*Application Note*

# *Thermostat Demo Using the eZ80Acclaim!™ MCU*

AN016804-0504

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

# List of Tables

# Abstract

The Thermostat Demo illustrates how easy it is to develop Internet-enabled process control and monitoring applications by embedding Java elements on an eZ80Acclaim!™ microcontroller. A variety of languages are used in the design and specification of hardware and software for embedded electronic systems. The integration of hardware and software components for this Demo is implemented using dissimilar languages. The Thermostat drivers described in this Application Note are written in the C language. A Graphical User Interface (GUI) for the embedded system is developed using Java applets. The Thermostat Demo is thus 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.

The source code file (saved in the WinZip format) associated with this Application Note is `AN0168-SC01.zip`, and is available on the ZiLOG website.

# ZiLOG Product Overview

This section contains brief overviews of the ZiLOG products used in this Application Note, which includes the award-winning eZ80Acclaim!™ microcontrollers and the full-featured ZiLOG TCP/IP software suite.

## eZ80Acclaim!™ MCU Family Overview

The eZ80Acclaim!™ family of microcontrollers includes Flash and non-Flash products. The Flash-based eZ80Acclaim!™ MCUs, device numbers eZ80F91,eZ80F92, and eZ80F93, are an exceptional value for customers designing high performance embedded applications. With speeds up to 50MHz and an on-chip Ethernet MAC (eZ80F91 only), designers have the performance necessary to execute complex applications supporting networking functions quickly and efficiently. Combining on-chip Flash and SRAM, eZ80Acclaim!™ devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

ZiLOG also offers two eZ80Acclaim!™ devices without Flash memory: the eZ80L92 and eZ80190 microprocessors.

## ZiLOG TCP/IP Software Suite Overview

The ZiLOG TCP/IP Software Suite (ZTP) integrates a rich set of networking services with an efficient real-time operating system (RTOS). The operating system is a compact preemptive multitasking, multithreaded kernel with inter-process communications (IPC) support and soft real-time attributes. Table 1 lists the standard

network protocols implemented as part of the embedded TCP/IP protocol stack in ZTP.

**Table 1. Standard Network Protocols in ZTP**

| | | | | | |
|---|---|---|---|---|---|
| HTTP | TFTP | SMTP | Telnet | IP | PPP |
| DHCP | DNS | TIMEP | SNMP | TCP | UDP |
| ICMP | IGMP | ARP | RARP | | |

Many TCP/IP application protocols are designed using the client-server model. The final stack size is link-time configurable and determined by the protocols included in the build.

## *Discussion*

The Thermostat Demo illustrates how to develop an Internet-based application to control or monitor processes by combining the advantage offered by Java's robustness with a real-time embedded system that talks to low level drivers. Implementing Java compensates for the drawbacks of an embedded system, such as, lack of a file system, restricted memory, and lack of multithreading capability, to name a few.

The architecture and design of the Thermostat Demo incorporates a GUI using Java applets to connect to the embedded environment. The user interface is a Java applet. The real-time and device-specific processes are written in C. The Java GUI obtains the data from the low-level embedded software code (C) and presents the data in a client browser. The flow of data from the embedded environment to the Java-based GUI is depicted in Figure 1.

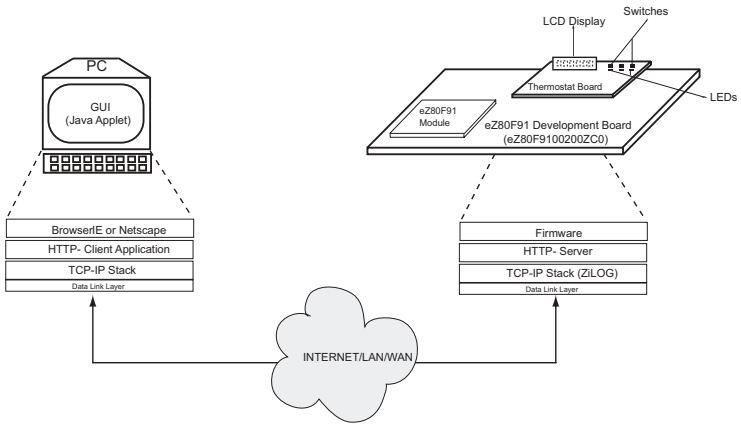


**Figure 1. Overview of Thermostat Demo using eZ80Acclaim!™**

The Thermostat Demo described in this Application Note 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.

An embedded system that uses a Java applet for its graphical user interface requires system software that supports TCP/IP with an HTTP web server running over it that serves HTML pages referring to the Java applet, and the code for the Java applet. The embedded system does not require a Java Virtual Machine. The user can 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, 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 executes the GUI applet actually runs on the web browser, and not on the embedded system.

## Advantages of Using Java

The advantages of using a Java applet are briefly described below.

