with reducing the memory footprint, increasing efficiency and reliability, and with providing mechanisms to generate dynamic data.

Embedded systems do not typically serve a multitude of static web pages. Most often, an embedded webserver exists in a system to provide real-time access to data, or to allow certain system configuration parameters to be modified on the fly. These types of behaviors require the content of most web pages to be generated dynamically.

Generic webservers cannot be adapted to suit the demands of embedded developers because their large memory footprints and resource expectations make them difficult to use with resource-constrained embedded systems.

Internet appliances (and embedded systems, in general) require webservers that enhance their existing functionality without impinging on vital device resources or requiring a redesign. Because many of these systems are cost-constrained, memory and CPU resources are usually at a premium. It is vital that embedded webservers offer minimal memory requirements and be efficient.

The requirements for an embedded webserver include:

- Memory usage
- Dynamic page generation
- Web page storage in Flash or ROM

These concepts are each described below.

Memory Usage. One of the most important requirements for an embedded webserver is small memory footprint. Not only must the webserver use very little memory (code, stack, and heap), it must not fragment memory. Many embedded devices employ simple memory allocations that cannot coalesce fragmented memory effectively. Because webservers often must respond to requests to serve pages as rapidly as possible, simple memory allocations can cause problems. When the memory used to serve a page is freed, it can be useless, as it cannot be merged with adjacent memory blocks on the heap. To solve this problem, embedded webservers should use only statically-allocated or preallocated memory blocks.

Dynamic page generation. An embedded device features only a small number of pages in memory, and often generates part of its content on the fly. These pages display ever-changing information about device status, the values read by sensors, and any other data that is available to the device.

Web page storage in Flash or ROM. Many embedded systems do not feature disk drives, yet still must be accessed and controlled via the web. In such cases, a method of storing web pages in ROM is required. Embedded webservers should

be able to access HTML, Java applets, image files, and any other web contents stored in Flash memory or ROM.

Java and Web Browsers

Web browsers have become the standard method for communicating with, and managing, remote embedded devices. The web browser is a common appliance on networked desktops, providing a rich set of functionality for communication and presentation of data from remote devices.

Implementing Java is important—and even critical—for the users of next-generation embedded systems; in particular, those with Internet and Intranet connectivity. Java enables dynamic software downloading, dynamic monitoring, and dynamic management to improve the capability of adapting to fast-changing market demands and for lowering operation and management expenses.

At present, the practical uses of Java in embedded systems is to develop user-friendly graphical user interfaces and to interface these interfaces to the more traditional parts of the embedded software. As more embedded systems are networked, it is expected that these systems will provide remote GUIs that display on popular types of personal computers and workstations.

In particular, Java is a design and specification language for embedded systems. Java is a pragmatic choice for several reasons. Because it is a member of the C family of languages, it is familiar to designers.

Java also offers practical features for embedded programmers, such as fixed-size integers, a portable binary format, guaranteed protection from memory leaks, and built-in support for threads and synchronization.

Moreover, Java is the development platform of choice for much of the basic research in academia on next-generation, network-processor-based active networks, most high-end, highly-parallel supercomputers, and in both commercial and university efforts to produce a common language environment for writing code that can either be expressed in software or in hardware.

Finally, while Java's expressive power is comparable to C++, it is a much simpler language, which reduces the difficulty of program analysis, optimization, and transformation. Using Java also has many practical benefits—its widespread adoption by the science and engineering communities promises a large base of support, in the form of compilers, debuggers, development environments, and class libraries.

Java in Embedded Systems

An embedded system that uses a Java applet for its graphical user interface requires system software that supports TCP/IP, an HTTP webserver, a short HTML page that refers to the applet, and the Java byte code for the applet. No

Java Virtual Machine is required for the embedded system. Users access the embedded system via any Java-enabled web browser.

A request to read an HTML page loads a GUI applet into the browser and starts executing it. The applet opens a socket and connects to the main application in the embedded system. The main application, which is probably written in C, opens a socket and listens for a connection by the applet. When a connection is made, messages can be sent back and forth between the applet and the main application in response to a user request to see data or change settings.

The Java Virtual Machine that is executing the GUI applet is actually running on the web browser, not on the embedded system. The HTTP server running on the embedded system must support HTML file loads, Java class file loads, and image filetypes, such as .gif, .jpg, etc.

Java Advantages

One advantage to using a Java applet as a GUI is that an applet is stored as a file in the webserver that does not require additional memory from an embedded device to operate. Another advantage of using a Java GUI is that an embedded system vendor can assume that his customers have access to a web browser, no matter what computer platform they choose to access the embedded system from. Additionally, the Java GUI technique works well on a slow network connection, such as a serial line.

A Java GUI is a much better solution than client software, because there is no need to ship any client side media or client side installation instructions with the product, There is no additional cost, because there is no need for OS upgrades or technical support for the client side software.

With a Java GUI, a client computer can exist on any platform, and Java provides a cross-platform GUI. Only one version of the GUI software is required; this version is stored in ROM in the embedded system.

Theory of Operation

This section discusses how the Thermostat controller works. The Java applet is responsible for the control of a Thermostat board, which is interfaced with the eZ80190 Evaluation Board. The Java applet features buttons for setting Thermostat control parameters, such as upper and lower setpoints. Clicking either button sets the point low or high. Figure 2 shows a screenshot of the applet.

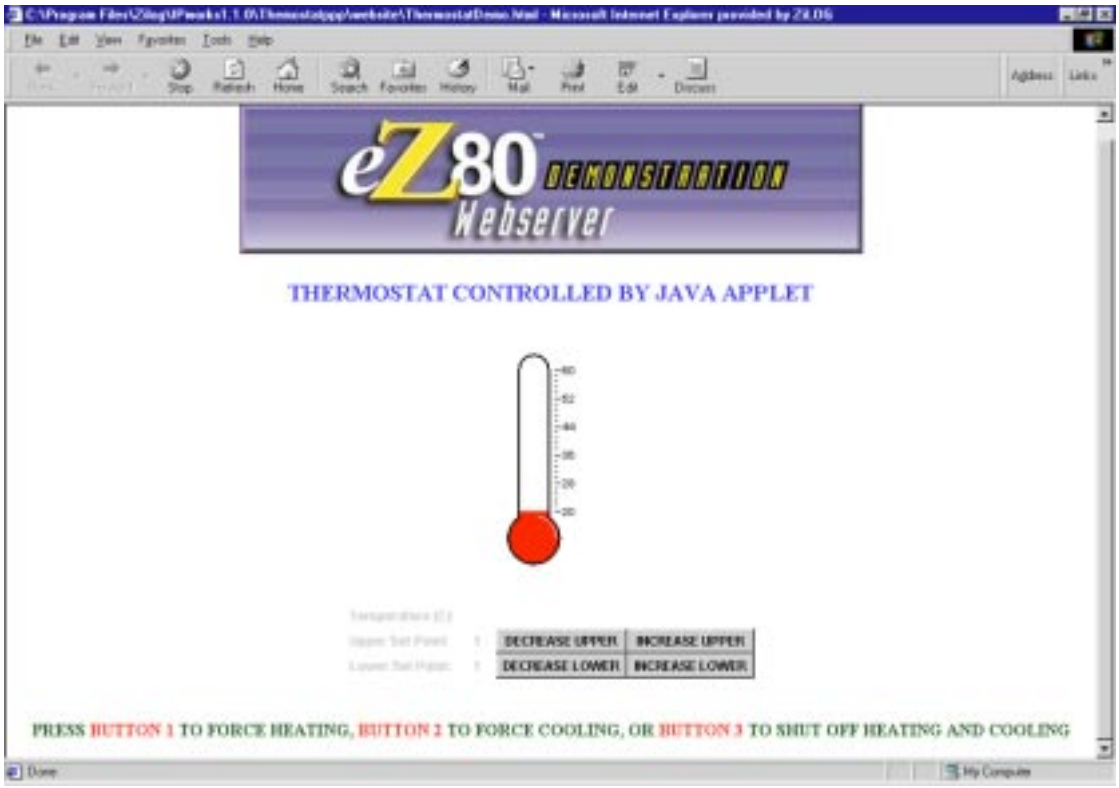


Figure 2. Thermostat Java Applet Screen

As an example, assume a value of 50 as the upper setpoint and 30 as the lower setpoint. These values are passed to the server using the Java socket connections that invokes the CGI script The new setpoint values are read by the main server program `main.c`, which changes the control variables accordingly. The `main.c` program switches on the Thermostat light when the temperature is 30ºC. The light remains on until it reaches a 50ºC upper limit, wherein the light switches off and the Thermostat fan turns on. This cycle repeats between the 30ºC and 50ºC range unless the setpoints are changed. In the meantime, the updated dynamic temperature values are read by the Java applet by invoking the CGI script, and displayed in graphical and numerical format in a web browser.

If the setpoints are changed in the control interface (Java applet), the Thermostat adjusts to the new setpoints, and the control changes.

Reads and Writes to the server occur every 2 seconds. The Java thread running in the applet controls this process.

Reading and Writing to the Embedded Webserver

When the user writes Java programs that communicate over the network, TCP and UDP layers are not a concern. Instead, the user can employ the socket or URL classes in the java.net package.

Direct Socket Connection

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes—Socket and ServerSocket—that implement the client side of the connection and the server side of the connection, respectively. The Java Thermostat demonstration uses only the Socket class because the server is implemented in C, not in Java.

URL Methods

Java programs that interact with the Internet also can use URLs to find Internet resources. Java programs can use a class called URL in the java.net package to represent a URL address. The URL.class represents a Uniform Resource Locator, which is the address of a resource on the network. Java applets can use a URL to reference and connect to these network resources.

If the server supports a protocol that a URL recognizes (for example, http), the URL can be used to create a URLConnection to the server (which normally connects via the Port 80 socket) to manage protocol-specific communication.

