



Application Note

***A Web Page – EEPROM
Interface Using the
eZ80F91 Web Server***

AN020901–1204



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

ZILOG is a registered trademark of ZILOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

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

©2004 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.



Table of Contents

List of Figures	iv
List of Tables	iv
Abstract	1
ZiLOG Product Overview	1
eZ80Acclaim!™ MCU Family Overview	1
ZiLOG TCP/IP Software Suite Overview	2
Discussion	2
Developing the Web Page-EEPROM Interface Application	4
Web Page-EEPROM Interface HTML Files	4
Application-Specific CGI Functions	7
I ² C and EEPROM APIs	7
Adding and Integrating Application-Specific Files to ZTP	8
Demonstration	16
Requirements	16
Setup	16
Settings	16
Procedure	17
Observations	18
Summary	18
Appendix A—References	19
Appendix B—Source Code	20
C Files	20
Header Files	44



List of Figures

Figure 1. General Data Communication Path via I ² C	3
Figure 2. Machine Control Demo Page	5
Figure 3. Screen Shot Illustrating the ZTP Shell Commands	13
Figure 4. Command Prompt View of File Transfer Using FTP	14
Figure 5. Screen Shot Illustrating the Files Stored in Flash after File Transfer Using FTP	15

List of Tables

Table 1. Standard Network Protocols in ZTP	2
Table 2. List of References	19



Abstract

This Application Note describes how to develop a web page interface to transfer data from/to a peripheral hardware device (an I²C EEPROM in this application) using the web server capabilities offered by the eZ80F91 MCU in conjunction with ZiLOG's ZTP software. The web page interface provides a communication interface between a web page and an EEPROM device via the I²C peripheral of the eZ80F91 MCU.

Using the HTTP-specific APIs of the ZTP stack and the on-chip I²C peripheral, the data received from the web page is stored in the I²C EEPROM and, upon a request received from a web page, the stored data is retrieved and displayed on the web page.

The source code file associated with this Application Note, AN0209-SC01.zip, is available on the [ZiLOG website](#).

- ▶ **Note:** The code files in this document are intended for use with the ZiLOG TCP/IP Software Suite (ZTP) version 1.4.1. If you do not intend to develop your application with ZTP v1.4.1, ZiLOG provides an Application Note about this topic that is intended for use with earlier versions of ZTP. Please see A Web-Page EEPROM Interface Using the eZ80F91 Web Server Application Note (AN0177), available on [zilog.com](#).

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-feature 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 (Client and Server)	IP	PPP
DHCP	DNS	TIMEP	SNMP	TCP	UDP
ICMP	IGMP	ARP	RARP	FTP (Client and Server)	

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

Figure 1 depicts the general data communication path using the eZ80F91 Development Kit with the EEPROM device, the ZPAKII, and an Ethernet Hub. With this hardware, the eZ80F91 MCU is ready to be used as an efficient web server when ZiLOG's ZTP software is downloaded on it. ZTP provides the software to drive the hardware used for TCP/IP connections. The hardware comprises of a SERIAL1 (UART1) port for PPP connections and the Ethernet Media Access Controller (EMAC) for Ethernet connections.

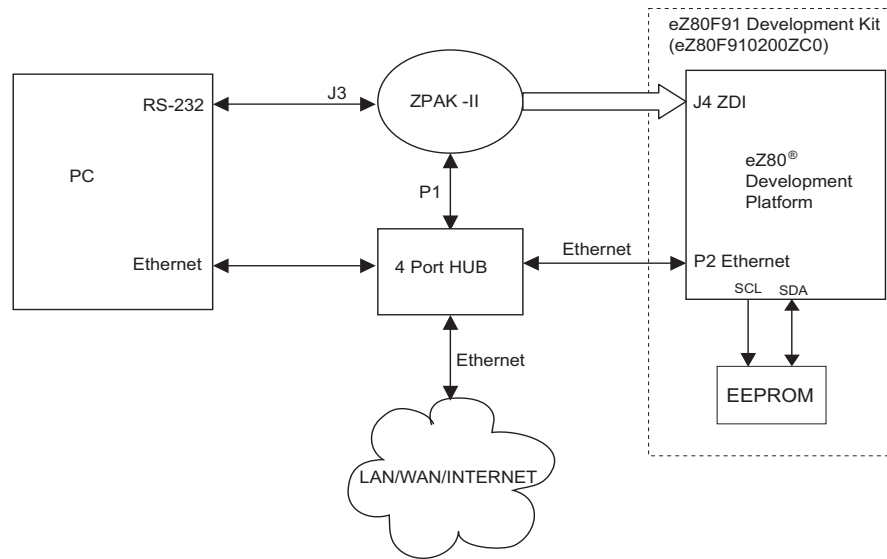


Figure 1. General Data Communication Path via I²C

The ZILOG TCP/IP Software Suite (ZTP) contains a set of libraries that implement an embedded TCP/IP stack and includes a pre-emptive, multi-tasking kernel.

The ZTP Applications Programming Interface (API) allows programmers using any member of the eZ80[®] family of microprocessors/controllers to rapidly develop Internet-ready applications with minimal effort. Because the API is common to all members of the eZ80[®] family, applications targeting one processor are easily ported to any other eZ80[®] devices.

ZTP HTTP CGI Functions

Embedded systems typically do not contain a file system. Lack of a file system implies that embedded systems cannot save CGI scripts as separate *.cgi files.

Instead of saving CGI scripts as separate *.cgi files, ZTP uses C function calls, collectively called CGI functions. When a CGI function is called, it generates an HTML page that is sent to the browser. It is in these function calls that a programmer writes code to read the information sent by a form via a web browser. This information is then processed as required by the application.

ZTP provides the following CGI functions to the user.

- `int http_output_reply(http_request *request, int reply)`
- `char *http_find_argument (http_request *request, char *arg)`



- `int _http_write (http_request *request, char *buff, int count)`

In each ZTP CGI function, the pointer to the `request` structure is used to keep the requests from different clients separate.

The function `http_output_reply()` is used to return an acknowledgement to the browser that made the request.

The `http_find_argument()` function is used to extract parameters from the received data in the parsed browser request.

The macro `_http_write()` is used to return data to the browser that sent the request that invoked the CGI function.

For a detailed information about the protocols used in ZTP, refer to the ZiLOG TCP/IP Software Suite v1.4 Programmer's Guide (RM0041), available with the ZTP software.

Developing the Web Page-EEPROM Interface Application

This section discusses the I²C APIs, the EEPROM APIs, the HTML pages and the CGI functions, and includes a section about how to interface the application-specific files to the standard ZTP.

The I²C peripheral on the eZ80F91 MCU is used to communicate with the EEPROM. The data received from the web page is stored in the EEPROM and upon request the same information or data is retrieved from the EEPROM and displayed on the web page.

Web Page-EEPROM Interface HTML Files

Figure 2 is a screen shot of the Machine Control Demo web page (`mcd.htm`) that contains two columns: Write Data column and the Read Data column. The Write Data column allows users to submit data to be programmed into the EEPROM memory, while the Read Data column allows users to read the data obtained from the EEPROM memory.



Figure 2. Machine Control Demo Page

The Write Data and the Read Data columns contain three buttons each, enabling a user to write to or read from three memory blocks on the EEPROM. These memory blocks are referred to as Machine-1, Machine-2 and Machine-3.

Writing Data to the EEPROM Device

On clicking a button (Machine-1) in the Write Data column, a pop-up window opens with an HTML submission form that is used to obtain user input. This is an instance of a static web page. To achieve this functionality, a JavaScript method for a pop-up window is called within the `mc.d.htm` file. The `wm1()` function is presented below as an example:



```
function wm1()  
{  
  popup =window.open("wm1.htm", "", "height=290,width=300,  
  scrollbars=no,screenX=350,screenY=250,left=350,top=250");  
}
```

When the form with the user input is submitted, a `hidden` variable with a value equal to 1 is sent to write the data to the Machine-1 memory block.