- The applet is stored as a file in the web server that does not require additional memory from an embedded device to operate; the restricted memory on the embedded system is neatly sidestepped.
- An embedded system vendor can be assured that customers have access to a web browser no matter what computer platform they choose to access the embedded system from; Java applets are platform independent and can provide cross-platform GUIs.
- The Java GUI technique works well on a slow network connection, such as a serial line; at a time many clients are served efficiently.
- A Java-based GUI is a much better solution than customized 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. Only one version of the Java-based GUI software is required, and this version is stored in ROM/Flash in the embedded system.

## An Embedded HTTP Webserver

HTTP web servers use a standard synchronous request or response design running over the TCP/IP, identical to classical client or 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 is 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.

Most web server software do not work well in embedded systems. The requirements for a web server designed to run on a workstation differs from that of designed to run in an embedded system. Features such as the presence of log files and larger memory footprint in non-embedded web servers are a definite hindrance when it comes to implementing an embedded web server. Embedded web servers lay emphasis on reducing memory footprint while increasing efficiency, reliability and providing mechanisms to generate dynamic data.

Embedded systems do not typically serve a multitude of static web pages. Internet appliances and embedded systems, in general, require web servers 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 web servers offer minimal memory requirements and are efficient. The requirements for an embedded web server include:

- Memory usage
- Dynamic page refreshing/update
- Web page storage in Flash or ROM

These concepts are elaborated below.

### Memory Usage

One of the most important requirements for an embedded web server is small memory footprint. The web server use very little memory (code, stack, and heap), and it must not fragment memory. Many embedded devices employ simple memory allocations that cannot combine fragmented memory effectively. Because web servers must often respond to requests to serve pages, 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 web servers should use only statically allocated or pre-allocated memory blocks.

### Dynamic Page Refreshing/Update

An embedded device features only a small number of pages in memory, and often refreshes the page contents on the fly. The web pages display ever-changing information about device status, values read by sensors, and any other data available to the device.

### Web Page Storage in Flash or ROM

Many embedded systems do not feature disk drives, yet they must be accessed and controlled via the web. In such cases, a method of storing web pages in ROM is required. Embedded web servers should be able to access HTML, Java applets, image files, and any other web contents stored in Flash memory or ROM.

## Reading and Writing to the Embedded Webserver

To communicate over a network using Java programs, the Socket or URL classes provided in the `java.net` package are used. The underlying TCP and UDP layers are not a concern. Socket is one end-point of a two-way communication link between two programs running on the network. URL is a Uniform Resource Locator that is a pointer to a resource on the World Wide Web.

The `java.net` package provides two socket-related classes—Class Socket and Class ServerSocket—that implement the client side and the server side of a network connection, respectively. Class Socket implements the client side of a two-way connection between the Java program and another program on the network, while Class ServerSocket provides a system-independent implementation of the server side of a client/server socket connection. The Thermostat Demo implements only the Socket class because the server program is implemented in C, and not in Java.

Java programs that interact over the Internet can also use URLs to find Internet resources. Java programs use Class URL in the `java.net` package to represent a URL address. Java applets 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 URL Connection to the server (which normally connects via the Port 80 socket) to manage protocol-specific communication.

Sockets can be used regardless of whether the server supports a protocol that recognizes a URL or not, but while using a socket to contact an HTTP server, some of the details managed by the URL must be entered manually by the user. When a user wants to avoid using established protocols, communication according to user specifications can be effectively managed via sockets.

For more information regarding sockets and URLs, please visit:

http://java.sun.com/j2se

# *Thermostat Implementation Using eZ80Acclaim!™*

To understand the Thermostat implementation, it is necessary to first look at how the temperature is controlled by the eZ80Acclaim!™ MCU before getting to the details of the hardware and the software implementations.

### Temperature Control

This section discusses how the eZ80Acclaim!™ processor controls the temperature. The Java applet features buttons that allow the user to send commands to the processor to control the temperature around the temperature sensor, for setting the Thermostat control parameters, such as upper and lower temperature set points. A screenshot of the GUI implemented using a Java applet is shown in Figure 3 on page 9.

The upper and lower set point values are passed to the embedded HTTP server using the Java socket connections that invoke the CGI script. The processor obtains these values via the firmware interface that use the CGI script. The processor continuously (every two seconds) reads the temperature of the sensor and switches on/off the bulb or the fan to maintain the temperature within these set limits. The new temperature values are sent via the CGI-Firmware interface to the HTTP webserver, where the temperature values in the Java applet are updated and finally displayed on the screen. The processor also updates the temperature values on the LCD display unit every two seconds.

## *Hardware Architecture*

Figure 2 is a block diagram of the hardware architecture featuring the eZ80® Development Platform and the Thermostat Board.