Sockets can be used at any time. To use a socket to contact an HTTP server, some of the details that a URL manages must be performed manually. However, using a socket allows a user to manage the communication as the user sees best, assuming the user does not want to use one of the established protocols.

For more information regarding sockets and URLs, please refer to Sun Microsystem's Java website.

Implementation Details

Java Applet Software Development

Creating a GUI for an embedded system with Java is interesting and challenging work. To develop the Java applet, a separate development environment is used. Any Java development environment will do. The embedded OS and tools are not necessary to support Java for this technique to work. Any Java GUI builder can be used. What makes an embedded GUI applet different from most is its interface to the main application running on the embedded system. For a better look and feel, Java Swing components can be used.

For purposes of this Application Note, Sun's JDK is used for compiling Java source files. The compilation process is published elsewhere in this document.

Java applets are only allowed to create socket connections back to the host computer they were loaded from. When the applet requires information from the main application, it sends a request message via the socket. The main application responds by sending a response message back through the socket. All of the graphical user interface details are handled by the applet.

The Java applet for this demonstration provides control and monitoring capabilities to the eZ80®-based Thermostat I/O board. The Java applet provides the following functions:

- It provides a facility for setting the upper and lower setpoints for a Thermostat
- It displays the updated temperature values on a client web browser in graphical and numerical formats

Embedded Thermostat Firmware Development

Main Application

The main application is written in C and is developed using ZiLOG Developer Studio II (ZDSII).

The main application builds the HTTP webserver and the TCP/IP socket connection that allows data to be sent and received as a stream of bytes. The eZ80190 Evaluation Kit provides the software TCP/IP stack that supports socket connections. Java also supports TCP/IP via socket connections. The main application program also calls the eZ80190 Evaluation Board initialization functions.

Applet Firmware Interface

To reiterate, interfaces to the Java applet are written in C and are developed using the C development tools provided by ZDSII. These interfaces are vital for establishing communication links between the Java GUI and the low-level application or firmware code, which is written in C and assembly.

The applet invokes the webserver's CGI scripts contained in the `tstat_control_cgi.c` to write and read from the HTTP webserver. The Thermostat applet receives the temperature data and displays it in the web browser.

How the Firmware Works

This section describes pieces of the source code.

Embedded Webserver Connections

A URL code segment can be used instead of a direct socket connection to a server. To write to a server, create a Port 80 socket connection to the server. The code for getting this socket connection is called a getupdate() method. The following lists part of the getupdate() code.

```
public int getupdate() {
  try {
       s=new Socket(server,80);
  } catch (Exception e) {
  ..........
  ..........
}
```

The code for writing to the socket is called via a sendRequest() method. The following lists part of the sendRequest() code.

```
// create a String Buffer
  request=new StringBuffer();
// Append the String Buffer with the Dynamic page name,
which is used for
// filling request data through HTTP protocol
  request.append("GET/Data.html");Ωß
// append all data that has to be sent to the server like
upper setpoint
// value lower setpoint value
  request.append("max=");
  request.append(Integer.toString(newmax));
// Create a output stream
  try {
       out=s.getOutputStream();
    } catch (Exception e) {
//send input data to server
  try {
       out.write(request.toString().getBytes());
   catch (Exception e) {
```

To read temperature values from the server using URL methods, the following Java code fragment shows how to get dynamic temperature data.

```
try{
   String input = null;
   URL url = new URL("http://192.168.1.1/ Data.txt");
   URLConnection connection = url.openConnection();
   InputStream inStream = new DataInputStream (connection.getInputStream());
   while (null != (input = inStream.readLine()))
      // Get Temp value
            temperature= Integer.parseInt(input);
   inStream.close();
}


}
 catch(Exception e){}
```

To read dynamic data, a URL to the CGI dynamic page (http://192.168.1.1/data.txt) must be built on the server from which the applet is delivered.

After opening the URL connection, the data should be received on this connection. When all of the data is sent through an output stream (by calling the getupdate() method), an input stream (inStream) for the URL connection reads the server's response. As a result, the temperature values are updated.

Interfacing to the Thermostat Hardware

The Main C Program

To this point, this document has discussed the front-end and interface parts of the demo application. This section discusses the low-level part of the application, where the actual control is performed. The main.c program is responsible for building the complete webserver and for controlling the Thermostat board. This main program configures the hardware, initializes the I/O ports and interrupts, builds the webserver, and finally controls the Thermostat device.

The main functions are:

- The eZ80® I/O ports must be configured for reading and writing to the Thermostat board. The following C code fragment shows some of these initializations.

```
/ * Need to setup I/O bits for output (PB7-PB3)
     PB_DDR = 0x07; //0= output 1 = input
/ *Need to setup I/O bits for SW I2C (PD7 & PD6)   */
     PD_DDR = PD_DDR & 0x3f
     PD_ALT2 = PD_ALT2 & 0x3f; //SW I2C lines (PD7 & PD6) are OPEN DRAIN
```

```
PD_ALT1 = PD_ALT1 | 0xc0;
PD_DR = PD_DR | 0xc0;
```

- The structure WebPages defines the kinds of pages that are embedded in the website. All necessary static and dynamic web pages that are built in this structure must be defined. For example, the following C code fragment shows that we are creating index.html and tstat_control.html as dynamic web pages and Thermostat. class as a static web page.

```
Webpage website [] = {
  HTTP_PAGE_DYNAMIC, "/index.htm", "text/html", index_cgi },
  HTTP_PAGE_DYNAMIC, "/Data.html","text/html",tstat_control_cgi },
  {HTTP_PAGE_STATIC, "/Thermostat.class", "application/octect-stream",
  &Thermostat_class },
  ..................................................
  ..................................................
}
```

- To build the webserver http_init() must be called with some parameters to build the webserver. The webserver creates several threads to improve performance. Multiple webservers can also be run on multiple ports.

The following C code fragment illustrates how to build the webserver.

```
http_init(http_defmethods,httpdefheaders,website,80);
```

- For reading the Thermostat temperature, the readtemp() function is used. This method reads the temperature from the Maxim MAX6625 Temperature Sensor. Simulated temperature values can be read by calling the SimulateReadTemp() function if the sensor is not connected to the Thermostat board.

The following C code fragment lists this temperature-reading routine.

```
int readtemp() {
  i2c_error = 0;       // Reset the I2C error indication flag
  i2c_shiftreg = 0x90; // The MAX6625 is at I2C slave address 0x90
  I2CStart();
  I2CShiftOut();
  if (ok) {// If 'ok' and I2C ACK was rec'd and the MAX6625 is there
    i2c_shiftreg = 0x00; // Address the temp register in the MAX6625
  if ok{
```

```
  I2CShiftOut();
  I2c_shiftreg = 0x91; // Repeat Start to MAX6625 to get into read mode
  I2CStart();
  I2CShiftOut();
  I2CShiftIn();   // Read the first (high) temp data byte
  MasterACK();
  temp_high_byte=i2c_shiftreg;
  I2CShiftIn(); // Read the second (low) temp data byte
  MasterNACK();
  temp_low_byte = i2c_shiftreg;
  I2CStop(); // Done reading, generate I2C stop condition
    }
  }
  else{
  SimulateReadTemp (); // If there appears to be no sensor, simulate the
  temp reading
  }
  temp = temp_high_byte;// Only need the High Byte, which is temp in degrees
  C
  return temp;
}
```

LCD Display

The Thermostat Demo also incorporates a liquid crystal display (LCD) panel to display dynamic temperature.The LCD panel is plugged into the Thermostat I/O board. The LCD program `lcdinit.c` initially displays the IP address.The code fragment follows.

```
void Icdinit(void){
U8  *pstring;
    PortAinit();
    LCDInit(M68_4bit,Array5_7,2,16,LCDcursorOff);
    pstring = &string[0];
    LCDwriteString(TextStandard,Line_2,1,pstring,strlen(pstring));
    pstring = &string1[0];
    LCDwriteString(TextStandard,Line_2,1,pstring,strlen(pstring));
    LCDwriteString(TextStadard,Line_2,4,Bootrecord.myip,strlen(Bootrecord.my
    ip));
    suspend(lcdinitpid);
}
```

The temperature is also displayed and the code fragment is given below:

```
int Thermostat_cgi(struct http_request *request) {
     .................................
     .................................
     // This will convert the binary value c into it's ascii representation
     in hex.
     bin_to_ascii(ambient_temp,pstring);

     ControlLCD(ClearDisplay,0);
     // This function writes a data to the LCD module.
     LCDwriteString(TextStandard,Line_1,1,"Thermostat Demo",15);
     LCDwriteString(TextStandard,Line_2,10,pstring,2);
     LCDwriteString(TextStandard,Line_2,8,(pstring+1),2);
   ..............................................................
}
```

Communicating to the Applet Using the Network

The CGI interfaces bridge the gap between Java and the low-level Thermostat control program. The CGI programs, developed in C, help to communicate to the Java applet from the webserver. The CGI interface programs make use of the HTTP functions **http_init()**, **http_get()**, **http_post()**, and **http_request()**. These programs are common functions used to build the embedded HTTP webserver.

For more details, please refer to the documentation contained in the `..\zilog\IPWorks<version>\doc` directory.

The CGI script file **input_cgi.c** passes the maximum and minimum setpoint values from the Java applet to the **main.c** application. This file assists in retrieving the temperature values from the Thermostat I/O board. The code fragment is listed below:

```
int input_cgi(struct http_request *request) {
     ................................................
     ................................................
     // get min and max values from browser

     tstr = http_find_argument(request,"min");
     lower_setpoint_work = atoi(tstr);

     tstr = http_find_argument(request,"max");
     upper_setpoint_work = atoi(tstr);
```

```
    // Fill the buffer with the actual values
    // Write to the server - Send the response to the browser
  __http_write(request,ttmpbuf,strlen(ttmpbuf));
}
```

Additional Demo Features

The present demo exhibits control actions by HTML forms and LED control via an applet.