In the `mcd.htm` file, three such methods (`wm1()`, `wm2()`, and `wm3()`) are implemented for the three memory blocks on the EEPROM.

A JavaScript function checks the input fields for the values entered. If the values are not according to specification, then an alert is sent to the Browser window to display an error message. The JavaScript also checks to see that none of the fields are left blank and throws up an error if a null value field is found.

Reading Data from the EEPROM Device

On clicking a button (Machine-1) in the Read Data column, a pop-up window opens displaying the data retrieved from the EEPROM memory block Machine-1. This is an instance of a dynamic web page. A JavaScript function calls the `i2cread_cgi()` function which is responsible for reading the data from the appropriate memory block (in this case Machine-1 memory block). The `read1()` function is presented below as an example:

```
Function read1()  
{  
  popup = window.open("/cgi-bin i2cread?hidden=1", "",  
  "height=290,width=300,scrollbars=no,screenX=350,screenY=250,  
  left=350,top=250");  
}
```

In the above script, a value identifying the clicked button is sent along with the read request. The value `hidden=1` identifies that a read request was generated for reading the values from the Machine-1 memory block on the EEPROM.

The `read2()` and `read3()` functions also call the `i2cread_cgi()` function. The values for `hidden` are 2 and 3, indicating that read requests were generated for reading the values from Machine-2 and Machine-3 memory blocks on the EEPROM.



Application-Specific CGI Functions

In the Web Page-EEPROM Interface application two CGI functions, `i2cread_cgi()` and `i2cwrite_cgi()`, are used. These functions are available in the `i2c_cgi.c` file. These CGI functions call the ZTP HTTP CGI functions to read the user input from the HTML form and to display the information fetched from the EEPROM on the web page. See the [ZTP HTTP CGI Functions](#) section on page 3.

The `i2cwrite_cgi()` function, invoked when a button on the Write Data column is clicked, is structured as follows:

```
int i2cwrite_cgi(struct http_request *request)
```

This CGI function calls the ZTP HTTP CGI function, `http_output_reply()`, to send an acknowledgement to the client browser. The `http_find_argument()` function is called to fetch the user input data from the HTML form. This user input data is then transferred over the I²C bus to the EEPROM using the I²C APIs, and the input data is written to the specified memory block in the EEPROM using the EEPROM-specific APIs. The ZTP macro `_http_write` is used to write a success message, which is displayed on the web page.

The `i2cread_cgi()` function, invoked when a button on the Read Data column is clicked, is structured as follows:

```
int i2cread_cgi(struct http_request *request)
```

The `i2cread_cgi()` function first calls the ZTP HTTP CGI function, `http_output_reply()`, to send an acknowledgement to the client browser. The `http_find_argument()` function is called next to determine from which memory block the data must be read. The EEPROM APIs are then called to read from the specified memory block on the EEPROM. Finally, the ZTP macro `_http_write` is used to write the retrieved data to a pop-up window.

I²C and EEPROM APIs

The I²C Controller on the eZ80F91 MCU controls the WRITE and READ operations on the I²C EEPROM. The I²C APIs are not re-entrant. Therefore, while one process is executing, the next process is in queue and executes only after the first completes. Semaphores are used to maintain process synchronization. These semaphores are created and handled within the application-specific CGI functions. The file, `i2c.c`, contains all the APIs specific to I²C peripheral of the eZ80F91 MCU.



The `eeeprom.c` file contains the APIs to perform the READ and WRITE operations on the I²C EEPROM. These APIs are called within the application-specific CGI functions to write to or read from the EEPROM device.

Adding and Integrating Application-Specific Files to ZTP

The Web Page-EEPROM Interface application described in this Application Note, requires the eZ80[®] Development Platform that contains an EEPROM device and the eZ80F91 MCU, with the ZiLOG TCP/IP stack (ZTP).

For executing the application, the files specific for the application must be added and integrated to the ZTP stack. Static web pages (static HTML files) must be uploaded to the external Flash memory on the eZ80F91 module. This section discusses the details of adding the application-specific files to the ZTP stack and loading the static web pages into the Flash memory.

The Web Page-EEPROM Interface files that must be loaded to the external Flash memory to integrate with ZTP are provided in the `AN0209-SC01.zip` file, available on the [ZiLOG website](http://www.zilog.com). The application-specific files are of the following types:

- C (`*.c`) files
- Header (`*.h`) files
- HTML (`*.htm`) files

The ZTP stack is available on the [ZiLOG website](http://www.zilog.com) and can be downloaded to a PC utilizing a user registration key. ZTP can be installed in any location as specified by the user; its default location is `C:\Program Files\ZiLOG`.

► **Note:** Please refer to the [Requirements](#) section on page 16, for the version of ZTP and ZDS II used in this application.

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

1. Download ZTP. Browse to the location where ZTP is downloaded.
2. Download the `AN0209-SC01.zip` file and extract all its contents to a folder on your PC (this folder is referred to as `\Webpage_Interface` in the rest of the Application Note). The two extracted folders within the `WebPage_Interface` folder are:

```
\WP_Demo  
\WP_Website.Acclaim
```

3. Copy all the `*.c` and `*.h` files located in the `\WebPage_Interface\WP_Demo` folder to the `..\ZTP\SamplePrograms\ZTPDemo` directory.



12. Add all the *.c files located in the ..\WebPage_Interface\WP_Demo folder to the ZTPDemo_F91.pro project. To do so, click **Project** and then click **Add Files**. The *.c files to be added are:

```
eeeprom.c  
i2c.c  
i2c_cgi.c
```

13. Open the main.c file of the ZTPDemo_F91.pro project and add the following include file:

```
#include<eeeprom.h>
```

14. Add the following function prototypes and global variables to the main.c file:

```
// global declarations  
extern SID SemaphoreID;  
// prototype functions  
char load_init_value_eeeprom (void);
```

15. Add the following code snippet before the open (SERIAL0, 0,0) function in the main.c file.

```
SemaphoreID = screate (1);  
if(SemaphoreID == NULLPTR)  
kprintf(" Semaphore NOT created");
```

Adding this code creates a semaphore, which is used for controlling the reentrancy issues associated with the I²C APIs.

16. At the end of the main() function, add the following function call just before the return(OK) statement.

```
load_init_value_eeeprom ();
```

17. Add the following function definition after the main() function in the main.c file.



```
struct If ifTbl[MAX_NO_IF]= {
// interface 0 -> Ethernet Configuration
{
&usrDevBlk[0],           // Control block for this
                        // device
ETH,                    // interface type
ETH_MTU,                // MTU
ETH_100,                // Speed can be ETH_10,
                        // AUTOSENSE
"172.16.6.200",         // Default IP address
    "172.16.6.1",       // Default Gateway
    0xffff0000UL        // Default Subnet Mask
}
}
```

The structures listed above contain network parameters and settings (in the four-octet dotted decimal format) specific to the local area network at ZiLOG, as default.

Modify the above structure definition with appropriate IP addresses within your local area network (For details about modifying the structure definition, refer to the ZTP 1.4 documents listed in [Appendix A](#) on page 19.