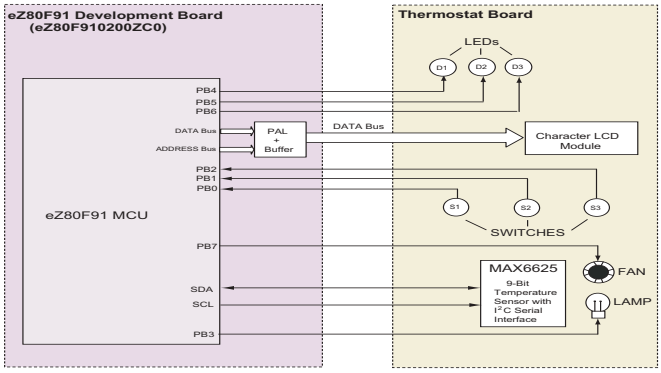


**Figure 2. Hardware Block Diagram for Thermostat Demo**

The pins PB4, PB5, and PB6 on the eZ80F91 MCU are connected to the LEDs on the Thermostat Board. The LCD Module is used as a memory map device, with `0x800002` as the memory address. The PAL (Programmable Array Logic) unit is used for address decoding. Refer to Figure 2 for details of the LCD connection.

The LCD module displays the changing temperature of the temperature sensor dynamically.

The pins PB0, PB1 and PB2 are connected to three press button switches, S1, S2, and S3 respectively. These switches can be used to force heating or cooling or prevent heating or cooling of the temperature sensor. Pin PB7 connects to the Fan that cools the temperature sensor when turned on, while pin PB3 connects to a Lamp that heats up the temperature sensor when turned on.

MAX6625 is a 9-bit Temperature Sensor with an $I^2C$ Serial interface that is connected to the SDA and SCL lines on the eZ80F91 MCU.

# *Software Implementation*

The software implementation for the Java-based Thermostat Demo is divided into the following sections:

Development of Java Applets: This section, on page 7, explains the implementation of the GUI in the form of several Java applets to create a socket connection, display temperature reading, allow setting of the upper and lower temperature values, indicate the status of heating /cooling as controlled by the hardware switches, and send inputs to control the LEDs.

Development of the Embedded Firmware: This section, on page 8, explains the two-sided Firmware interface. One side interfaces with the CGI script inputs from the HTTP web server, while the other side interfaces with the actual hardware devices like the temperature sensor and the LCD panel.

Adding and Integrating Thermostat Demo Files to ZTP: The software implementation for the Java-based Thermostat Demo involves the use of the ZiLOG TCP/IP stack (ZTP). The eZ80F91 Development Kit contains the ZiLOG TCP/IP stack software (ZTP) that supports socket connections. This section, on page 13, contains the details for adding and integrating the Thermostat Demo application with the ZTP stack. For more details about the ZiLOG TCP/IP stack, contact the ZiLOG help desk.

## Development of Java Applets

Java applets are developed using a GUI builder (any text-based editor like Notepad can be used). Sun's Java Development Kit (JDK) is used for compiling the applet source files.

The Java applet, `TstatHttpClient.class` creates the socket connection to communicate to the host computer from where the applets are loaded. When any applet requires information from the main application, it sends a request message via this socket. The main application responds by sending a response message through the same socket.

The rest of the Java applets provide control and monitoring capabilities to the Thermostat Board on the eZ80® Development Platform. The Java applets provide the following functionality:

- The facility to manipulate the upper and lower set points for the temperature sensor.
- The facility to display the updated temperature values on a client web browser in graphical and numerical formats.
- The facility to display the status of the hardware switches, which are used to force heating or cooling or turn off the heating and cooling, on the temperature sensor.

## Development of the Embedded Firmware

The embedded firmware interfaces at two levels: at one level it interfaces between the HTTP web server on the ZiLOG TCP/IP stack (ZTP) and the Java applets, and at another level the embedded firmware interfaces with the actual hardware devices on the Thermostat Board and the LCD panel.

The following sub-sections explain the details of the embedded firmware interfaces.

### Firmware Interface to Java Applets

The firmware interface establishes communication links between the Java GUI and the firmware. This firmware interface is written in C and developed using the ZiLOG Development Studio (ZDSII).

Figure 3 is a screen shot of the web page using the Thermostat applet.


**Figure 3. Control Panel for Temperature Sensor**

In Figure 3, the `Thermostat.class` applet receives the temperature data and displays it via the applet, on the web browser. The applet invokes the webserver's CGI scripts contained in the `tstat_control_cgi.c` to write to and read from the HTTP webserver.

Figure 4, is a screen shot of the web page using the Button applet.