LED Control Applet

The Buttonapplet demo shows how to control the LEDs, how to read the status of the LEDs, how to start the Thermostat heating and cooling process, and how to read the response. These LEDs are shown in Figure 3.

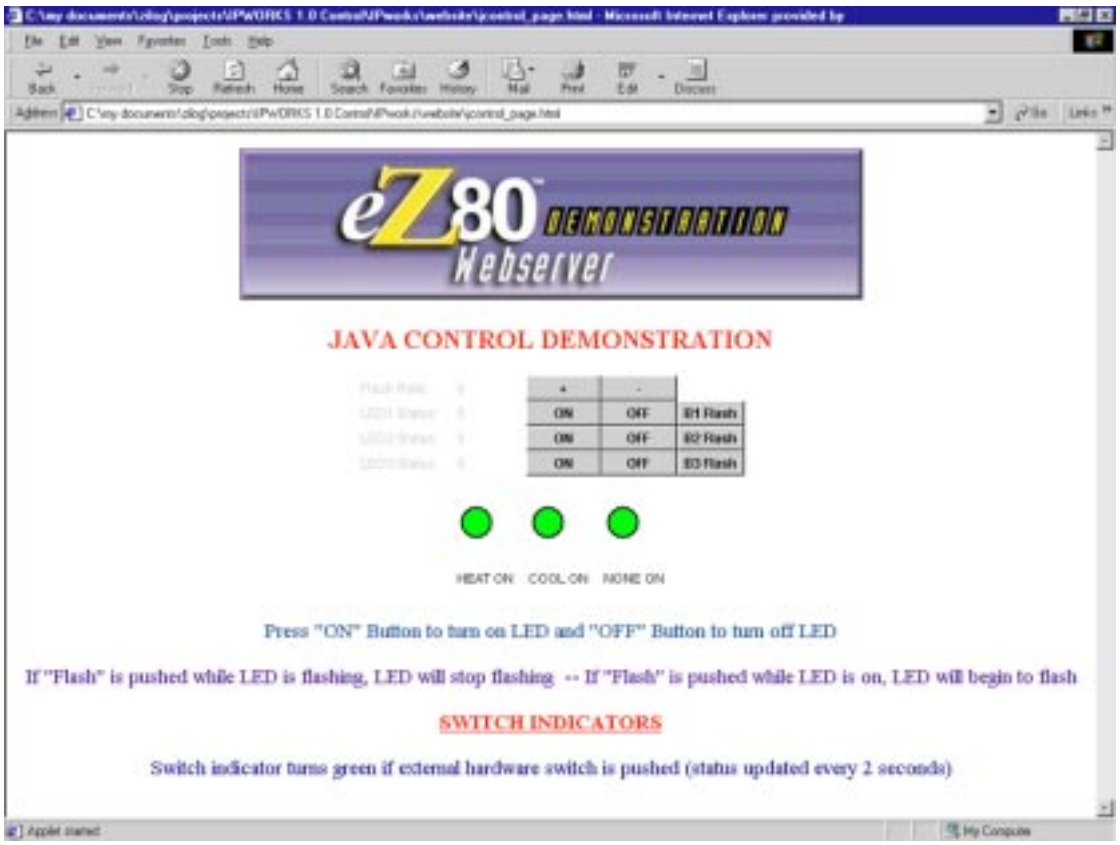


Figure 3. Java Control Demonstration

The Buttonapplet makes use of the CGI script file **java_control_cgi.c** to check the switch inputs and control the LEDs. The code fragment is listed below for reference.

```
// get flash speed from browser entry
jstr = http_find_argument(request,"jflash_speed_browser");
/* check switch 1 and red LED1 status settings */
jstr = http_find_argument(request,"LED1");
jswitch1 = atoi(jstr);
if(jswitch1 == 1) { j
  out_hold &= ~LED1_out_mask;
  PB_DR &= ~LED1_out_mask;
  LED1_status=1;
  }else if(jswitch1 == 2) {
  {
  jflash_mask &= ~LED1_flash_mask; flash_mask = jflash_mask; jout_hold |=
  LED1_out_mask;
  PB_DR |= LED1_out_mask;
  LED1_status=2;
  }
```

LED Control via an HTML Form

This demo shows how to control the LEDS. It demonstrates On/Off and flash rate control using HTML forms. Figure 4 shows a typical GUI form.