21. Click **Project Settings**, and select the **Preprocessor** category from the **C** tab. Remove the `I2C` definition from the **Preprocessor** definitions field. Ensure that the `ERASE_FLASH` variable is set to 1. Save the files, and compile and build the project. Further, select the appropriate target under the **Debugger** tab.
22. Launch the HyperTerminal application and configure the HyperTerminal application to the following settings: 57600 bps baud, 8-N-2, with no flow control.
23. Download the `ZTPDemo_F91.lod` (Flash or RAM) output file to the target platform. Execute the program. To do so, click the **GO** icon in the ZDSII-IDE. Observe the ZTP shell prompt `ZTP EXTf :/ %` in the HyperTerminal application.
24. Stop program execution. Click **Project Settings**, and select the **Preprocessor** category from the **C** tab. Ensure that the `ERASE_FLASH` variable is set to 0. Compile, build and load the project file to the target board. Execute the program and observe the shell prompt in the HyperTerminal application.
25. Create a user on the eZ80[®] target board, where the ZTP stack is executing through the shell prompt. Utilize the **addusr** command to create a user and password if required. Creating a user is a must to store html pages. Different options are displayed by typing the `?` symbol, in the shell.

Figure 3 illustrates the ZTP Shell commands.

```
Initializing network stack...
Initializing network interface(s)...
Configuring Ethernet on interface 0...Done
Initialization Done

ZTP HTTP server is ready.
ZTP FTP server is ready.
ZTP TELNET server is ready.
ZTP SNMP agent is ready.
Initializing File System, Please Wait...Done
---, 1 Jan 9 0:160:124

ZTP EXTf:/ %?
Commands are:
?          addusr      arp          bpool
cd         copy         cwd          del
deldir    deleteusr  deltree     devs
dir       echo        exit        ftp
gettime   hang       help        ifstat
igmp      kill       mail        md
mem       move       netstat     ping
port      ps         reboot      ren
rendir    sem        settime     sleep
telnet    tftp_get  tftp_put    type
vol

ZTP EXTf:/ %addusr
Usage: addusr Username password
ZTP EXTf:/ %addusr zilog zilog
ZTP EXTf:/ %_
```

Figure 3. Screen Shot Illustrating the ZTP Shell Commands

26. All html pages must be stored in the external Flash memory, on the eZ80[®] target board. Launch the command prompt in the PC and select the directory from which the html files are to be copied (In this case, <web files installed folder>\WP_Website.Acclaim\). Use FTP to transfer the html pages from the PC to the eZ80[®] target board.

The following example illustrates the transfer of html pages from the PC to the eZ80[®] target board, using FTP.

At the <web files installed folder>\WP_Website.Acclaim\ PC command prompt, type the IP address of the target platform. Enter the user name and password when prompted. Use the `put` function to load the following files:



Wm1.htm
Wm2.htm
Wm3.htm
Mcd.htm
Mcdf.htm

These files are stored in the root directory of the target file system. If you want to store these files in any other directory, then the path has to be defined accordingly. By default, the `char hotpot []` array in the `ZTPCon-fig.c` file is set to the root directory as follows:

```
char httpspath[ ] = "/" // Default set to root
```

Figure 4 shows the command prompt view of transferring files using FTP.

A screenshot of a Windows Command Prompt window titled "Command Prompt - ftp 172.16.6.250". The window shows a series of FTP commands and their corresponding responses. The user is in the directory "C:\web_pages" and connects to the IP address 172.16.6.250. The user is identified as "zillog" and logs in successfully. The user then transfers five files: "wm1.htm", "wm2.htm", "wm3.htm", "mcd.htm", and "mcdf.htm". Each file transfer is successful, with the response "File received OK" and a confirmation of bytes sent and time taken. The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
C:\web_pages>ftp 172.16.6.250
Connected to 172.16.6.250.
220 ZTP1.4 FTP version 0.1 (wed 08 15:06:07 IST 2004) is ready
User (172.16.6.250:(none)): zillog
331 Enter PASS command
Password:
230 Logged in
ftp> put wm1.htm
200 Port command okay
150 Opening data connection for STOR ./wm1.htm
226 File received OK
ftp: 4581 bytes sent in 0.00Seconds 4581000.00Kbytes/sec.
ftp> put wm2.htm
200 Port command okay
150 Opening data connection for STOR ./wm2.htm
226 File received OK
ftp: 3746 bytes sent in 0.00Seconds 3746000.00Kbytes/sec.
ftp> put wm3.htm
200 Port command okay
150 Opening data connection for STOR ./wm3.htm
226 File received OK
ftp: 3745 bytes sent in 0.00Seconds 3745000.00Kbytes/sec.
ftp> put mcd.htm
200 Port command okay
150 Opening data connection for STOR ./mcd.htm
226 File received OK
ftp: 3802 bytes sent in 0.00Seconds 3802000.00Kbytes/sec.
ftp> put mcdf.htm
200 Port command okay
150 Opening data connection for STOR ./mcdf.htm
226 File received OK
ftp: 608 bytes sent in 0.00Seconds 608000.00Kbytes/sec.
ftp>
```

Figure 4. Command Prompt View of File Transfer Using FTP



27. In the HyperTerminal application, type `dir` to ensure that the files are copied properly.

Figure 5 shows the files stored in Flash after file transfer using FTP.

```

port          ps          reboot      ren
readdir       sem         settime     sleep
telnet        tftp_get    tftp_put    type
vol
ZTP EXTf:/ %dir

*****
DATE          TIME          TYPE          SIZE(bytes)   NAME
*****
01/01/0009    00:32:60      4581          wm1.htm
01/01/0009    00:32:60      3746          wm2.htm
01/01/0009    00:32:60      3745          wm3.htm
01/01/0009    00:32:60      3802          mcd.htm
01/01/0009    00:32:60      608           mcdf.htm