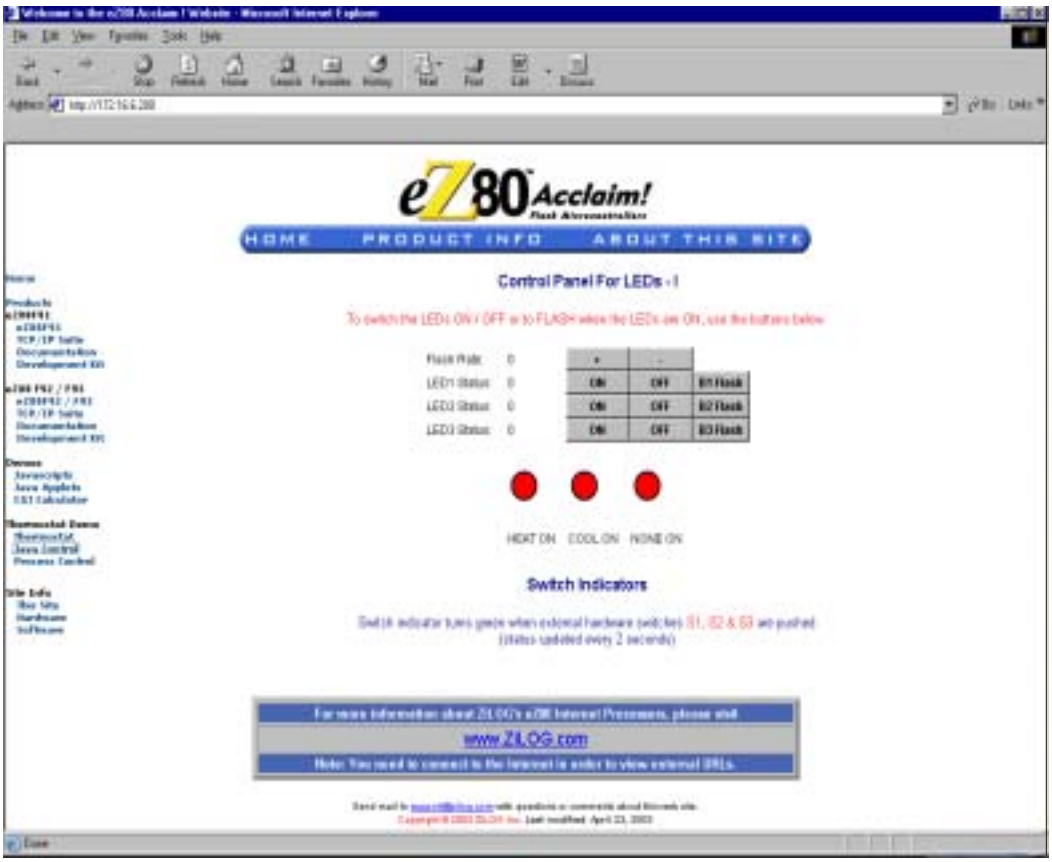


**Figure 4. Control Panel for LEDs - I**

In Figure 4, the `Button.class` applet demonstrates how the LEDs are controlled remotely and how the hardware-controlled Thermostat heating and cooling is displayed remotely. The Button applet uses the CGI script file, `java_control_cgi.c` to control the LEDs and check the hardware switch inputs.

Figure 5, is a screen shot of the web page generated by an HTML form.


**Figure 5. Control Panel for LEDs - II**

In Figure 5, The LEDs are controlled by inputs into this HTML form. The HTML page, **Control panel for LEDs-II** demonstrates ON/OFF and Flash rate control using HTML forms. The HTML form handler uses the CGI script file `switches_cgi.c`.

Embedded webserver connections are established using either the Socket Class or the URL Class available in the `java.net` package. The URL code segment that can be used instead of a direct socket connection to a server is explained here. To write to a server, a Port 80 socket connection to the server is created. The code to obtain this socket connection is called via the `getupdate()` method and the code to write to the socket is called via a `sendRequest()` method.

To read dynamic data, a URL to the CGI dynamic page is built on the server from where the applets are delivered. After opening the URL connection, the data is received in this connection. When all of the data is sent through an output stream (by calling the `getupdate()` method), an input stream (in Stream) for the URL connection reads the server's response.

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. The CGI script file, `input_cgi.c`, passes the maximum and minimum set point values from the Java applet to the `main.c` program. The functions in the `main.c` file, calls the appropriate functions to read the temperature values from the temperature sensor on the Thermostat Board and maintains the temperature between the upper and lower set points, (by turning on the bulb or the fan) by dynamically upgrading the limits set by the user using the buttons on the GUI.

The constantly changing temperature values from the temperature sensor are read by the Java applet by invoking the CGI script, and displayed in a graphical and numerical format on the web browser. Reading and writing to the server occur every two seconds. A thread running in the applet controls this process.

### Firmware Interface to Hardware

The firmware interface to the hardware performs the following tasks:

- Initializes the eZ80Acclaim!™ I/O ports to configure them for reading and writing to the devices on the Thermostat Board and the LCD panel.
- The firmware interface calls the function `http_init()` with appropriate parameters to build the web server. The web server creates several threads so that multiple web servers run on multiple ports. 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 are defined. The `index.html` and `tstat_control.html` are created as dynamic web pages and the `Thermostat.class` is created as a static web page.
- The firmware interface initializes the $I^2C$-temperature sensor.
- Reads the temperature from the MAX6625 Temperature Sensor using the `readtemp()` function.
- The Firmware interface is used to exchange data between the temperature sensor and a Liquid Crystal Display (LCD) panel, to display temperature variation dynamically. The LCD panel is plugged to the port A of the eZ80F91 MCU. The LCD program initially displays the IP address of the server. When the user requests a URL via the Browser, the program displays the set upper/lower temperature values, and the current temperature on the LCD panel. The temperature reading is updated every two seconds.