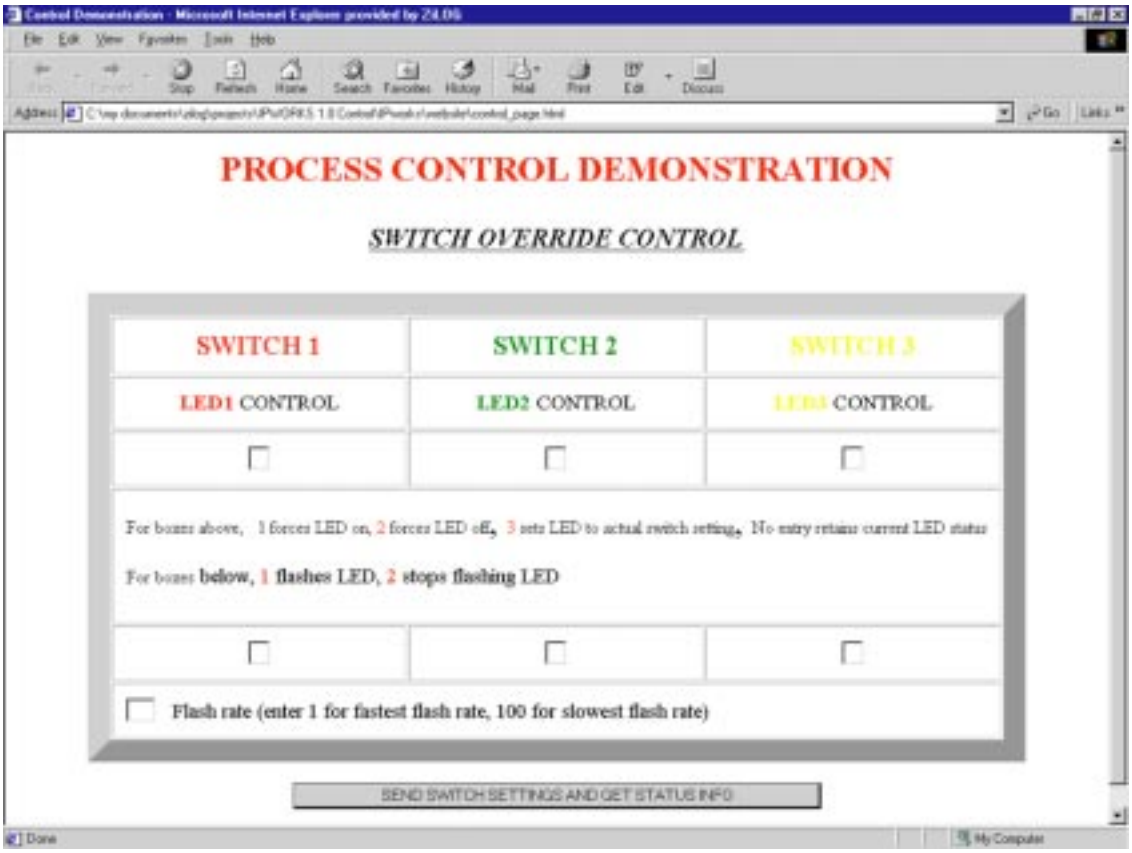


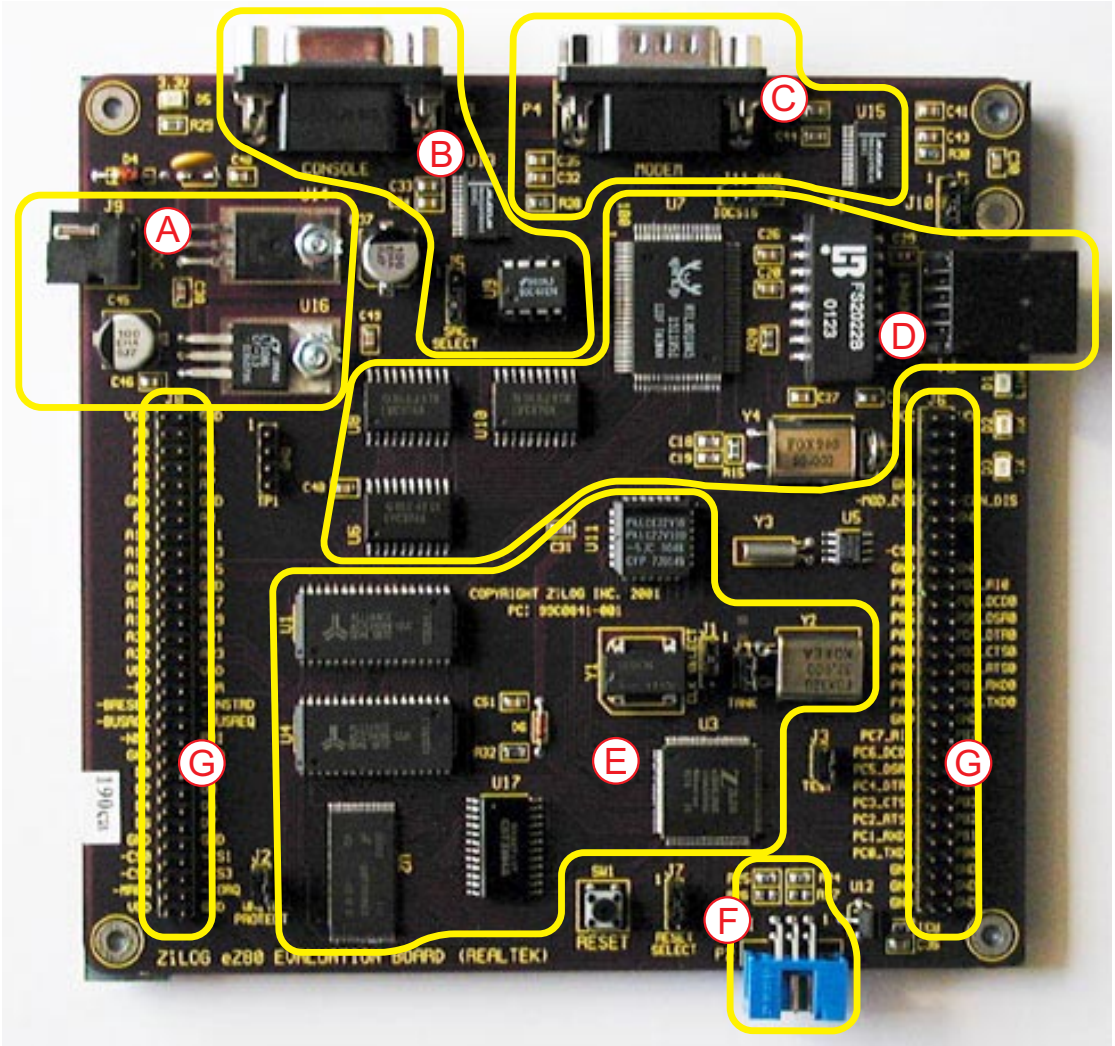
Figure 4. Process Control Demo

The HTML form handler uses the CGI script file **switches_cgi.c**. The code fragment, which is used to check the switch and LEDs, is listed below for reference:

```
/* check switch 1 and red LED1 status settings */
    io_work = (io_hold & sw1_in);
   str = http_find_argument(request,"switch1");
   str2 = http_find_argument(request,"switch1_flash");
```

Getting Started

The easiest way to start with the development process is to build the Thermostat Demo on your own. Building the Thermostat demo requires certain hardware and software. This section lists the hardware and software necessary to get started. One of the tools required is the eZ80190 Evaluation Board, which is shown in Figure 5.



Key to blocks A–F:

A. Power Supply	C. Serial Interface	E. CPU and Memory	G. Expansion Headers
B. Serial Interface	D. Ethernet Interface	F. ZDI Interface	

Figure 5. The eZ80190 Evaluation Board

Hardware Requirements

The hardware used for the Java Thermostat demonstration is listed below.

1. Thermostat daughterboard
2. eZ80190 Evaluation Board
3. ZPAKII

The Realtek RTL8019AS Ethernet Media Access Controller (EMAC) device manages Ethernet communications for the eZ80190 Evaluation Board. Figure 6 illustrates a more detailed block diagram, showing the Flash, SRAM, and EMAC blocks.

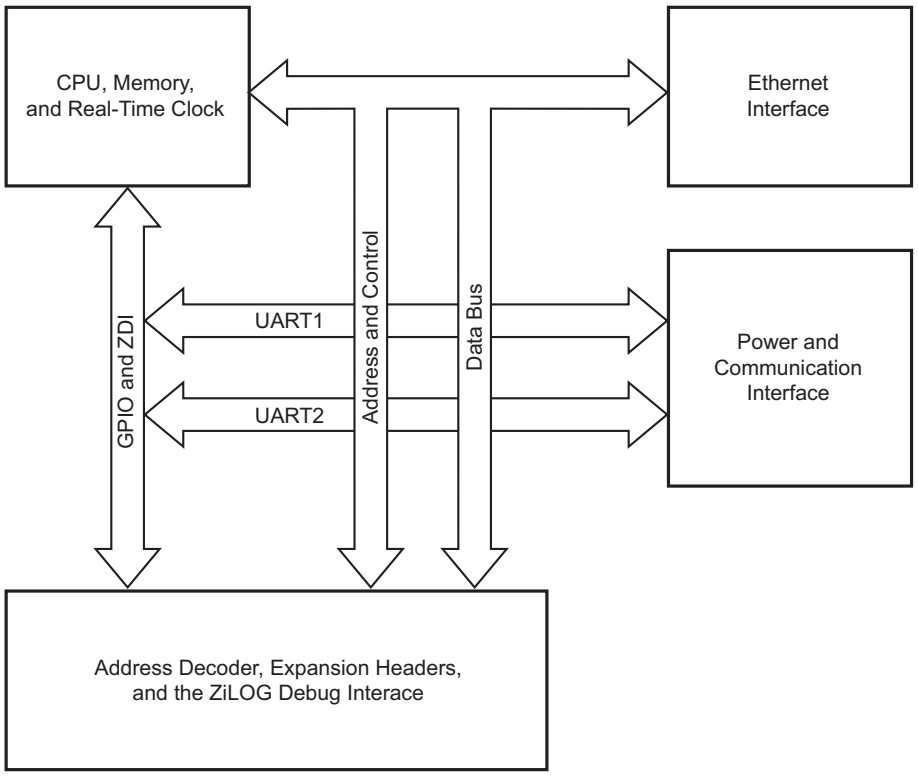


Figure 6. eZ80190 Evaluation Board Simple Block Diagram

Software Requirements

ZDSII version 3.66 supports the following:

- Real-time embedded operating system
- TCP/IP networking stack

- Micro-webserver
- ZPAKII C-Compiler v1.01
- Java JDK v1.3

The initial set-up process is performed as follows:

1. Connect the ZPAKII debug interface tool to a COM port on the host PC via a serial RS-232 cable.
2. Plug the Thermostat board onto the eZ80190 Evaluation Board via the J6 connector.

Load the Build file from the demo directory and experiment with the demo.

Converting Java and other Web Files

Before experimenting, the user an overview of how Java source files are compiled and linked with ZDSII is in order.

Java Build Process

To compile Java Source files, a source program must be compiled using Java into a class file. A class file is similar to an executable image file. However, instead of a specific machine code, a class file contains instructions for the Java Virtual Machine (JVM). Each platform that supports Java supplies a JVM program that essentially emulates the implied Java microprocessor.

How a source program is transformed into a Java class file depends on the Java development environment being employed. If using Sun's JDK (JDK v1.2 or above), issue the command:

```
javac Thermostat.java
```

Note: For other Java development environments such as JBuilder, Visual Café, etc., please see the appropriate product documentation for compiling Java source files.

The Thermostat.java file is available in the ThermostatDemo directory.

Java Output Class Files

The generated output files are Thermostat.class, MiniHttpClient.class, and Thermometer.class.

All Java class, HTML, and image files must be converted into C source files before linking them with the `main.c` program. Class files must be transformed into image files that can be downloaded to ZPAKII.

Thermostat.class, Tstat-ttpClient.class, Thermometer.class, and Thermostat-Demo.html files are added to the Project directory under the Web files section of the Thermostat Demo project.

To run the demo program follow the instructions given in the Readme file. A brief instruction follows here.

1. Create the ThermostatDemo.html and embed the applet tag in it.
2. Create an HTML file called ThermostatDemo.html and provide the applet tag as shown in the following example:

```
<HTML>
<TITLE>Thermostat Java</TITLE>
<body bgcolor="#ffffff"> <div align="center">
<h1>Thermostat Demo</h1> <BODY BGCOLOR=#FFFFFF> <CENTER>
<applet code="Thermostat.class" width=400 height=600 >
</applet </HTML>
```

3. Convert the Java class files and HTML files to C files.

Java in Embedded Applications

A very broad variety of embedded systems have already started to benefit from Java capability. These types of embedded systems include:

- Control networks used in many embedded industrial control applications.
- Remote management systems based on SNMP or HTTP and using Java applets to gather information about the performance of a particular node.
- Internet Appliances and other Consumer Devices
  - STB
  - Digital TV
  - PDA
- Telecommunications Industry
  - Cellular equipment
  - Programmable terminals
  - Intelligent modems, xDSL
- Data Communication Industry
  - Switches, routers, intelligent hubs
- Control and Automation

  - Factory automation
  - Street traffic control systems
- Automotive Industry
  - Navigation aids
  - Computerized control
- Vertical Markets
  - Lottery systems
  - Ticketing systems
  - Security systems

For manufacturers of embedded devices, the development of extensible, portable and downloadable applets and applications dramatically reduces development costs and time-to-market. By employing embedded applications, manufacturers can continue using commercial and proprietary real-time kernels.

