

EMBEDDED ETHERNET REFERENCE DESIGN

Relevant Devices

This application note applies to the following devices:

C8051F120, C8051F121, C8051F122, C8051F123, C8051F124, C8051F125, C8051F126, and C8051F127.

Introduction

This reference design demonstrates how to set up an Embedded Ethernet web server on a C8051F12x device. It provides instructions on how to set up and use the AB2 Ethernet Daughter Cards and example web servers that use the CMX Micronet™ protocol stack. It also provides example low-level Ethernet transmit and receive routines.

Note: We encourage you to read all documentation before starting a new project.

This reference design includes:

- A schematic and bill of materials for the AB2 Ethernet board.
- Example Ethernet transmit and receive routines that communicate with a CS8900A Ethernet Controller.
- A simple example of configuring and using the CMX Micronet™ HTTP Web Server.
- Instructions on how to create a simple “Hello World” web server.
- Downloadable object files in HEX format for the “Hello World” server and “uWeb”, an interactive temperature measurement demo server.

Building an Ethernet Interface

The Ethernet interface to the C8051F12x consists of an Ethernet Adapter, an RJ-45 connector, and magnetics. The AB2 Ethernet cards available from the Silicon Labs website use a Cirrus Logic CS8900A Ethernet adapter and a Trans-Power RJ-45 connector with integrated magnetics and LEDs. The AB2 board schematic can be found in Appendix E. For more information about interfacing to a CS8900A, please visit the Cirrus Logic website at www.cirruslogic.com.

Step by Step Guide to Setting Up the CMX Micronet™ HTTP Web Server (Version 2.17j)

1. The first step in using the CMX Micronet™ protocol stack is to install it in a known directory. In the following text, we refer to this directory as the ‘Micronet’ directory.

Creating a New Project

2. To create a new project, copy all files from the ‘Micronet\netlib’ directory to a new directory. We refer to the folder as the ‘HelloWorld’ directory. Copy the files in Table 1 into the ‘HelloWorld’ directory.

Table 1. Files to Copy to the ‘HelloWorld’ Directory

1. All files in ‘Micronet\netlib’
2. ‘callback.c’ found in the ‘Micronet\k51app’ folder

Table 1. Files to Copy to the ‘HelloWorld’ Directory

3. ‘STARTUP.A51’ found in AN001SW.zip
4. ‘sfrdefs.h’ found in AN001SW.zip
5. ‘main.c’ found in AN001SW.zip

- Next create a subdirectory called ‘HTML’ in the ‘HelloWorld’ folder. Copy the files listed in Table 2 to the ‘HTML’ folder. This folder contains all HTML files, graphics, Java applets, and a copy of ‘mn_defs.h’. It is also handy, but not required, to place a copy of ‘html2c.exe’ in this folder.

Table 2. Files to Copy to the ‘HTML’ Folder

1. ‘index.h’ found in AN001SW.zip
2. ‘index.c’ found in AN001SW.zip
3. ‘index.html’ found in AN001SW.zip
4. ‘mn_defs.h’ found in the project folder
5. ‘html2c.exe’ found in the ‘Micro-net\Tools’ folder (optional)

- Start a new project in the Silicon Labs IDE. Add all ‘.C’ files in the project directory to the project except for ‘RTL8019.c’, ‘physmc65’ and ‘ipseth16.c’. Also add the files in Table 3 to the project.

Table 3. Files to Add to a New Project

1. All ‘.C’ files in the project directory
2. ‘STARTUP.A51’
3. ‘mnconfig.h’
4. ‘mn_env.h’
5. ‘micronet.h’

The ‘STARTUP.A51’ file is necessary for this project because the CMX protocol stack uses

re-entrant functions and the *LARGE* memory model. The ‘mnconfig.h’, ‘mn_env.h’, and ‘micronet.h’ header files are added to the project for convenience.

- Add a new group to the project. Add all ‘.C’ files in the ‘HTML’ directory to this group. Separating project files into several groups is recommended because it makes searching for a specific file much easier.
- Next select the *LARGE* memory model from the *Compiler* tab of the ‘Tool Chain Integration’ window in the *Project* menu. Be sure to save the project settings using the *Save Project* command.

Configuring the CMX Micronet™ Protocol Stack

This section outlines the changes required for the CMX Micronet™ HTTP Web Server to run on a C8051F12x device. The following steps involve modifying constants or small blocks of code in select files.

Modifying ‘STARTUP.A51’

- Depending on your version of ‘STARTUP.A51’, you may need to add the following include statement to the beginning of the file:

```
$include (C8051F120.H)
```

- This step initializes the re-entrant stack pointer. This is accomplished by setting XBPSTACK to ‘1’ and XBPSTACKTOP to ‘1FFFH+1’. XBPSTACKTOP is set to the highest memory location in XRAM plus one. For 8KB of XRAM on ‘F12x devices, this equates to 0x2000.

Modifying 'micronet.h'

9. Figure 1 contains bolded definitions that should be added at the beginning of 'micronet.h' to specify the base address of the CS8900A in external memory and that the Ethernet controller is being used in polled mode.

Modifying 'mn_env.h'

10. In 'mn_env.h' header file, change the DYNAMIC_MEM_AVAILABLE constant to '0'.

Modifying 'mn_port.c'

11. In 'mn_port.c', modify the Timer 0 reload value to overflow every 10 ms. For example, if SYCLK is 49 MHz and Timer 0 is clocked by SYCLK/12, then the reload value should be $-(\text{SYCLK}/12/100) = -40833d = 0x607F$.