**Note:** The Thermostat Demo code files are located in the `AN0168-SC01.zip` file that is available on the ZiLOG website.

## Adding and Integrating Thermostat Demo Files to ZTP

The Thermostat Demo described in this Application Note requires the eZ80® Development Platform that contains the eZ80F91 MCU with the ZiLOG TCP/IP stack (ZTP), and the Thermostat Board. For the Thermostat Demo execution, the files specific to the demo must be added and integrated to the ZTP stack before it is downloaded onto the eZ80® Development Platform. This section contains the details of adding the Thermostat Demo's files to the ZTP stack.

The Thermostat Demo files that must be added to the ZTP project files are in the `AN0168-SC01.zip` file available on the ZiLOG website. The Demo files are of the following types:

- Assembly (*.asm) files
- C (*.c) files
- Header (*.h) files
- HTML (*.htm) files
- Java (*.class) files

The ZTP stack is available on the ZiLOG website and can be downloaded to a PC with a user registration key. ZTP can be installed in any location as specified by the user; its default location is `C:\Program Files\ZiLOG`.

Perform the following steps to add and integrate the Demo files to the ZTP stack:

1. Download ZTP, browse to the location where ZTP is downloaded, and open the `\website.Acclaim` folder.
2. Download the `AN0168-SC01.zip` file and extract its contents to a folder on your PC (this folder is referred to as `\Thermostat Demo` folder in the rest of the Application Note). Notice the two extracted folders within the `\Thermostat Demo` folder:

   `\TD_Demo`

   `\TD_Website.Acclaim`

3. Select and copy all the `*.htm/*.html` and `*.class` files in the `\Thermostat Demo\TD_Website.Acclaim` folder and paste them into the `..\ZTP\website.Acclaim` folder.
4. Select and copy all the `*.c, *.h,` and `*.asm` files located in the `\Thermostat Demo\TD_Demo` folder and paste them into the `..\ZTP\Demo` directory.
5. Launch ZDSII and open the project file, `website.pro` located in the path: `..\ZTP\website.Acclaim`

6. Now add all the `*.htm` and `*.class` files located in the `..\website.Acclaim` folder to the project, using the sequence of steps: **Project** → **Add Files**.

   The `*.htm` files to be added are listed below:

   ```
   control_page.htm
   jcontrol_page.htm
   ThermostatDemo.html
   thermostatf.htm
   tstat_control_page.htm
   ```

   The `*.class` files to be added are listed below:

   ```
   ButtonApplet.class
   CustomParser.class
   LEDBulb.class
   messagerA.class
   MiniHttpClient.class
   ParamParser.class
   Thermometer.class
   Thermostat.class
   TstatHttpClient.class
   ```

7. Open the `website.c` file from within ZDSII, and enter the following prototype declarations into it:

   ```
   // Thermostat pages
   extern struct staticpage control_page_htm;
   extern struct staticpage jcontrol_page_htm;
   extern struct staticpage thermostatf_htm;
   extern struct staticpage thermostat_htm;
   extern struct staticpage ThermostatDemo_html;
   extern struct staticpage tstat_control_page_htm;

   extern int input_cgi(struct http_request *request);
   extern int java_control_cgi(struct http_request *request);
   extern int switches_cgi(struct http_request *request);
   ```

```
extern int Thermostat_cgi(struct http_request *request);


// Java Applets
extern struct staticpage Thermometer_class;
extern struct staticpage Thermostat_class;
extern struct staticpage LEDBulb_class;
extern struct staticpage ButtonApplet_class;
extern struct staticpage CustomParser_class;
extern struct staticpage messagerA_class;
extern struct staticpage MiniHttpClient_class;
extern struct staticpage ParamParser_class;
extern struct staticpage TstatHttpClient_class;
```