Program Listings

Java Client Program

```
 /****************************** Thermostat. java Version 0.4

Authors:

Rajiv I. Pujar 10/10/2001

Staff Application Engineer, ZiLOG, Inc.

Denny Hopp

Field Application Engineer, ZiLOG, Inc.

Joshua Nekl (original)

This applet is for use with the eZ80/Z8 SDK Thermostat project.

The thermostat applet makes the query to the Thermostat Board and gets the
temperature values. To obtain updated values, an HTTP client has been built
in.

This program makes standard HTTP.cgi queries to the eZ80 to obtain updated
information and also to  modify any of the setpoints using buttons.

This applet draws a thermometer and also displays the values for current tem-
perature, upper setpoint, and lower setpoint.


/*****************************************************************/

import java.applet.applet;
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.text.*;

/*****************************************************************/ public
class Thermostat extends applet implements ActionListener {
Image iBuffer;
Graphics gBuffer;
TstatHttpClient client
  Thermometer thermometer
  Label tempLabel;
  Label maxLabel
  Label minLabel;
  Label tempValue;
  Label maxValue;
  Label minValue;
  Button maxButtonUp;
  Button maxButtonDown;
```

```
  Button minButtonUp;
  Button minButtonDown;
  GridBagConstraints gbc;
  GridBagLayout gbl;
  final int absolutemax=55;
  final int absolutemin=25;
public void init() {
  Font f;
  gbc=new GridBagConstraints()
       gbl=new GridBagLayout()
       setLayout(gbl);
       setBackground(Color.white);
       thermometer=new Thermometer((float)(absolutemin),this);
       maxButtonDown=new Button("DECREASE UPPER")
       maxButtonUp=new Button("INCREASE UPPER")
       minButtonDown=new Button("DECREASE LOWER")
       minButtonUp=new Button("INCREASE LOWER");
       tempLabel=new Label("Temperature (C): ")
       maxLabel=new Label("Upper Setpoint: ")
       minLabel=new Label("Lower Setpoint: ");
       tempValue=new Label("   ");
       maxValue=new Label("   ")
       minValue=new Label("   ");
       f=new Font("dialog",Font.BOLD,12)
       maxButtonDown.setFont(f)
       maxButtonUp.setFont(f)
       minButtonDown.setFont(f);
       minButtonUp.setFont(f);
       f=new Font("dialog",Font.PLAIN,12)
       tempLabel.setFont(f)
       maxLabel.setFont(f)
       minLabel.setFont(f)
       tempValue.setFont(f)
       maxValue.setFont(f);
       minValue.setFont(f);
       maxButtonDown.addActionListener(this)
       maxButtonUp.addActionListener(this)
       minButtonDown.addActionListener(this)
       minButtonUp.addActionListener(this);
       maxButtonDown.setActionCommand("maxdown")
       maxButtonUp.setActionCommand("maxup")
       minButtonDown.setActionCommand("mindown");
       minButtonUp.setActionCommand("minup");
       gbc.fill=GridBagConstraints.BOTH;
       place(tempLabel,0,1,1,1,0,0);
       place(tempValue,1,1,1,1,0,0);
       place(maxLabel,0,2,1,1,0,0);
       place(maxValue,1,2,1,1,0,0);
```

```
        place(maxButtonDown,2,2,1,1,0,0);
        place(maxButtonUp,3,2,1,1,0,0);
        place(minLabel,0,3,1,1,0,0);
        place(minValue,1,3,1,1,0,0);
        place(minButtonDown,2,3,1,1,0,0);
        place(minButtonUp,3,3,1,1,0,0);
        place(thermometer,2,0,1,1,0,0);
        client=new TstatHttpClient(getCodeBase().getHost());
        client.addActionListener(this);
        client.start();
        iBuffer=createImage(getSize().width, getSize().height);
        gBuffer=iBuffer.getGraphics();
}
  void place(Component c, int x, int y, int w, int h, float wx, float wy) {
        gbc.gridx=x
        gbc.gridy=y;
        gbc.gridwidth=w;
        gbc.gridheight=h;
        gbc.weightx=wx;
        gbc.weighty=wy;
        add(c,gbc); }
public boolean isDoubleBuffered() {
        return true;
}
  // stop http client public void destroy() {
        client.running=false
        client=null; }
// double buffer image to prevent flickering
public void update(Graphics g) {
  gBuffer.setClip(g.getClip());
  gBuffer.clearRect(0,0,getSize().width,getSize().height);
  paintComponents(gBuffer);
  g.drawImage(iBuffer,0,0,null);
}
public void paintComponents(Graphics g) {
  gBuffer.setClip(g.getClip());
  gBuffer.clearRect(0,0,getSize().width,getSize().height);
  super.paintComponents(gBuffer);
  paint(gBuffer);
  g.drawImage(iBuffer,0,0,null);
}
public void paint(Graphics g) {
  super.paint(g);
  if(client != null) {
        if(client.status == 0) {
             tempLabel.setForeground(Color.black);
             maxLabel.setForeground(Color.black);
             minLabel.setForeground(Color.black);
```

```
            tempValue.setForeground(Color.black);
            if(client.newmax == 0) {
            maxValue.setForeground(Color.black);
            } else {
            maxValue.setForeground(Color.red); }
            if(client.newmin == 0) {
            minValue.setForeground(Color.black);
            } else {
            minValue.setForeground(Color.red);
            }
  // if we lost connection to SDK, color everything red
  } else if(client.status == 1) {
            tempLabel.setForeground(Color.red);
            maxLabel.setForeground(Color.red);
            minLabel.setForeground(Color.red);
            tempValue.setForeground(Color.red);
            maxValue.setForeground(Color.red);
            minValue.setForeground(Color.red);
  // if we lost connection to eZ80, color everything lightGray
  } else {
            tempLabel.setForeground(Color.lightGray);
            maxLabel.setForeground(Color.lightGray);
       minLabel.setForeground(Color.lightGray);
       tempValue.setForeground(Color.lightGray);
       maxValue.setForeground(Color.lightGray);
       minValue.setForeground(Color.lightGray);
  // if our http client died, color everything yellow
  } else {
       tempLabel.setForeground(Color.yellow);
       maxLabel.setForeground(Color.yellow);
       minLabel.setForeground(Color.yellow);
       tempValue.setForeground(Color.yellow);
       maxValue.setForeground(Color.yellow);
       minValue.setForeground(Color.yellow);
  }
}
// handle any buttons being pressed or http updates
public void actionPerformed(ActionEvent e) {
  if(client != null) {
       synchronized(client) {
            if(e.getActionCommand()
                 compareTo("maxdown") == 0) {
            if(client.newmax == 0)
                 client.newmax=client.max;
            {
            client.newmax--;

            if(client.newmax < client.min) {
```

```
                client.newmax=client.min;                    }
            if(client.newmax < absolutemin) {
                client.newmax=absolutemin;
            }
            maxValue.setText(Integer.

toString(client.newmax));
            client.notify();
        } else if(e.getActionCommand().

compareTo("maxup") == 0) {
            if(client.newmax == 0) {

client.newmax=client.max;
            }
            client.newmax++;
            if(client.newmax > absolutemax) {
                 client.newmax=absolutemax;
            }
            if(client.newmax < absolutemin)
                client.newmax=absolutemin;
            }
            maxValue.setText(Integer.
                 toString(client.newmax));
            client.notify();
        } else if(e.getActionCommand().
                              compareTo("mindown") == 0)
            if(client.newmin == 0)
                client.newmin=client.min;
                              }
                 client.newmin--;
            if(client.newmin < absolutemin) {
                client.newmin=absolutemin;
        }
        minValue.setText(Integer.
            toString(client.newmin));
        client.notify();
        } else if(e.getActionCommand().
                  compareTo("minup") == 0) {
            if(client.newmin == 0) {
                  client.newmin=client.min;
        }
        client.newmin++;
        if(client.newmin > client.max) {
            client.newmin=client.max;
        }
        if(client.newmin < absolutemin) {
                  client.newmin=absolutemin;
```

```
}       if(client.newmin > absolutemax){
                              client.newmin=absolutemax;
        }
        minValue.setText(Integer.
                  toString(client.newmin));
        client.notify();
} else if(e.getActionCommand().
  compareTo("httpupdate") == 0) {
  if(client.newmin == 0) {
            minValue.setText(Integer.
            toString(client.min));
 }else {
            minValue.setText(Integer.
            toString(client.newmin));
        }
            if(client.newmax == 0) {
                  maxValue.setText(Integer.
                              toString(client.max));
} else {
                maxValue.setText(Integer.
                toString(client.newmax));
        }
            //tempValue.setText(Integer.
                  toString(client.temperature));
            tempValue.setText(client.labelupdate));
            thermometer.setTemp((float)(client.
                              temperature));             } } }
        repaint(); } }

/*************************************************************  This
class acts as the http client and it creates a thread to query the eZ80 sdk *
every 2 seconds. This class establishes Socket connection and will update *
the lower/upper setpoints and reads the dynamic temperature values.
*************************************************************/
class TstatHttpClient extends Thread {
  public boolean running;
  public int temperature;
  public int max;
  public int min;
  public int newmax;
  public int newmin;
  public int status;
  ActionListener listener;
  final String actionCommand=new String("httpupdate");
  ActionEvent e;
  String server;
  Socket s;
  StringBuffer response;
```

```
  URL url;
  String protocol = "http"
  URL Connection connection;
  String InputLine;
  DataInputStream inStream;
  String templabel;