          Number of File(s) 5
          Number of Dir(s) 0
*****
ZTP EXTf:/ %_
    
```

Figure 5. Screen Shot Illustrating the Files Stored in Flash after File Transfer Using FTP

28. Compile and build the `ZTPDemo_F91.pro` project.



Demonstration

This section contains the requirements and instructions to set up the Web Page-EEPROM Interface demo and run it.

Requirements

The requirements are classified under hardware and software.

Hardware

- eZ80F91 Development Kit (eZ80F910200ZCO) with the I²C EEPROM device
- PC with an Internet Browser and the HyperTerminal application

Software

- ZiLOG Developer Studio II—IDE for eZ80Acclaim!™ 4.8.0 (ZDS II 4.8.0)
- ZiLOG's TCP/IP Software Suite 1.4.1 (ZTP v1.4.1)

Setup

The basic setup to assemble the Web Page-EEPROM Interface demo is illustrated in [Figure 1](#) on page 3. This setup illustrates the connections between the PC, LAN/WAN/Internet and the eZ80F91 Development Kit.

Settings

HyperTerminal Settings

- Set the HyperTerminal application to 56700 bps Baud and 8-N-2, with no flow control

Jumper Settings

For the 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 mounted on the eZ80[®] Development Platform.

- JP3 is ON



Procedure

The procedure to build and run the Web Page-EEPROM Interface demo is described in this section.

1. Ensure that the required Web Page-EEPROM Interface demo files are added and integrated to ZTP before proceeding. See section [Adding and Integrating Application-Specific Files to ZTP](#) on page 8, for details.
2. Make the connections as per [Figure 1](#). Follow the jumper settings provided in the section on [Jumper Settings](#) above.
3. Connect the 5-volt power supply to ZPAKII and the 7.5-volt power supply to the Ethernet HUB.
4. Launch the HyperTerminal application and follow the settings provided in the [HyperTerminal Settings](#) section on page 16.
5. From within the HyperTerminal application, press z repeatedly, and then press the reset button on ZPAKII to view the menu to set the ZPAKII IP address.
6. 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.
7. Launch ZDSII - eZ80Acclaim!™ and open the Web Page-EEPROM Interface demo project file (`ZTPDemo_F91.pro`) located in the path:
`..\ZTP\SamplePrograms\ZTPDemo.`
8. Open the `ZTPConfig.c` file. Ensure that the structure (refer the [Adding and Integrating Application-Specific Files to ZTP](#) section on page 8) contains information that is relevant to your network configuration. Use the IP address in the structure to browse the internet and view the Web Page-EEPROM Interface demo.
9. Build the project and download the resulting file to the eZ80F91 module mounted on the eZ80® Development Platform, using ZDSII.
10. Run the Web Page-EEPROM Interface demo. Refer to [Running the Web Page-EEPROM Interface Demo](#) section below.

Running the Web Page-EEPROM Interface Demo

1. Launch the Internet Browser on the PC. Enter the IP address specified in `ZTPConfig.c`. The `Index.html` page is displayed.
2. Click on the **Machine control** link located at the left pane. The **Machine Control Demo** page is displayed.



3. Click on the **Machine-1** button under the **Write Data** column. The **Write data** pop-up window opens displaying a submission form. The submission form consists of the **Max Temperature, Min Temperature, Max Voltage** and **Min Voltage** fields.

► **Note:** Enter values between 0 and 99.99. Do not enter characters; only integers and floating-point values are accepted. The value field is limited to 5 integers with a dot in between to represent the floating-point value. Entries not complying with these specifications cause an error message to be displayed.

4. Set the maximum and minimum temperature and voltage values for Machine-1. Click the **Submit** button to submit the data. A status pop-up window displays that the data is successfully stored in the EEPROM.
5. Repeat steps 3 and 4 using the **Machine-2** and **Machine-3** buttons under the **Write Data** column to set the values for those machines.
6. Go back to the **Machine Control Demo** page by clicking on the **Machine control** link in the left pane.
7. Click on **Machine-1** button under the **Read Data** column. The **Read data** pop-up window displays the values set for Machine-1.
8. To read the values set for Machine-2 and Machine-3, click on the **Machine-2** and **Machine-3** buttons under the **Read Data** column.

Observations

The values set for the machines were successfully written to the EEPROM device via the web page interface and I²C. This is confirmed by comparing the data read from the EEPROM device.

Summary

This Application Note demonstrates a method of capturing user input data from a web page and storing it on an external device, and further displaying the stored data on the web page upon a user request.

This application can be further extended and used as an industrial web server, where the web server can be connected to different machines or control equipments that can be monitored and controlled through the intranet or the Internet.



Appendix A—References

Further details about the eZ80F91 MCU and ZTP can be found in the references listed in Table 2.

Table 2. List of References

Topic	Document Name
eZ80 [®] CPU	eZ80 [®] CPU User Manual (UM0077)
eZ80F91 MCU	eZ80F91 Development Kit User Manual (UM0142)
	eZ80F91 Flash MCU with Ethernet MAC Product Specification (PS0192)
	eZ80F91 Module Product Specification (PS0193)
ZTP	ZiLOG TCP/IP Stack v1.4.1 API Reference Manual (RM0040)
	ZiLOG TCP/IP Software Suite v1.4 Programmer's Guide (RM0041)
	ZiLOG TCP/IP Software Suite Quick Start Guide (QS0049)
ZDS II	ZiLOG Developer Studio II - eZ80Acclaim! [™] User Manual (UM0144)



Appendix B—Source Code

This appendix provides a listing of the source code for the Webpage-EEPROM Interface application. The source code file `AN0209-SC01.zip` is available along with the Application Note on the [ZiLOG website](#).

C Files

The following C files are listed in this section:

- `eprom.c`
- `i2c.c`
- `i2c_cgi.c`

```
/*
*****
* File: eprom.c
* Description: This file contains implementations for I2C-EEPROM
* routines. This contains implementations that are specific
* to the EEPROM device, which makes use of the i2c.c
* for I2C protocol.
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*/

#include <ez80.h>
#include "eprom.h"

/*
* initialize the eZ80 I2C device; perform an ack
* polling just to ensure the target device exists
```



```
*/

UINT EEPROM_Open( VOID )
{
/* Initialize the eZ80 I2C device */
    I2C_Init() ;

    return EEPROMERR_SUCCESS ;
}/* End of EEPROM_Open() */

/*
 * Close the I2C EEPROM device
 */

UINT EEPROM_Close( VOID )
{
/** Close the eZ80 I2C device */
    return I2C_Close() ;
}/** End of EEPROM_Close() */

/*
 * Write one or more bytes into the eeprom device sends 'device
 * addressing' at the beginning of every block/page and then sends data
 * bytes to write.
 * This routine implements both the 'byte write' and 'page write'
 * operations of the eeprom device
 */

UINT EEPROM_Write( BYTE* address, BYTE* buffer, UINT16 count )
{
    UINT16 index ;
    UINT    status ;
    BYTE    *ptr ;

/*
 * Send a start condition, configure the slave address, send
 * memory address bytes, then followed by the data bytes
 * the data bytes are sent
 */

```



```
*/
/** Transmit data */
for( index=0,ptr=address; index<count; index++,ptr++ )
{
    /*
    * Initiate 'device addressing' before sending the 0th byte
    * or sending the bytes positioned at the beginning of pages
    */
    if((( (UINT32)ptr % EEPROM_PAGESIZE) == 0) || ( index==0 ) )
    {
        /** Perform ack polling before sending a new page */
        EEPROM_AckPolling() ;

        /* Device addressing */
        status = EEPROM_DeviceAddressing( ptr, I2CMODE_MASTERTRANSMIT ) ;

        /** 'Device addressing' failed */
        if( status != EEPROMERR_SUCCESS )
        {
            return status ;
        }
    }
    /* End 'initiate device addressing' */

    /* Send a byte */
    status = I2C_TransmitDataByte( buffer[ index ] ) ;
    status = EEPROM_GetError( status ) ;

    /* 'Send a byte' failed */
    if( status != EEPROMERR_SUCCESS )
    {
        return status ;
    }
}
/** Send a STOP for the last byte of the buffer/page */
if(
```



```
(UINT32) (ptr+1)%EEPROM_PAGESIZE == 0/** Last byte of a page*/
        ||                               /** OR*/
        (index+1) == count)           /** Last byte of the buffer */

{
    I2C_SendStop() ;
}

}/** end for:'transmit data' */

return EEPROMERR_SUCCESS ;

}/** end of EEPROM_Write() */

/**
 * Read one or more bytes from the eeprom device
 * -set the eeprom device address pointer with the 'address'
 * then perform read for 'count' number of times
 *
 * This routine implements only the 'sequential read' operation
 * of the eeprom device
 */
UINT EEPROM_Read( BYTE* address, BYTE* buffer, UINT16 count )
{
    UINT16 index ;
    UINT    status ;
    VOID    *ptr ;
    BYTE    databyte ;
    BYTE    temp ;

    /** Perform an ack polling */

    EEPROM_AckPolling() ;

    /**
     * Set eeprom device address pointer with the value of
     * 'address' by performing a dummy write operation on the device
     */
}
```



```
ptr = address ;

/** Set the address pointer --'device addressing' will do this job */
status = EEPROM_DeviceAddressing( ptr, I2CMODE_MASTERTRANSMIT ) ;
status = EEPROM_GetError( status ) ;

/** 'Device addressing' failed */

if( status != EEPROMERR_SUCCESS )
{
    return status ;
}

/** Device addressing for master receive */

status = EEPROM_DeviceAddressing( ptr, I2CMODE_MASTERRECEIVE ) ;
status = EEPROM_GetError( status ) ;

/** 'Device addressing' failed */

if( status != EEPROMERR_SUCCESS )
{
    return status ;
}

/** Now that the eeprom device is ready to transmit, perform reads */
for( index=0; index<count; index++ )
{
    /* Clear iflag to receive data bytes from the slave */

    /* This byte is the last one to be read */
    if( (index+1) == count )
    {
        /*
         * A 'not ack' has to be sent on receiving this
         * byte
         * reset aak and clear iflag */