8. The `website.c` file contains the an array, `Webpage website[],` with information on the HTML pages. Replace the last line of the array, `{0, NULL, NULL, NULL }`, with the following lines:

```
{HTTP_PAGE_STATIC, "/Thermometer.class", "application/octect-
stream", &Thermometer_class },
{HTTP_PAGE_STATIC, "/Thermostat.class", "application/octect-
stream", &Thermostat_class },
{HTTP_PAGE_STATIC, "/LEDBulb.class", "application/octect-
stream", &LEDBulb_class },
{HTTP_PAGE_STATIC, "/ButtonApplet.class", "application/
octect-stream", &ButtonApplet_class },
{HTTP_PAGE_STATIC, "/CustomParser.class", "application/
octect-stream", &CustomParser_class },
{HTTP_PAGE_STATIC, "/messagerA.class", "application/octect-
stream", &messagerA_class },
{HTTP_PAGE_STATIC, "/MiniHttpClient.class", "application/
octect-stream", &MiniHttpClient_class },
{HTTP_PAGE_STATIC, "/ParamParser.class", "application/octect-
stream", &ParamParser_class },
{HTTP_PAGE_STATIC, "/TstatHttpClient.class", "application/
octect-stream", &TstatHttpClient_class },
```

```
{HTTP_PAGE_STATIC, "/control_page.htm", "text/html",
&control_page_htm },

{HTTP_PAGE_DYNAMIC, "/cgi-bin/switches", "text/html", (struct
staticpage*)&switches_cgi },

{HTTP_PAGE_DYNAMIC, "/cgi-bin/java_control","text/
html",(struct staticpage*)&java_control_cgi },

{HTTP_PAGE_DYNAMIC, "/Thermostat.html", "text/html", (struct
staticpage*)&Thermostat_cgi },

{HTTP_PAGE_STATIC, "/jcontrol_page.htm", "text/html",
&jcontrol_page_htm },

{HTTP_PAGE_STATIC, "/thermostatf.htm", "text/html",
&thermostatf_htm },

{HTTP_PAGE_STATIC, "/tstat_control_page.htm", "text/html",
&tstat_control_page_htm },

{HTTP_PAGE_DYNAMIC, "/Data.html", "text/html", (struct
staticpage*)&input_cgi },

{0, NULL, NULL, NULL }
```

9. From within ZDSII, open the `left.htm` file under `\Web Files`. Search for **CGI Calculator** and locate the following line:

```
   <a href="cgif.htm" target="_top">CGI
Calculator</a></font><p><font face="Verdana" size="1"><b>Site
Info<br>
```

10. Replace the `</font>` with the following peice of HTML code, to create a link from the default eZ80Acclaim!™ web page to the Thermostat Demo web page.

```
Thermostat Demo<br>

  <a href="tstat_control_page.htm">Temp.Sensor</
a><br>

  <a href="jcontrol_page.htm" target="main">LED
Ctrl Panel-I</a><br>

   <a href="control_page.htm" target="main">LED
Ctrl Panel-II</a></font>
```

11. Build the `website.pro` project to obtain the new library file, `Acclaim_website.lib`. Copy this library file to the path: `..\ZTP\libs`.

**Note:** Please note that the `..ZTP\libs` folder already contains an `Acclaim_website.lib` file and it must be replaced with the newly generated file. Click **Yes** to replace the file.

12. Close the `website.pro` project.
13. In ZDSII, open the `AcclaimDemo.pro` file available in the path: `..\ZTP\Demo.`
14. Add all the `*.c`, `*.h,` and `*.asm` files located in the `..\Thermostat Demo\TD_Demo` folder to the project, using the sequence of steps: **Project** → **Add Files**.

    The `*.c, *.h` and `*.asm` files to be added are listed below:

    ```
    timer_isr.asm
    LCD_API.h
    initialization.c
    input_cgi.c
    java_control_cgi.c
    LCD_API_port.c
    switches_cgi.c
    temp_read.c
    Thermostat_cgi.c
    tstat_control_cgi.c
    ```

15. Open the `main.c` file of the AcclaimDemo project and add the following include file:

    ```
    #include <LCD_API.h>
    ```

16. In the `main.c` file, observe the following `BootInfo` structure definition:

    ```
    struct BootInfo Bootrecord = {
            "192.168.1.1",/* Default IP address */
            "192.168.1.4",/* Default Gateway */
            "192.168.1.5",/* Default Timer Server */
            "192.168.1.6",/* Default File Server */
            "",
            "192.168.1.7",/* Default Name Server */
            "",
    ```

```
0xffffff00UL/* Default Subnet Mask */
};
```

The `Bootrecord` variable contains the network parameters and settings (in the four-octet dotted decimal format) that are specific to the local area network at ZiLOG as default.

*Modify the above structure definition with appropriate IP addresses within your local area network.*

17. In the `main.c` file, add the following function prototypes and global variables:

```
//prototype functions
extern void reg_init_function(void);
extern void tstat_function();


//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;
```

18. At the end of the `main.c` file, add the following lines of code:

```
reg_init_function();
LCD_init ();
LCD_prints("Zilog Acclaim!");// Print a string
LCD_setposition(1,0);
LCD_prints("IP");
LCD_setposition(1,3);
LCD_prints(Bootrecord.myip);// Print a string
```

```
while(1)
{
     tstat_function();
     sleep(2);
}
```

19. Save the files and close the project.

# *Demonstration*

This section contains the requirements and instructions to set up the Thermostat Demo and run it.

## Requirements

The requirements are classified under hardware and software.

### Hardware

- eZ80F91 Development Kit (eZ80F910200ZCO)
- Thermostat Board (eZ801900100ZAC) along with a 9-volt power supply
- PC with an Internet Browser

### Software

- ZiLOG Developer Studio II—IDE for eZ80Acclaim! (ZDSII)
- ZiLOG's TCP/IP stack (ZTP)
- Project file for the Thermostat Demo for eZ80Acclaim!™(`AcclaimDemo.pro`) located within the `AN0168-SC01.zip` file.