  String labelupdate;

  TstatHttpClient(String newServer) {
       server=newServer;
       e=new ActionEvent(this,0,actionCommand);
                temperature=0;
                max=1;
       min=1;
       newmax=0;
       newmin=0;
       status=2;
       running=true;
}
public void addActionListener(ActionListener l) {
  if(l == null) {
       return;
  }
  listener=AWTEventMulticaster.add(listener,l);
}
public void removeActionListener(ActionListener l) {
  if(listener == null) {
       return;
}
listener=AWTEventMulticaster.remove(listener,l);
}
  public void run() {
       do {
            getURL();
       getupdate();
       if(listener != null) {  listener.actionPerformed(e);
  }
       synchronized(this) {
            try {
                              wait(2000);
       }    catch(Exception e) {
                 System.out.println(e);
                 break;
            }
       }
  } while(running);
}
```

```
  public int getupdate() {
  try {
       s=new Socket(server,80);
  } catch (Exception e) {
       System.out.println("eZ80 disconnected...Socket() failed");
       status=2;
       return -1;
  }
  if( sendRequest(s) != 0) {
       try {
            s.close();
            s=null;
       } catch(Exception e) {
            System.out.println(e); }                         status=2; return
-1;
}
  response = new StringBuffer();
       if( getResponse(s) != 0) {
       try {
                                                             s.close();
            s=null;
       } catch(Exception e) {
            response=null;
            System.out.println(e);
  }            status=2;                    return -1;    }
       try {
            s.close();
            s=null;
  } catch(Exception e) {
       System.out.println(e);
  }
  parseData();
  response = null;
  if(newmax == max) {
       newmax=0;
  }
  if(newmin == min) {
       newmin=0; }
  return 0;
}
int sendRequest(Socket s) {
  StringBuffer request=null;
  OutputStream out=null;
  System.out.println("requesting update...");
  request=new StringBuffer();
  request.append("GET /data.html");
  if(newmax == max) {
       newmax=0;
```

```
}
  if(newmin == min) {
       newmin=0;
}
if(newmax != 0 || newmin != 0) {
  System.out.println("updating...");
  request.append("?");
  if(newmax != 0) {
       System.out.println(
                 "new Upper_Set = " +
            Integer.toString(max));
       request.append("max=");
       request.append(Integer.toString(newmax));
       if(newmin != 0) {
            request.append("&");
       newmax=0;
       }
  }
  if(newmin != 0) {
       System.out.println(
                  "new Lower_Set = " +
            Integer.toString(min));
       request.append("min=");
       request.append(Integer.toString(newmin));
       newmin=0;
  }
}
request.append(" HTTP/1.0\r\nHost: server\r\n\r\n");
try {
  out=s.getOutputStream();
} catch (Exception e) {
  System.out.println(e);
  return -1;
}
try {
  out.write(request.toString().getBytes());
catch (Exception e) {
  System.out.println(e);
  eturn -1;
}
request=null;
out=null;
return 0;
}
int getResponse(Socket s) {
  InputStream in=null;
  int c;
```

```
  char ch;
  byte responsestatus;
  responsestatus=0;
  try {
       in=s.getInputStream();
} catch (Exception e) {
       System.out.println(e);
       return -1;
}
  do {
       try {
            c=in.read();
       } catch(Exception e) {
            System.out.println(e);
            return -1;
}
  if(c == -1) {
       break;
}
  if(status == 4) {
       ch=(char)c;
       response.append(ch);
       } else if(c == '\r' &&
                 (responsestatus==0 || responsestatus==2)) {
            status++;           }
       } else if(c == '\n' &&
            responsestatus==1 || responsestatus==3)) {
            responsestatus++;
       } else {
            responsestatus=0;
       }
  } while(c != -1);
  in=null;
  System.out.println("received response...");
  return 0;
}
int parseData() {
  int begin;
  begin=0;
  synchronized(this) {
       tatus=2;
       do {
                 if(response.toString().substring(begin)
                  startsWith("Ambient_Temp")) {
                  temperature=
                  getValueFromString(response.
                  toString().substring(begin));
                 status=0;
```

```
            } else if(response.toString().
                substring(begin).
                startsWith("Lower_Set") ) {
                min=getValueFromString(response.
                 toString().substring(begin));
                status=0;
            } else if(response.toString().
                substring(begin).
                startsWith("Upper_Set") ) {
                max=getValueFromString(response.
                 toString().substring(begin));              status=0;
            } else if(response.toString().
                substring(begin).
                startsWith("Disconnected") ) {
                 status=1;
            System.out.println(
                "SDK disconnected...update failed");
            break;
        }
                begin=response.toString().indexOf('\n',begin);
        begin++;
    } while(begin > 0);
    if(status == 2) {
        System.out.println("Connected and received response, but no data");
    }
    return 0;
}
int getValueFromString(String str) {
    int start;
    int end;
    int value;
    end=str.indexOf('\r');
    start=str.indexOf('=');
    if(end != -1 && start > end) {
            return 0;
    }
    try {
        while(! Character.isDigit(str.charAt(start))) {
            start++;
        }
            end=start;
            while(Character.isDigit(str.charAt(end))) {
            end++;
            if(end > str.length()) {
                break;
            }
        }
```

```
  } catch(Exception e) {
       System.out.println(e);
  }
  value=Integer.parseInt(str.substring(start,end));
  return value;
   }
/*This method reads the temperature values from the Thermostat Board*/
public void getURL(){
  try{
  url = new URL("http://192.168.1.1/Data.txt");
  connection = url.openConnection();
  InStream = new DataInputStream
(connection.getinputStream());
  while (null !=(inputLine = inStream.readLine())){

  templabel=inputLine;
  labelupdate=templabel
  temperature-integer.parselnt(labelupdate);
  inStream.close();
}
  }
  catch(Exception e){}
}// end getURL
}
/*****************************************************************
* This class is responsible for drawing the thermometer                *
*****************************************************************************
/
class Thermometer extends Component {
  private float temp;
  private float displayTemp;
  private final int columnWidth=40;
  private final int columnHeight=145;
  private final int bulbWidth=70;
  private final int areaRound=20;
  private final int paddx=1; private final int paddy=1;
  private final float maxTemp=(float)55.0;
  private final float minTemp=(float)25.0;
  private final int labelInterval=3;
  private final int tickInterval=5;
  Thermometer(float newTemp,Component parent) {
       setTemp(newTemp);
       setSize(paddx*2+bulbWidth,
            paddy*2+columnHeight+bulbWidth/2+columnWidth/2);
  }
```

```
    public void setTemp(float newTemp) {
         temp=newTemp;
         displayTemp=temp;
         if(displayTemp > maxTemp) {
              displayTemp=maxTemp;
         }
         if(displayTemp < minTemp) {
              displayTemp=minTemp;
         }
         repaint();
    }
    public Dimension getSize() {
         return new Dimension(paddx*2+bulbWidth,
              paddy*2+columnHeight+bulbWidth+bulbWidth/2+columnWidth/2);
    }
    public Dimension getMinimumSize(){
         return getSize();
    }
    public Dimension getPreferredSize() {
         return getMinimumSize();
    }
    public void paint(Graphics g) {
         int i;
         int x;
         int y;
         double z;
// draw thermometer outline
g.setColor(Color.black);
g.fillRect(paddx+bulbWidth/2-columnWidth/2,
   paddy+columnWidth/2,
   columnWidth,
   columnHeight+bulbWidth/2);
g.fillOval(paddx+bulbWidth/2-columnWidth/2,
   paddy,
   columnWidth,
   columnWidth);
g.fillOval(paddx,
   paddy+columnHeight+columnWidth/2,
   ulbWidth,
   bulbWidth);
//fill in thermometer leaving 2 pixels black around edges
g.setColor(Color.white);
g.fillRect(paddx+bulbWidth/2-columnWidth/2+2,
   paddy+columnWidth/2,
   columnWidth-4,
   columnHeight+bulbWidth/2);
g.fillOval(paddx+bulbWidth/2-columnWidth/2+2,
   paddy+2,
```

```
  columnWidth-4,
  columnWidth-4);
g.setColor(Color.red);
g.fillOval(paddx+2,
  paddy+columnHeight+columnWidth/2+2,
  bulbWidth-4,
  bulbWidth-4);
// fill column based on temp
y=(int)((displayTemp-minTemp)*columnHeight/(maxTemp-minTemp));
g.fillRect(paddx+bulbWidth/2-columnWidth/2+2,
  paddy+columnWidth/2+columnHeight-y,
  columnWidth-4,
  y+bulbWidth/2);
//draw a white pinstripe
g.setColor(Color.white);
g.fillRect(paddx+bulbWidth/2+columnWidth/2-5,
  paddy+columnWidth/2,
  2,
  columnHeight);
//make the bottom bulb shine
g.drawArc(paddx+5,
  paddy+columnHeight+columnWidth/2+4,
  bulbWidth-10,
  bulbWidth-10, 0,80);
// draw tick marks & labels g.setColor(Color.black);
g.setFont(new Font("dialog",Font.PLAIN,10));
for(i=0, z=minTemp;
  i<=(labelInterval)*tickInterval;
  i++,
  z=(float)i/((labelInterval)*tickInterval)*
  (maxTemp-minTemp)+minTemp) {
  y=(int)(paddy+columnHeight+columnWidth/2-
  float)i/((labelInterval)*tickInterval)*columnHeight);
  if(i%tickInterval == 0) {
       g.drawLine(paddx+bulbWidth/2+columnWidth/2+4,
       y,
       paddx+bulbWidth/2+columnWidth/2+9,
       y);
  g.drawString(NumberFormat.getInstance().format(z),
       paddx+bulbWidth/2+columnWidth/2+11,
       y+4);
  } else {
                g.drawLine(paddx+bulbWidth/2+columnWidth/2+4,
                                                y,
       paddx+bulbWidth/2+columnWidth/2+6,
       y);
       }
```

```
  }
}
}
```