        I2C_ResetAAKClearIFlag() ;
    }
    else
```



```
{
    /**
     * An 'ack' has to be sent on receiving this
     * byte
     *
     * --set aak and clear iflag
     */
    I2C_SetAAKClearIFlag( temp ) ;
}

/** Receive the data byte */

status = I2C_ReceiveDataByte( &databyte ) ;

/**
 * A data byte received
 * --read and validate the status before reading the data
 */

/** Not all the bytes received */

if( (index+1) != count )
{
    /** A 'not ack' was transmitted */

    if( status != I2CERR_DATABYTERXDMMODE_ACKTXD )
    {
        status = EEPROMERR_FAILURE ;
        break ;
    }

    /** End 'not ack transmitted' */
}

/** End 'not all bytes received' */

else /** All the bytes received */
{
    /** An ack was transmitted */
    if( status != I2CERR_DATABYTERXDMMODE_NACKTXD )
    {
        status = EEPROMERR_FAILURE ;
        break ;
    }

    /** End 'ack transmitted' */
}
```



```
        /** Send STOP */
        I2C_SendStop() ;

    }/** End 'all bytes received' */

    /** Read the data into buffer */

    buffer[ index ] = databyte ;
    status = EEPROM_GetError( status ) ;

}/** End 'for' */

/** Set the AAK bit back */

I2C_SetAAK() ;

return status ;

}/** End of EEPROM_Read() */

// eeprom supporting functions
// Acknowledge polling
UINT EEPROM_AckPolling( VOID )
{
    UINT status ;
    UINT32 i ;

    // Send a START, send write control byte, then poll for ack
    // typical timeout 5ms

    for(i=0;i<0xffff;i++)
    {
        status = EEPROM_SendStartWriteControlByte() ;

        if( status == EEPROMERR_SUCCESS )
        {
```



```
        break ;
    }
}

return status ;

}/** End of EEPROM_AckPolling() */

// Send a START, then followed by control byte for a write

UINT EEPROM_SendStartWriteControlByte( VOID )
{
    return EEPROM_GetError(I2C_ConfigSlaveAddress( EEPROM_SLVADD,
                                                    I2CMODE_MASTERTRANSMIT ) ) ;
}

// End of EEPROM_SendStartWriteControlByte()

// Sends a START, the control byte, the address high byte, then the
// address low byte

UINT EEPROM_DeviceAddressing(VOID* address, I2CENUM_ModeMask modemask)
{
    UINT status ;

    // Send a start condition, configure the slave address, send
    // memory address bytes

    // Send a start condition, and the control byte
    // --configure slave address

    status = I2C_ConfigSlaveAddress( EEPROM_SLVADD, modemask ) ;
    status = EEPROM_GetError( status ) ;

    // 'config slave address' failed

    if( status != EEPROMERR_SUCCESS )
    {
        return status ;
    }
}
```



```
// if eZ80 I2C is set to receive mode, return the status from here

if( modemask == I2CMODE_MASTERRECEIVE )
    return status ;

// Send address high byte

status = I2C_TransmitDataByte( (BYTE)((UINT16)address>>8) ) ;
status = EEPROM_GetError( status ) ;

// 'send address high byte' failed

if( status != EEPROMERR_SUCCESS )
{
    return status ;
}

// now, send address low byte

status = I2C_TransmitDataByte( (BYTE)address ) ;
status = EEPROM_GetError( status ) ;
return status ;

} // End of EEPROM_DeviceAddressing()

// Translates the i2c error codes to the corresponding eeprom error
// codes to the higher layers of the application. This routine can be
// enhanced to widen the error interpretation at higher layers of the
// application

UINT EEPROM_GetError( UINT status )
{
    switch( status )
    {
        case I2CERR_SUCCESS:                // success */
        case I2CERR_DATABYTERXDMMODE_ACKTXD: // data byte received in
                                                // master mode, ack
                                                // transmitted

        case I2CERR_DATABYTERXDMMODE_NACKTXD: // data byte received in
                                                // master mode, not ack
                                                // transmitted

    }
}
```



```

        status = EEPROMERR_SUCCESS ;
        break ;
    }

    case I2CERR_STARTFAIL:           // fail to send start bit
    case I2CERR_ARBLOST:             // arbitration lost
    case I2CERR_ARBLOST_SLAWRXD_ACKTXD:// arbitration lost, slave
                                        // address + W bit received, ack
                                        // transmitted
    case I2CERR_ARBLOST_SLARRXD_ACKTXD:// arbitration lost, slave
                                        // address + R bit received, ack
                                        // transmitted
    case I2CERR_ARBLOST_GLCLADRXD_ACKTXD:// arbitration lost, general
                                        // call address received, ack
                                        // transmitted
    case I2CERR_SLAWTXD_ACKNRXD: // slave address + W bit transmitted,
                                        // acknowledge not received
    case I2CERR_SLARTXD_ACKNRXD: // slave address + R bit transmitted,
                                        // ack not received
    case I2CERR_DATABYTETXDMODE_ACKNRXD:// data byte transmitted in
                                        // master mode, ack not received
    {
        status = EEPROMERR_BUSERROR ;
        break ;
    }

    case I2CERR_TIMEOUT:             // timeout occurred
    {
        status = EEPROMERR_TIMEOUT ;
        break ;
    }

    break ;

// unkown i2c error/status

    case I2CERR_FAILURE:
    default:
    {
        status = EEPROMERR_FAILURE ;
        break ;
    }

}/** end of 'switch' */

```



```
    return status ;

}/** end of EEPROM_GetError() */

/
*****
***** end of file *****
*****
*/

/*
*****
* File:          i2c.c
* Description:   This contains implementations for I2C protocol
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*/

#include <ez80.h>
#include "i2c_an0209.h"

// Initialize I2C

VOID I2C_Init( VOID )
{
    I2C_SRR = I2C_SRESET ;
```



```
// set the clock register for configuring the I2C sampling frequency
// (Fsamp) and the I2C clock output frequency (Fscl)
// The Fsamp should at least be 1MHz (for 100kbps) and be 4 MHz (for
// 400kbps)
// For Fsamp=50 MHz, it would result in Fscl=416 KHz and it turns out
// to: N=0 and M=11.
// CCR Register: field --0(1):M(11):N(0)
// value --0:1011:000 = 0x58

    I2C_CCR = 0X58;
    I2C_CTL = 0x44 ;                // set aak and enab bit
    return ;

} // end of I2C_Init()

// Configure slave address with user given value for master transmit or
// receive mode

UINT I2C_ConfigSlaveAddress( BYTE slave_address, I2CENUM_ModeMask
                             modemask )
{
    BYTE ctlbyte ;
    UINT status ;
    UINT count = 0 ;

    status = I2C_SendStart() ; // send a start condition to begin the
                               // transaction

    if( status != I2CERR_SUCCESS )
    {
        I2C_SendStop() ;           // fail!!!
        I2C_Init() ;              // soft reset the I2C
        return status ;
    } // end 'start failed'

    ctlbyte = slave_address | modemask ; // frame the control byte
    I2C_DR = ctlbyte ;              // load the control byte to be
                                   // transmitted
    I2C_ClearIFlag() ;             // clear iflag bit to transmit
                                   // the control byte
    I2C_POLL_IFLG( count, COUNT_POLL_IFLG ) ; // poll on iflag

    if( I2C_CTL & I2CMASK_GET_IFLG ) // iflag set
```



```
{
    status = I2CERR_FAILURE ;/** default the status to failure */

    switch( I2C_SR )/** check status */
    {

        // slave address is transmitted successfully
        // slave address + W transmitted, ack received

        case I2CSTAT_SLAWTXD_ACKRXD:
        {
            if( modemask == I2CMODE_MASTERTRANSMIT )
            {
                status = I2CERR_SUCCESS ;
            }

            break ;
        }

        // slave address + R transmitted, ack received

        case I2CSTAT_SLARTXD_ACKRXD:
        {
            if( modemask == I2CMODE_MASTERRECEIVE )
            {
                status = I2CERR_SUCCESS ;
            }

            break ;
        }

        // slave address + W transmitted, ack not received

        case I2CSTAT_SLAWTXD_ACKNRXD:
        {
            status = I2CERR_SLAWTXD_ACKNRXD ;
            break ;
        }