## Setup

The basic setup to assemble the Thermostat Demo is illustrated in Figure 6. This setup illustrates the connections between the PC, LAN/WAN/Internet and the eZ80F91 Development Kit.

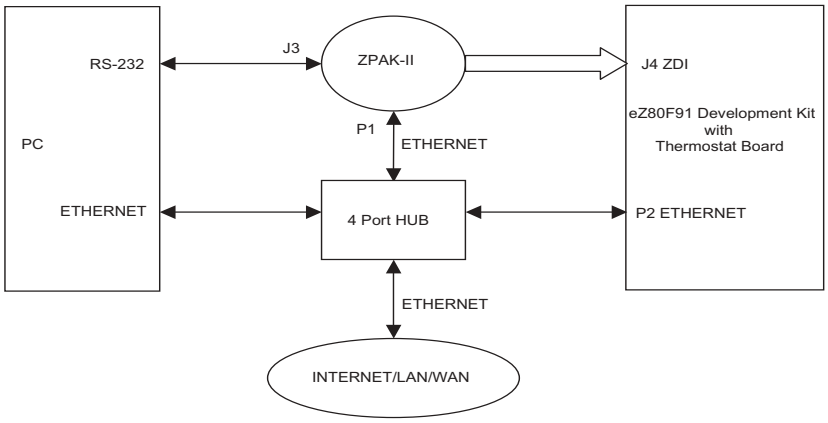


**Figure 6. Setup for Thermostat Demo**

## Settings

### HyperTerminal Settings

- Set HyperTerminal to 57.6 Kbps Baud and 8-N-2, with no flow control

### Jumper Settings

#### For eZ80® Development Platform

- J11, J7, J2 are ON
- J3, J20, J21, J22 are OFF
- For J14, connect 2 and 3
- For J19, MEM_CEN1 is ON, and CS_EX_IN, MEM_CEN2, and MEM_CEN3 are OFF

#### For the eZ80F91 Module on eZ80® Development Platform

- JP3 is ON

## Procedure

The procedure to build and run the Thermostat Demo is described in this section.

1. Ensure that the required Thermostat Demo files are added and integrated to ZTP before proceeding. See section Adding and Integrating Thermostat Demo Files to ZTP on page 13, for details.
2. Make the connections as per Figure 6. Follow the jumper settings provided in the section on Jumper Settings above.
3. Connect the 9-volt power supply to the eZ80F91 Development Kit and the Thermostat Board separately.
4. Connect the 5-volt power supply to ZPAKII and the 7.5-volt power supply to the Ethernet HUB.
5. Launch the HyperTerminal and follow the settings provided in the HyperTerminal Settings section on page 20.
6. From within the HyperTerminal, press *z* repeatedly, and then press the reset button on ZPAKII to view the menu to set the ZPAKII IP address.
7. Enter H to display help menu, and follow the menu instructions to obtain the IP address for ZPAKII in order to download the Demo file. This ZPAKII IP address must be entered in the ZDSII.
8. Launch ZDSII–for eZ80Acclaim!™ and open the Thermostat Demo project file (`AcclaimDemo.pro`) located in the path: `..\ZTP\Demo`.
9. Open the `main.c` file. Ensure that the `BootInfo` structure contains information that is relevant to your network configuration. Use the IP address in the structure to browse the Internet to view the Thermostat Demo web pages.
10. Build the project and download the resulting file to the eZ80F91 Module on the eZ80® Development Platform using ZDSII.
11. Run the Thermostat Demo. Refer to Running the Thermostat Demo section below.

### Running the Thermostat Demo

1. Launch the Internet Browser on the PC. Enter the IP address specified in `main.c`. The `Index.html` page is displayed.
2. Click on the **Temp. Sensor** link in the left pane. The **Control Panel for Temperature Sensor** page is displayed. Observe the temperature displayed in graphical and numerical form. Observe the upper and lower limits set for the temperature.