Thermostat Firmware Program

main.c

```
#include <kernel.h
#include <stdlib.h>
#include <http.h>
#include <bootinfo.h>
#include <shell.h>
#include <netapp.h>
#include <lcd_lib.h>

/*  * Default Network Configuration  */

struct BootInfo Bootrecord = {
"192.168.1.1", /* Default IP address */
"10.0.0.1",/* Default Gateway */
"10.0.0.1",/* Default Timer Server */
"10.0.0.1",/* Default File Server - Not currently Used */        "",
"10.0.0.1",/* Default Name Server */
"",
0xffffff00UL/* Default Subnet Mask */
   };

/*
* Web Site Description
*/

/*
* If a page is a static page, then the type HTTP_PAGE_STATIC should be
* specified, and a struct staticpage used to describe the page data. If  * a
page is dynamic, then the type HTTP_PAGE_DYNAMIC should be used, and  * the
address of the CGI routine should be provided.  */

// HTML pages
extern struct staticpage demo_htm;
extern struct staticpage htmlpost_htm;
extern struct staticpage htmlget_htm;
extern struct staticpage javaapplet_htm;
extern struct staticpage javascript_htm;
extern struct staticpage products_htm;
extern struct staticpage siteinfo_htm;
extern struct staticpage webcam_htm;
extern struct staticpage zoffices_htm;
extern struct staticpage jcontrol_page_html;
extern struct staticpage control_page_html;
extern struct staticpage ThermostatDemo_html;
```

```
// Pictures: JPG, GIF extern struct staticpage aqua_bar1_gif;
extern struct staticpage ez80banner_jpg;
extern struct staticpage ez80chip_jpg;
extern struct staticpage ez80logo_gif;
extern struct staticpage pioneer_banner_jpg;
extern struct staticpage metro_gif;
extern struct staticpage zilog_jpg;
// Applets
extern struct staticpage messagerA_class;
extern struct staticpage JavaClock_class;
extern struct staticpage AnalogClock_class;
extern struct staticpage CustomParser_class;
extern struct staticpage ParamParser_class;
extern struct staticpage Buttonapplet_class;
extern struct staticpage LEDBulb_class;
extern struct staticpage MiniHttpClient_class;
extern struct staticpage TstatHttpClient_class;
extern struct staticpage Thermostat_class;
extern struct staticpage Thermometer_class;
// CGIs
extern int index_cgi(int);
extern int add_cgi(int);
extern int data_cgi(int);
extern int input_cgi(int);
extern int Thermostat_cgi(int);


//LCD function
extern int lcdinitpid=0x00;
extern voidlcdinit(void);

//global declarations

unsigned char flash_mask,jflash_mask;
unsigned char flash_rate, flash_speed_entry,jflash_rate,jflash_speed_entry;
unsigned char LED1_status,LED2_status,LED3_status,jout_hold;
int ambient_temp,upper_setpoint,lower_setpoint,j;
char temp_rising,bypass_counter;
unsigned char io_hold,io_work;
long delay_count=0;
int temp;
int temp_low_byte,temp_high_byte,temp_degrees_f,temp_degrees_c; int
i2c_shiftreg,i2c_count,i2c_error,ok;

Webpage website[] = {
/* 3 different ways of specifying the default web page */
{HTTP_PAGE_DYNAMIC, "/", "text/html", index_cgi },
{HTTP_PAGE_DYNAMIC, "/default.htm", "text/html", index_cgi },
{HTTP_PAGE_DYNAMIC, "/index.htm", "text/html", index_cgi },
```

```
{HTTP_PAGE_DYNAMIC, "/cgi-bin/add", "text/html", add_cgi },
{HTTP_PAGE_DYNAMIC, "/Data.html", "text/html", input_cgi },
{HTTP_PAGE_DYNAMIC, "/Data.html", "text/html", data_cgi },
{HTTP_PAGE_DYNAMIC, "/Thermostat.html", "text/html", & Thermostat_cgi },

{HTTP_PAGE_DYNAMIC, "/cgi-bin/ml_reflector","text/plain",{reflect_cgi}},
{HTTP_PAGE_DYNAMIC, "/cgi-bin/ml_replacer","text/plain",{replace_cgi}},
{HTTP_PAGE_DYNAMIC, "/cgi-bin/switches", "text/html", switches_cgi },
{HTTP_PAGE_DYNAMIC, "/cgi-bin/java_control","text/html",java_control_cgi},

/* Java classes */

{HTTP_PAGE_STATIC, "/messagerA.class", "application/octect-stream",
&messagerA_class },
{HTTP_PAGE_STATIC, "/JavaClock.class", "application/octect-stream",
&JavaClock_class },
{HTTP_PAGE_STATIC, "/AnalogClock.class", "application/octect-stream",
&AnalogClock_class },
{HTTP_PAGE_STATIC, "/CustomParser.class", "application/octect-stream",
&CustomParser_class },
{HTTP_PAGE_STATIC, "/ParamParser.class", "application
octect-stream", &ParamParser_class
},

{HTTP_PAGE_STATIC, "/Buttonapplet.class","application/octet-stream",
&Buttonapplet_class },
{HTTP_PAGE_STATIC, "/MiniHttpClient.class","application/octet-stream",
&MiniHttpClient_class },
{HTTP_PAGE_STATIC, "/LEDBulb.class","application/octet-stream",
&LEDBulb_class },
{HTTP_PAGE_STATIC, "/Thermostat.class","application/octet-stream",
&Thermostat_class },
{HTTP_PAGE_STATIC, "/Thermometer.class","application/octet-stream",
&Thermometer_class },
{HTTP_PAGE_STATIC, "/TstatHttpClient.class","application/octet-stream",
&TstatHttpClient_class },

/* Html files to embed Java applets, HTML forms and other Image files*/

{HTTP_PAGE_STATIC, "/jcontrol_page.html","text/html",
&jcontrol_page_html },
{HTTP_PAGE_STATIC, "/control_page.html","text/html", &control_page_html },
{HTTP_PAGE_STATIC, "/ThermostatDemo_.html","text/html",
&ThermostatDemo_html },

{HTTP_PAGE_STATIC, "/demo.htm", "text/html", &demo_htm },
{HTTP_PAGE_STATIC, "/htmlpost.htm", "text/html", &htmlpost_htm },
{HTTP_PAGE_STATIC, "/htmlget.htm", "text/html", &htmlget_htm },
{HTTP_PAGE_STATIC, "/javaapplet.htm", "text/html", &javaapplet_htm },
{HTTP_PAGE_STATIC, "/javascript.htm", "text/html", &javascript_htm },
```

```
{HTTP_PAGE_STATIC, "/products.htm", "text/html", &products_htm },
{HTTP_PAGE_STATIC, "/siteinfo.htm", "text/html", &siteinfo_htm },
{HTTP_PAGE_STATIC, "/webcam.htm", "text/html", &webcam_htm },
{HTTP_PAGE_STATIC, "/zoffices.htm", "text/html", &zoffices_htm },

{HTTP_PAGE_STATIC, "/aqua_bar1.gif", "image/gif", &aqua_bar1_gif },
{HTTP_PAGE_STATIC, "/ez80banner.jpg", "image/jpg", &ez80banner_jpg },
{HTTP_PAGE_STATIC, "/ez80chip.jpg", "image/jpg", &ez80chip_jpg },
{HTTP_PAGE_STATIC, "/ez80logo.gif", "image/gif", &ez80logo_gif },
{HTTP_PAGE_STATIC, "/pioneer_banner.jpg", "image/jpg", &pioneer_banner_jpg},
{HTTP_PAGE_STATIC, "/metro.gif", "image/jpg", &metro_gif },
{HTTP_PAGE_STATIC, "/zilog.jpg", "image/jpg", &zilog_jpg },
{HTTP_PAGE_DYNAMIC,
"/D12345678/D12345678/D12345678/
D12345678A123456789B123456789C123456789D123456789E123456789F123456789G123456
789H123456789I123456789J123456789K123456789L123456789M123456789N123456789O12
3456789P123456789Q123456789R123456789S123456789T123456789U123456789V12345678
9W123456789X123456789.html",
"text/plain",{index_cgi}},
{0, NULL, NULL, NULL }
};

main()
{
extern struct cmdent *allcmds;
extern int nallcmds;
int fd;

kprintf("\nSimple Webserver Demo\n");
/*
* If you want a telnet daemon running that will provide unprotected access
* to the system, call telnet_init() here.
*/
telnet_init(&allcmds, nallcmds);

/*
* To initialize a webserver, call http_init(). You need to pass in a few
* parameters, but the default values shown below provide a typical webserver.
* The webserver will create several threads to improve performance. You can
* also run multiple webserver on multiple ports.
*/ http_init(http_defmethods,httpdefheaders,website,80);

/* LCD  initialization process */

lcdinitpid = (create(lcdinit, 1024, 20, "lcdinit",0));
resume(lcdinitpid);

// Here is where our own functionality is inserted.  The while loop keeps
repeating
// our functions

// Initilaize eZ80  ports, timer registers and interrupts
```

```
reg_init_function();

while(1)
{
tstat_function();
}
/*
* To start up a shell on the serial port, call shell_init(), and pass it the
* device to use.
*/
open(SERIAL0, 0,0);
if ((fd=open(TTY, (char *)SERIAL0,0)) == SYSERR) {
kprintf("Can't open tty for SERIAL0\n");
return SYSERR;
}
kprintf("Starting up a shell on device %d\n", fd );
shell_init(fd,&allcmds,nallcmds);

}

//*********************************************************************

 lcdinit.c

/* This program provides display of IP address and text message as  "Z80-WEB-
SERVER" on the LCD Panel  */

#include <kernel.h>
#include <stdlib.h>
#include <httpd.h>
#include <stddefs.h
#include <lcd_lib.h>
#include <bootinfo.h>

extern const U8 string[] = "eZ80-WEBSERVER";
extern const U8 string1[] = "IP";
extern int lcdinitpid;
extern voidlcdinit(void);

voidlcdinit(void)
{
U8*pstring;

PortAinit();
LCDinit(M68_4bit,Array5_7,2,16,LCDcursorOff);
pstring = &string[0];
LCDwriteString(TextStandard,Line_1,2,pstring,strlen(pstring)); pstring =
&string1[0]; LCDwriteString(TextStandard,Line_2,1,pstring,strlen(pstring));
LCDwriteString(TextStan-
dard,Line_2,4,Bootrecord.myip,strlen(Bootrecord.myip));
suspend(lcdinitpid); }

CGI Interface Programs
```

```
/*****************************************************************
* input_cgi.c
*
* Rajiv I.Pujar
*
*
* Will update the max and min  setpoint values from the Java applet.
* Responsible for setting the upper setpoint, lower setpoint and for reading
the dynamic temperature.
*
*****************************************************************/


#include <kernel.h>
#include <stdlib.h>
#include <httpd.h>
#include <http.h>
#include <bootinfo.h>
#include <shell.h>
#include <netapp.h>


extern char vers[];
extern int readtemp();
int low ;
int high;


int input_cgi(struct http_request *request)
{
longnow;
//char str[80];
char tmpbuf[32];
char*str;
//int x,y;
unsigned char x,y;
int i,value ,ambient ;


http_output_reply(request,HTTP_200_OK);


str=http_find_argument(request,"min");
if( str ) {
x=atoi(str);
sprintf(tmpbuf,"<BR>Lower_Set=%d",x);
```

```
__http_write(request,tmpbuf,strlen(tmpbuf));
low=x;


// update upper setpoint
} else {
__http_write(request,"<B>Didn't find parameter 'minimum'</B>",33);
}


str=http_find_argument(request,"max");
if( str ) {
x=atoi(str);
sprintf(tmpbuf,"<BR>Upper_Set=%d",x);
__http_write(request,tmpbuf,strlen(tmpbuf));
high=x;


// update upper setpoint
} else {
__http_write(request,"<B>Didn't find parameter 'maximum'</B>",33);
}
return 0;
}
int getUppervalue(){
return high;
}
int getLowervalue(){
 return low;
}
data_cgi.c
/*****************************************************************
* data_cgi.c
*
* Rajiv I.Pujar
*
* Responsible for reading the dynamic temperature and to write to the Http
server
* Display the temperature on the LCD panel after converting the binary tem-
perature value into its ASCII representation.
*
```

```
*****************************************************************/

#include <kernel.h>
#include <stdlib.h>
#include <httpd.h>
#include <http.h>
#include <bootinfo.h>
#include <shell.h>
#include <netapp.h>
#include <lcd_lib.h>

extern void delay(int);
extern char vers[];
extern int readtemp();

//extern void delay(int);

int small,large,count,c ;

void bin_to_ascii(unsigned char c, char *buff);
//delay func

int data_cgi(struct http_request *request)
{
longnow;
//char str[80];
char tmpbuf[32];
char*str;
//intx,y;
int i,value ,ambient ;
unsigned char*pstring;

http_output_reply(request,HTTP_200_OK);
// Get ambient temp value and send that to applet
ambient = getAmbientTemp();
sprintf(tmpbuf,"%d",ambient);
__http_write(request,tmpbuf,strlen(tmpbuf));
bin_to_ascii(ambient,pstring);
LCDwriteString(TextStandard,Line_2,8,(pstring+1),2);

return 0;


}
/*****************************************************************/

// This will convert the binary value c into it's ascii representation

//     in hex. It will append the two ascii characters at the end of *buff

//     If you want to save it at the start of buff, make buff[0]='\0';

//     This function will also null terminate the string.
```

```
void bin_to_ascii(unsigned char c, char *buff)
{
unsigned charh,k;
for(h = 0;
c >= 100;
c -= 100,h++);
for(k = 0;
c >= 10;
c -= 10,k++);
*(buff+2) = ('0'+(c & 0x0f));
*(buff+1) = ((k & 0x0f) + '0');
*buff = ((h & 0x0f) + '0');
if(*buff == '0')
{
*buff = ' ';
if(*(buff+1) == '0')
{
*(buff+1) = ' ';
}
}
}
```