        // slave address + R transmitted, ack not received

        case I2CSTAT_SLARTXD_ACKNRXD:
        {
            status = I2CERR_SLARTXD_ACKNRXD ;
        }
    }
}
```



```
        break ;
    }
    // arbitration lost

    case I2CSTAT_ARBLOST:
    {
        status = I2CERR_ARBLOST ;
        break ;
    }

    // arbitration lost, slave address + W received, ack
    // transmitted

    case I2CSTAT_ARBLOST_SLAWRXD_ACKTXD:
    {
        status = I2CERR_ARBLOST_SLAWRXD_ACKTXD ;
        break ;
    }

    // arbitration lost, slave address + R received, ack
    // transmitted

    case I2CSTAT_ARBLOST_SLARRXD_ACKTXD:
    {
        status = I2CERR_ARBLOST_SLARRXD_ACKTXD ;
        break ;
    }

    // arbitration lost, general call address received, ack
    // transmitted

    case I2CSTAT_ARBLOST_GLCLADRXD_ACKTXD:
    {
        status = I2CERR_ARBLOST_GLCLADRXD_ACKTXD ;
        break ;
    }

    }// end 'switch'

} // end of 'iflag is set'

// timeout

else
```



```
{
    status = I2CERR_TIMEOUT ;

} // end 'timeout'

return status ;

} // end of I2C_ConfigSlaveAddress()

// Send START condition and return the status to the caller

UINT I2C_SendStart( VOID )
{
    UINT count = 0 ;
    UINT status ;
    BYTE temp ;

    // Enter master transmit mode: send a start condition to begin the
    // transmission

    temp = I2C_CTL ;
    temp = temp | I2CMASK_SET_STA ;
    temp = temp & I2CMASK_CLR_IFLG ;
    I2C_CTL = temp ;

    I2C_POLL_IFLG( count, COUNT_POLL_IFLG ) ; // poll on iflag

    if( I2C_CTL & I2CMASK_GET_IFLG )          // iflag set
    {
        // START or repeated START transmitted

        if((I2C_SR == I2CSTAT_STARTTXD) || (I2C_SR == I2CSTAT_RPTDSTARTTXD))
            status = I2CERR_SUCCESS ;
        else
            status = I2CERR_STARTFAIL ;      // START fail
    } // end 'iflag set'

    else
    {
        status = I2CERR_TIMEOUT ; // timeout
    }
}
```



```
    }// end 'timeout' */
    return status ;

} // end of I2C_SendStart()

VOID I2C_SendStop( VOID )           // send stop condition
{
    BYTE temp ;
    temp = I2C_CTL ;
    temp = temp | I2CMASK_SET_STP ;
    temp = temp & I2CMASK_CLR_IFLG ;
    I2C_CTL = temp ;
    return ;

} // end of I2C_SendStop()

// Close the I2C communication by soft-resetting the setup

UINT I2C_Close( VOID )
{
    I2C_SRR = I2C_SRESET ;           // soft reset the I2C

    // Make sure the reset has happened by checking the STA, STP and IFLG
    // bits for zeros

    if( I2C_CTL & I2CMASK_SRESET_CONFIRM )
        return I2CERR_CLOSEFAIL ;

    return I2CERR_SUCCESS ;

} // end of I2C_Close()

// transmits the given data byte over the I2C bus

UINT I2C_TransmitDataByte( BYTE data )
{
    UINT count = 0 ;
    UINT status ;
```



```
I2C_DR = data ;           // place the data byte to be
                          // transmitted in the data register

I2C_ClearIFlag() ;       // clear iflag bit to transmit the data
                          // byte

// poll on iflag to see if the data byte transfer is complete

I2C_POLL_IFLG( count, COUNT_POLL_IFLG ) ;

// iflag is set

if( I2C_CTL & I2CMASK_GET_IFLG )
{
    switch( I2C_SR )
    {
        case I2CSTAT_DATABYTETXDMMODE_ACKRXD:
        {
            status = I2CERR_SUCCESS ;
            break ;
        }

        case I2CSTAT_DATABYTETXDMMODE_ACKNRXD:
        {
            status = I2CERR_DATABYTETXDMMODE_ACKNRXD ;
            break ;
        }

        case I2CSTAT_ARBLOST:
        {
            status = I2CERR_ARBLOST ;
            break ;
        }

        default:
        {
            status = I2CERR_FAILURE ;
            break ;
        }
    }

} // end of 'switch'
```



```
    }// end 'iflag set'

    // timeout
    else
    {
        status = I2CERR_TIMEOUT ;

    }// end 'timeout'

    return status ;

} // end of I2C_TransmitDataByte()

// receives a data byte over the I2C bus

UINT I2C_ReceiveDataByte( BYTE *data )
{
    UINT count = 0 ;
    UINT status ;

    // poll on iflag

    I2C_POLL_IFLG( count, COUNT_POLL_IFLG ) ;

    // iflag is set

    if( I2C_CTL & I2CMASK_GET_IFLG )
    {
        while((I2C_SR == I2CSTAT_SLARTXD_ACKRXD) &&
            (count < COUNT_POLL_SR) ) count++ ;

        // read the data

        *data = I2C_DR ;

        // data byte received, ack transmitted

        if( I2C_SR == I2CSTAT_DATABYTERXDMMODE_ACKTXD )
        {
            status = I2CERR_DATABYTERXDMMODE_ACKTXD ;

        } // end 'data received, ack transmitted'
```



```
// data byte received, not ack transmitted

else if( I2C_SR == I2CSTAT_DATAbyterxmode_NACKTXD )
{
    status = I2CERR_DATAbyterxmode_NACKTXD ;
}

// end 'data received, not ack transmitted'

// arbitration lost

else if( I2C_SR == I2CSTAT_ARBLOST )
{
    status = I2CERR_ARBLOST ;
}

// end 'arb lost'

// unknown error
else
{
    status = I2CERR_FAILURE ;
}

// end 'unknown' error

}

// end 'iflag set'

// timeout

else
{
    status = I2CERR_TIMEOUT ;
}

// end 'timeout'

return status ;

}

// end of I2C_ReceiveDataByte()

/*
*****
***** end of file *****
*****
*/
```



```
/*
*****
* File:          i2c_cgi.c
* Description:   This file contains CGI functions for invoking the write
*               and read on I2C device.
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*/

#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include "ZSysgen.h"
#include "ZTypes.h"
#include "basetypes.h"
#include "ZThread.h"
#include "ZDevice.h"
#include "XtlTcp.h"
#include "emulator.h"
#include "xc_lib.h"
#include "kernel.h"
#include "http.h"
#include "httpd.h"
#include <socket.h>
#include <eeprom.h>