3. Click on the **DECREASE UPPER/DECREASE LOWER/INCREASE UPPER/INCREASE LOWER,** buttons to change the upper/lower limits of the temperature. Observe the temperature reading.
4. On the Demo Thermostat Board, hold down switch **S1.** The bulb glows, and heats the temperature sensor. Observe that the temperature reading rises above the set upper limit as long as the S1 switch is held down.
5. Hold down switch **S2**. The fan rotates, and cools the temperature sensor. Notice that the temperature reading falls below the set lower limit as long as the S2 switch is held down.
6. Hold down switch **S3.** Neither the light bulb nor the fan work. Notice that the temperature reading reaches the ambient temperature irrespective of the set upper and lower limits and remains steady as long as the S3 switch is held down.
7. Click on the **LED Ctrl Panel - I** link in the left pane of the Browser window. The **Control Panel for LEDs - I** page is displayed.
8. Switch on the LEDs using the **ON** button. Click on the Flash Rate buttons to specify the rate for blinking, and click the **Flash** button to activate blinking. On the Board, notice that all the LEDs are on and are blinking at the rate specified.
9. Switch off the LEDs using the **OFF** button. Notice that the LEDs on the Board are switched off.
10. On the Demo Thermostat Board, hold down switch **S1.** The bulb glows. The Switch Indicator for **HEAT ON** turns green, indicating that the S1 switch on the Thermostat Board is ON. Repeat this for the remaining switches and observe the effects.
11. Click on the **LED Ctrl Panel - II** link in the left pane of the Browser window. The **Control Panel for LEDs - II** page is displayed.
12. Enter **1** in all the fields, to turn the LEDs ON.
13. In the **Flash** text box, enter **1** to make the LEDs blink. Turning the blinking on is possible only for those LEDs that are ON.
14. In the Flash rate text box, enter a number between **1-100** to set the blinking at a specified rate. Entering 1 sets the blinking to the fastest rate while entering 100 sets it to the slowest rate.
15. Click the **Send LED Settings and Get Status Info** button to effect the changes and obtain an LED status report.

## Summary

This Application Note highlights the eZ80Acclaim!™ MCU's capability to perform as efficient embedded web servers by demonstrating an Internet-enabled process control and monitoring application in the form of a Thermostat Demo that embeds Java elements on the eZ80Acclaim!™ microcontroller.

The 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 a control system in real time and display values dynamically.

The advantage of using eZ80Acclaim!™ with Java is that platform independence is achieved with a Java GUI as a client application, and real-time processes are controlled and monitored by the eZ80Acclaim!™ MCU interacting with the hardware devices.

# Appendix A—Reference

Further details about the eZ80F91 MCU, ZDSII, ZPAKII and the Thermostat Board can be found in the references listed in Table 2.

**Table 2. List of References**

| Topic | Document Name |
|---|---|
| eZ80Acclaim!™ MCU | eZ80F91 MCU Product Specification (PS0192) |
| Thermostat Board | Thermostat Application Module Product User Guide (PUG0014) |
| ZDSII for eZ80Acclaim!™ MCUs | ZiLOG Developer Studio II - eZ80Acclaim!™ User Manual (UM0144) |
| ZPAK II | ZPAK II Debug Interface Tool Product User Guide (PUG0015) |
| ZTP | ZiLOG TCP/IP Software Suite Programmer's Guide Reference Manual (RM0008) |

# Appendix B—Flowcharts

This appendix contains the flowcharts for the Thermostat Demo implementation on the eZ80F91 MCU.

Figure 7 illustrates the flowchart for the main routine.

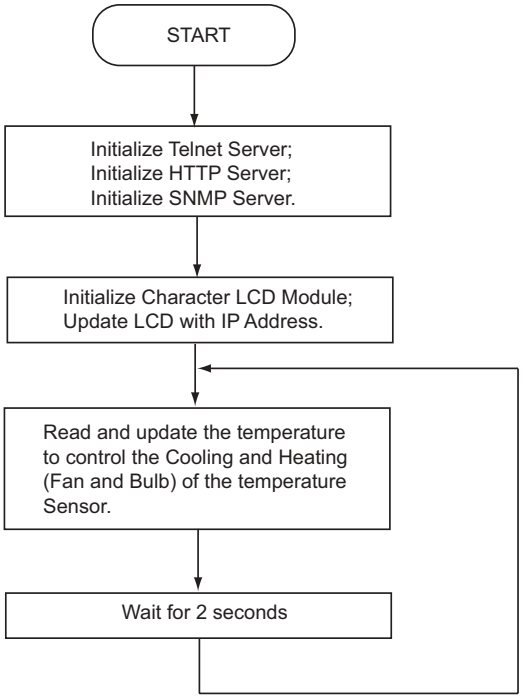


**Figure 7. Flowchart for the Main Routine**

Figure 8 illustrates the flowchart for the Thermostat Applet and LCD update.

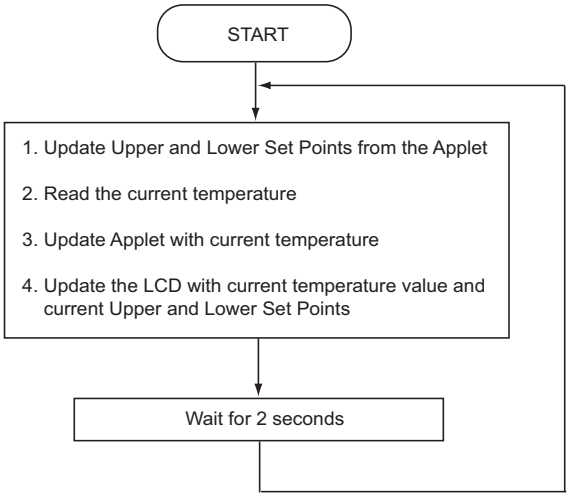


**Figure 8. Flowchart for the Thermostat Applet and LCD Update**