java_control_cgi.c

```
/*****************************************************************
java_control_cgi.c
*
* Programmer:  Denny Hopp
* FAE ZiLOG/Ohio
* 10/9/01
*
* This CGI script file checks the switch inputs and controls the LED's
* for the Java Control demo.
 *****************************************************************/
#include <stdlib.h>
#include <httpd.h>
#include <ez80.h>
#include <ez80def.h>
extern struct staticpage jcontrol_page_html;

int java_control_cgi(struct http_request *request)
{
char jtmpbuf[255];
char*jstr,*jstr2;
```

```
unsigned int jio_hold,jio_work;
unsigned charjLED1,jLED2,jLED3; /* variables that are passed from the Applet
*/
unsigned char
jswitch1_flash,jswitch2_flash,jswitch3_flash,jflash_speed_browser,flash_spee
d_browser;
unsigned char
jswitch1,jswitch2,jswitch3,jswitch1flash,jswitch2flash,jswitch3flash;

extern unsigned char jflash_rate,jflash_speed_entry,jflash_mask;
extern unsigned char flash_rate,flash_speed_entry,flash_mask;
extern unsigned char LED1_status,LED2_status,LED3_status,jout_hold;

#define bit0 1
#define sw1_in 1 /* switch 1 input for LED1*/

#define bit1 2
#define sw2_in 2 /* switch 2 input for LED2*/

#define bit2 4
#define sw3_in 4 /* switch 3 input for LED3 */

#define bit3 8

#define bit4 16
#define LED1_out_mask 16 /* LED1 output */
#define LED1_flash_mask 16

#define bit5 32
#define LED2_out_mask 32 /* LED2 output */
#define LED2_flash_mask 32

#define bit6 64
#define LED3_out_mask 64 /* LED3 output */
#define LED3_flash_mask 64
#define bit7 128

http_output_reply(request,HTTP_200_OK);

/* get the Port A register Status */

jio_hold = jio_work = (PB_DR &(sw1_in + sw2_in + sw3_in));

// get flash speed from browser entry

jstr = http_find_argument(request,"jflash_speed_browser");

jflash_speed_browser = atoi(jstr);

if((jflash_speed_browser >= 1) && (jflash_speed_browser <= 100))
{
jflash_speed_entry = flash_speed_entry = flash_speed_browser =
jflash_speed_browser;
}

jswitch1=jswitch2=jswitch3=jswitch1flash=jswitch2flash=jswitch3flash=0;

/* now let's check each physical switch setting and each forced setting */
```

```
/* check switch 1 and red LED1 status settings */
jstr = http_find_argument(request,"LED1");
jswitch1 = atoi(jstr);
if(jswitch1 == 1)
{
jout_hold &= ~LED1_out_mask;
PB_DR &= ~LED1_out_mask;
LED1_status=1;
}
else if(jswitch1 == 2)
{
jflash_mask &= ~LED1_flash_mask;
flash_mask = jflash_mask;
jout_hold |= LED1_out_mask;
PB_DR |= LED1_out_mask;
LED1_status=2;
}
jstr = http_find_argument(request,"LED2");
jswitch2 = atoi(jstr);
if(jswitch2 == 1)
{
jout_hold &= ~LED2_out_mask;
PB_DR &= ~LED2_out_mask;
LED2_status=1;
}
else if(jswitch2 == 2)
{
jflash_mask &= ~LED2_flash_mask;
flash_mask = jflash_mask;
jout_hold |= LED2_out_mask;
PB_DR |= LED2_out_mask;
LED2_status=2;}
jstr = http_find_argument(request,"LED3");
jswitch3 = atoi(jstr);
if(jswitch3 == 1)
{
jout_hold &= ~LED3_out_mask;
PB_DR &= ~LED3_out_mask;
LED3_status=1;
}
else if(jswitch3 == 2)
{
jflash_mask &= ~LED3_flash_mask;
flash_mask = jflash_mask;
jout_hold |= LED3_out_mask;
PB_DR |= LED3_out_mask;
```

```
LED3_status=2;
}
jstr = http_find_argument(request,"flashcontrolLED1");
jswitch1flash = atoi(jstr)
if(jswitch1flash == 1)
{
if((jout_hold & LED1_out_mask) == 0)
{
jflash_mask ^= LED1_flash_mask;
flash_mask = jflash_mask;
PB_DR &= ~LED1_out_mask;
}
}
jstr = http_find_argument(request,"flashcontrolLED2");
jswitch2flash = atoi(jstr);
if(jswitch2flash == 1)
{
if((jout_hold & LED2_out_mask) == 0)
{
jflash_mask ^= LED2_flash_mask;
flash_mask = jflash_mask;
PB_DR &= ~LED2_out_mask;
}
}
jstr = http_find_argument(request,"flashcontrolLED3");
jswitch3flash = atoi(jstr);
if(jswitch3flash == 1)
{
if((jout_hold & LED3_out_mask) == 0)
{
jflash_mask ^= LED3_flash_mask;
flash_mask = jflash_mask;
PB_DR &= ~LED3_out_mask;
}
}
// send the switch status info to the browser
//sprintf(jtmpbuf,"\nSwitch_Snapshot=%d\r",jio_hold);

//
sprintf(jtmpbuf,"\nSwitch_Snapshot=%d\r\nFlash_Rate_Set=%d\r\nLED1_Set=%d\r\
nLED2_Set=%d\r\nLED3_set=%d\r\nLED1_Flash_Set=%d\r\nLED2_Flash_Set=%d\r\nLED
3_Flash_Set=%d\r",

//jio_hold,jflash_rate,LED1_status,LED2_status,LED3_status,10,10,10);

sprintf(jtmpbuf,"\r\nFlash_Rate_Set=%d\r\nLED3_Set=%d\r\nLED1_Set=%d\r\nLED2
_Set=%d\r\nLED1_Flash_Set=%d\r\nLED2_Flash_Set=%d\r\nLED3_Flash_Set=%d\r\nSw
itch_Snapshot=%d\r\n",
```

```
jflash_speed_browser,LED3_status,LED1_status,LED2_status,10,10,10,jio_hold);

__http_write(request,jtmpbuf,strlen(jtmpbuf));
return 0;}
```

Web Files

ThermostatDemo.html

```
<HTML>
<HEAD>
  <!-- Created with AOLpress/2.0 -->
  <TITLE>Thermostat HTML Page</TITLE>
</HEAD>
<BODY>
<P ALIGN=Center>
<IMG SRC="ez80banner.jpg" WIDTH="574" HEIGHT="139">
<P ALIGN=Center>
<FONT COLOR="1111ff"> THERMOSTAT CONTROLLED BY JAVA APPLET </FONT>
<P ALIGN=Center>
<APPLET CODE="Thermostat.class" WIDTH="600" HEIGHT="340" ALIGN="Top"></APPLET>
<P ALIGN=Center>
<FONT COLOR="#004000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>PRESS
</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#ff0000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>BUTTON
1
</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#004000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>TO
FORCE HEATING,
</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#ff0000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>BUTTON
2</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#004000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>
TO FORCE COOLING, OR
</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#ff0000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>BUTTON
3</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT><FONT
    COLOR="#004000"><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL><SMALL>
TO SHUT OFF HEATING AND
COOLING</SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></SMALL></FONT>
</BODY></HTML>
```