#define MACHINE1 1 // SET MACHINE ID
#define MACHINE2 2
#define MACHINE3 3
```



```
extern UINT EEPROM_Open(VOID);
extern UINT EEPROM_Close(VOID);
extern UINT EEPROM_Write( BYTE* address, BYTE* buffer, UINT16 count );
extern UINTEEPROM_Read( BYTE* address, BYTE* buffer, UINT16 count );

char i2c_done[] = {"<html><title>Update Status</title><body
bgcolor=#D2C87B><p align = \"left\"><font face=\"arial\"
color=\"black\" size=\"2\">DATA IS UPDATED SUCCESSFULLY TO I2C EEPROM</
font></body></html>"};

char body_tag[] = {"<html><head></head><body bgcolor=#D2C87B><div
align=\"center\"><center><br><table width=\"330\" border=\"0\"
><td><font face=\"arial\" color=\"black\" size=\"2\">"};

char body_tag1[] ={"<br></td></font></table></body></html>"};

char read_m1[] = {"<body topmargin=\"0\"><p align=\"center\"><font
face=\"arial\" size=\"3\" color=\"red\"><strong>Reading Machine 1
Data</strong></font></body>"};

char read_m2[] = {"<body topmargin=\"0\"><p align=\"center\"><font
face=\"arial\" size=\"3\" color=\"red\"><strong>Reading Machine 2
Data</strong></font></body>"};

char read_m3[] = {"<body topmargin=\"0\"><p align=\"center\"><font
face=\"arial\" size=\"3\" color=\"red\"><strong>Reading Machine 3
Data</strong></font></body>"};

    SID SemaphoreID;

// This function reads the input variables from the web page and write
// the data to appropriate location on I2C EEPROM

int i2cwrite_cgi(struct http_request *request)
{
    short int machine_no;
    char *max_temp,*min_temp,*max_voltage,*min_voltage, *machine_id;
    http_output_reply(request,HTTP_200_OK);

    max_temp = http_find_argument(request,(unsigned char*)"mat");
    min_temp = http_find_argument(request,(unsigned char*)"mit");
    max_voltage = http_find_argument(request,(unsigned char*)"mav");
    min_voltage = http_find_argument(request,(unsigned char*)"miv");
    machine_id = http_find_argument(request,(unsigned char*)"hidden");
}
```



```
machine_no = atoi(machine_id);
wait( SemaphoreID );

if(machine_no == MACHINE1)
{
    unsigned char *ptr = (unsigned char*)0x0000 ;// WRITE DATA FROM
                                                // LOCATION 0X0000

    EEPROM_Open();
    EEPROM_Write( ptr,(unsigned char*) max_temp, 6 ) ;
    EEPROM_Write( ptr+6, (unsigned char*)min_temp, 6 ) ;
    EEPROM_Write( ptr+12, (unsigned char*)max_voltage, 6 ) ;
    EEPROM_Write( ptr+18, (unsigned char*)min_voltage, 6 ) ;
    EEPROM_Close() ;
    _http_write(request,i2c_done,strlen(i2c_done));
}
else if(machine_no == MACHINE2)
{
    unsigned char *ptr = (unsigned char*)0x0100;// WRITE DATA FROM
                                                // LOCATION 0x0100

    EEPROM_Open();
    EEPROM_Write( ptr,(unsigned char*) max_temp, 6 ) ;
    EEPROM_Write( ptr+6, (unsigned char*)min_temp, 6 ) ;
    EEPROM_Write( ptr+12, (unsigned char*)max_voltage, 6 ) ;
    EEPROM_Write( ptr+18, (unsigned char*)min_voltage, 6 ) ;
    EEPROM_Close() ;
    _http_write(request,i2c_done,strlen(i2c_done));
}
else if(machine_no == MACHINE3)
{
    unsigned char *ptr = (unsigned char*)0x0200;// WRITE DATA FROM
                                                // LOCATION 0x0200

    EEPROM_Open();
    EEPROM_Write( ptr, (unsigned char*)max_temp, 6 ) ;
    EEPROM_Write( ptr+6, (unsigned char*)min_temp, 6 ) ;
    EEPROM_Write( ptr+12, (unsigned char*)max_voltage, 6 ) ;
    EEPROM_Write( ptr+18,(unsigned char*) min_voltage, 6 ) ;
    EEPROM_Close();
    _http_write(request,i2c_done,strlen(i2c_done));
}
```



```
    }
    signal( SemaphoreID );
}

int i2cread_cgi(struct http_request *request)
{
    char m_temp1[100]={" The MAX temp in Deg.C for machine1: "};
    char m_temp2[100]={" The MAX temp in Deg.C for machine2: "};
    char m_temp3[100]={" The MAX temp in Deg.C for machine3: "};

    char low_temp1[100] ={"The MIN temp in Deg.C for machine1: "};
    char low_temp2[100] ={"The MIN temp in Deg.C for machine2: "};
    char low_temp3[100] ={"The MIN temp in Deg.C for machine3: "};

    char m_voltage1[100] = {"The MAX voltage set for machine1: "};
    char m_voltage2[100] = {"The MAX voltage set for machine2: "};
    char m_voltage3[100] = {"The MAX voltage set for machine3: "};

    char low_voltage1[100]={"The MIN voltage set for machine1: "};
    char low_voltage2[100]={"The MIN voltage set for machine2: "};
    char low_voltage3[100]={"The MIN voltage set for machine3: "};

    short int machine_no;

    char max_temp[10],min_temp[10],max_voltage[10],min_voltage[10],
        *machine_id;

    http_output_reply(request,HTTP_200_OK);

    machine_id = http_find_argument(request,(unsigned char*)"hidden");
    machine_no = atoi(machine_id);
    wait( SemaphoreID );

    if(machine_no == MACHINE1)
    {
        unsigned char *ptr = (unsigned char*)0x0000 ;// Read DATA FROM
        // LOCATION 0X0000

        EEPROM_Open();
        EEPROM_Read( ptr,(unsigned char*) max_temp, 6 ) ;
        EEPROM_Read( ptr+6, (unsigned char*)min_temp, 6 ) ;
    }
}
```



```
EEPROM_Read( ptr+12, (unsigned char*)max_voltage, 6 ) ;
EEPROM_Read( ptr+18, (unsigned char*)min_voltage, 6 ) ;
EEPROM_Close() ;

strcat(m_temp1,max_temp);
strcat(low_temp1,min_temp);
strcat(m_voltage1,max_voltage);
strcat(low_voltage1,min_voltage);

_http_write(request,read_m1,strlen(read_m1));
_http_write(request,body_tag,strlen(body_tag));
_http_write(request,m_temp1,strlen(m_temp1));
_http_write(request,"<br><br>",8);
_http_write(request,low_temp1,strlen(low_temp1));
_http_write(request,"<br><br>",8);
_http_write(request,m_voltage1,strlen(m_voltage1));
_http_write(request,"<br><br>",8);
_http_write(request,low_voltage1,strlen(low_voltage1));
_http_write(request,body_tag1,strlen(body_tag1));
}
else if(machine_no == MACHINE2)
{
    unsigned char *ptr = (unsigned char*)0x0100;// Read DATA FROM
                                                // LOCATION 0x0100

    EEPROM_Open();
    EEPROM_Read( ptr,(unsigned char*) max_temp, 6 ) ;
    EEPROM_Read( ptr+6, (unsigned char*)min_temp, 6) ;
    EEPROM_Read( ptr+12, (unsigned char*)max_voltage, 6 ) ;
    EEPROM_Read( ptr+18, (unsigned char*)min_voltage, 6 ) ;
    EEPROM_Close() ;

    strcat(m_temp2,max_temp);
    strcat(low_temp2,min_temp);
    strcat(m_voltage2,max_voltage);
    strcat(low_voltage2,min_voltage);

    _http_write(request,read_m2,strlen(read_m2));
    _http_write(request,body_tag,strlen(body_tag));
    _http_write(request,m_temp2,strlen(m_temp2));
    _http_write(request,"<br><br>",8);
    _http_write(request,low_temp2,strlen(low_temp2));
    _http_write(request,"<br><br>",8);
    _http_write(request,m_voltage2,strlen(m_voltage2));
    _http_write(request,"<br><br>",8);
}
```



```

        _http_write(request,low_voltage2,strlen(low_voltage2));
        _http_write(request,body_tag1,strlen(body_tag1));
    }
else if(machine_no == MACHINE3)
{
    unsigned char *ptr = (unsigned char*)0x0200;// Read DATA FROM
                                                // LOCATION 0x0200

    EEPROM_Open();
    EEPROM_Read( ptr, (unsigned char*)max_temp, 6 );
    EEPROM_Read( ptr+6, (unsigned char*)min_temp, 6 );
    EEPROM_Read( ptr+12, (unsigned char*)max_voltage, 6 );
    EEPROM_Read( ptr+18, (unsigned char*) min_voltage, 6 );
    EEPROM_Close();

    strcat(m_temp3,max_temp);
    strcat(low_temp3,min_temp);
    strcat(m_voltage3,max_voltage);
    strcat(low_voltage3,min_voltage);

    _http_write(request,read_m3,strlen(read_m3));
    _http_write(request,body_tag,strlen(body_tag));
    _http_write(request,m_temp3,strlen(m_temp3));
    _http_write(request,"<br><br>",8);
    _http_write(request,low_temp3,strlen(low_temp3));
    _http_write(request,"<br><br>",8);
    _http_write(request,m_voltage3,strlen(m_voltage3));
    _http_write(request,"<br><br>",8);
    _http_write(request,low_voltage3,strlen(low_voltage3));
    _http_write(request,body_tag1,strlen(body_tag1));
}
signal( SemaphoreID );
}
/
*****
***** end of file *****
*****
*/

```

Header Files

The following Header files are presented in this section:

- eeprom.h
- i2c_an0209.h



```
/*
*****
* File:          eeprom.h
* Description:   Header file
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*/

#ifndef EEPROM
#define EEPROM

#include "i2c_an0209.h"

#define EEPROM_PAGESIZE      64
#define EEPROM_SLVADD        0xA0

/** prototypes */

UINT EEPROM_Open( VOID ) ;
UINT EEPROM_Close( VOID ) ;
UINT EEPROM_Write( BYTE*, BYTE*, UINT16 ) ;
UINT EEPROM_Read( BYTE*, BYTE*, UINT16 ) ;
UINT EEPROM_AckPolling( VOID ) ;
UINT EEPROM_GetError( UINT ) ;
UINT EEPROM_DeviceAddressing( VOID*, I2CENUM_ModeMask ) ;
UINT EEPROM_SendStartWriteControlByte( VOID ) ;

enum eeprom_errors
{
    EEPROMERR_SUCCESS,
    EEPROMERR_BUSERROR,

```



```
    EEPROMERR_TIMEOUT,
    EEPROMERR_FAILURE
} ;

/
*****
***** end of file *****
*/

/*
*****
* File:          i2c_an0209.h
* Description:   Header file
*
* Copyright 2004 ZiLOG Inc. ALL RIGHTS RESERVED.
*
* The source code in this file was written by an
* authorized ZiLOG employee or a licensed consultant.
* The source code has been verified to the fullest
* extent possible.
*
* Permission to use this code is granted on a royalty-free
* basis. However users are cautioned to authenticate the
* code contained herein.
*
* ZiLOG DOES NOT GUARANTEE THE VERACITY OF THE SOFTWARE.
*****
*/

#include "basetypes.h"

/** I2C masks */

#define I2CMASK_SRESET_CONFIRM    0x38
#define I2CMASK_SET_STA          0x20
#define I2CMASK_SET_STP          0x10
#define I2CMASK_SET_AAK          0x04
#define I2CMASK_CLR_AAK          0xFB
#define I2CMASK_GET_IFLG         0x08
#define I2CMASK_CLR_IFLG         0xF7
#define I2CMASK_CLR_IFLGAAK      0xF3
```



```
// macros

#define COUNT_POLL_IFLG          500//5000
#define COUNT_POLL_SR           0x00FFFF
#define I2C_SRESET               0x12           /** arbitrary value */

// I2C specific

#define I2C_POLL_IFLG( x, y ) while( (!(I2C_CTL & I2CMASK_GET_IFLG)) &&
                                     (x < y) ) x++
#define I2C_SetAAK() I2C_CTL = I2C_CTL | I2CMASK_SET_AAK
#define I2C_ResetAAK() I2C_CTL = I2C_CTL & I2CMASK_CLR_AAK
#define I2C_ClearIFlag() I2C_CTL = I2C_CTL & I2CMASK_CLR_IFLG
#define I2C_ResetAAKclearIFlag() I2C_CTL = I2C_CTL & I2CMASK_CLR_IFLGAAK
#define I2C_SetAAKclearIFlag( x )  x = I2C_CTL ; \
                                     x = x | I2CMASK_SET_AAK ; \
                                     x = x & I2CMASK_CLR_IFLG ; \ I2C_CTL = x

// I2C status codes --for use within the i2c layer only

enum i2c_status
{
    I2CSTAT_BUSERROR           = 0x00,
    I2CSTAT_STARTTXD          = 0x08,
    I2CSTAT_RPTDSTARTTXD      = 0x10,
    I2CSTAT_SLAWTXD_ACKRXD    = 0x18,
    I2CSTAT_SLAWTXD_ACKNRXD   = 0x20,
    I2CSTAT_DATABYTETXDMMODE_ACKRXD = 0x28,
    I2CSTAT_DATABYTETXDMMODE_ACKNRXD = 0x30,
    I2CSTAT_ARBLOST           = 0x38,
    I2CSTAT_SLARTXD_ACKRXD    = 0x40,
    I2CSTAT_SLARTXD_ACKNRXD   = 0x48,
    I2CSTAT_DATABYTERXDMMODE_ACKTXD = 0x50,
    I2CSTAT_DATABYTERXDMMODE_NACKTXD = 0x58,
    I2CSTAT_ARBLOST_SLAWRXD_ACKTXD = 0x68,
    I2CSTAT_ARBLOST_GLCLADRXD_ACKTXD = 0x78,
    I2CSTAT_ARBLOST_SLARRXD_ACKTXD = 0xb0
} ;
```



```
// I2C error codes --for use outside the i2c layer

enum i2c_errors
{
    I2CERR_SUCCESS,
    I2CERR_TIMEOUT,

    I2CERR_STARTFAIL,
    I2CERR_CLOSEFAIL,

    // status

    I2CERR_SLAWTXD_ACKNRXD,
    I2CERR_SLARTXD_ACKNRXD,

    I2CERR_DATABYTERXDMMODE_ACKTXD,
    I2CERR_DATABYTERXDMMODE_NACKTXD,
    I2CERR_DATABYTETXDMMODE_ACKNRXD,
    I2CERR_ARBLOST,
    I2CERR_ARBLOST_SLAWRXD_ACKTXD,
    I2CERR_ARBLOST_SLARRXD_ACKTXD,
    I2CERR_ARBLOST_GLCLADRXD_ACKTXD,

    I2CERR_FAILURE
} ;

//I2C modes

typedef enum I2CENUM_ModeMask_t
{
    I2CMODE_MASTERTRANSMIT = 0x00,
    I2CMODE_MASTERRECEIVE = 0x01,
    I2CMODE_SLAVETRANSMIT,
    I2CMODE_SLAVERECEIVE
} I2CENUM_ModeMask ;

//function prototypes

VOID I2C_Init( VOID ) ;
UINT I2C_ConfigSlaveAddress( BYTE, I2CENUM_ModeMask ) ;
UINT I2C_SendStart( VOID ) ;
VOID I2C_SendStop( VOID ) ;
```



```
UINT I2C_Close( VOID ) ;
UINT I2C_Stop( VOID ) ;
UINT I2C_TransmitDataByte( BYTE data ) ;
UINT I2C_ReceiveDataByte( BYTE *data ) ;

/*
*****
***** end of file *****
*****
*/
```