Modifying 'mnconfig.h'

12. The 'mnconfig.h' header file allows the user to select the protocols used, size of buffers, and other options. Disabling unnecessary options will save memory and code space. Table 4 shows the options enabled for the 'HelloWorld' web server.

All other protocols and options are assumed to be inactive or at their default state.

Table 4. Constant Settings in 'mnconfig.h'

Constant	Value
TCP	1
ETHERNET	1
SLIP	0
PING	1
NUM_SOCKETS	2
RECV_BUFF_SIZE	1024
POLLED_ETHERNET	1
ARP	1
ARP_TIMEOUT	0
ARP_AUTO_UPDATE	1
ARP_CACHE_SIZE	2
DHCP	0
HTTP	1
SERVER_SIDE_INCLUDES	0
INCLUDE_HEAD	0
FTP	0
NEED_MEM_POOL	0

Figure 1. Definitions to Add to 'micronet.h'

```
#ifndef MICRONET_H_INC
#define MICRONET_H_INC 1

#define POL8051
#define iPS_ETH8_100_RDWR ((volatile unsigned char xdata*)0xC000)

#include "mn_defs.h"
#include "mn_env.h"
...
```

Table 4. Constant Settings in 'mnconfig.h'

Constant	Value
VIRTUAL_FILE	1

Designing an Application Around the CMX Micronet™ HTTP Server

All projects that use the CMX stack require the user to provide the *main()* routine, callback routines, device and protocol stack initialization routines, and HTML content.

Typical Project Structure

A typical web-enabled project structure is shown in Figure 2. In most cases, the user will only be writing the initialization and callback routines in 'main.c' and 'callback.c'. All HTML content is added to the project in the form of a file array generated by the 'html2c.exe' utility. The linker combines all project files to generate an object file that is downloaded to FLASH.

Initialization Routines

Because of device dependencies, the order of performing initializations is important for the system to function properly. The first set of initializations should include peripherals. In the software example, these would include the following function calls:

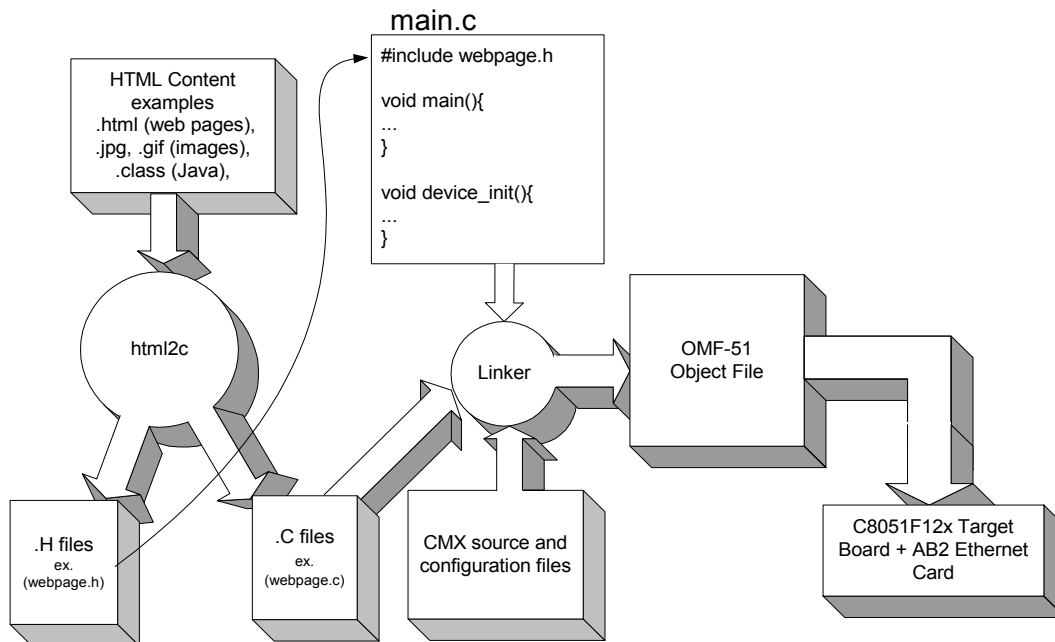
```
SYSClk_Init (); // Initialize PLL
EMIF_Init ();  // Initialize External
                // Memory Interface
PORT_Init ();  // Initialize Port I/O
UART1_Init (); // Initialize UART1
                // to 9600 baud
```

The second initialization includes programming the device IP and MAC addresses, or copying these addresses from FLASH if the device has already been programmed. This is done by the following function call:

```
ipconfig();
```

Since CMX Micronet™ uses Timer 0, the *ipconfig()* routine sets the Timer Mode (TMOD) and Clock Control (CKCON) registers to their reset value.

Figure 2. A Typical Web-Enabled Project Structure



The fourth set of initializations includes the CMX Micronet™ variables. This is accomplished with the following function call:

```
mn_init();           // initialize all
                     // protocol stack
                     // variables
```

Please note that the CMX stack variables must be initialized before any other CMX-provided functions are called.

Adding Files to the Virtual File System

The Virtual File System associates a file name with a file array generated by the 'html2c' utility. Files added to the Virtual File System can be requested from a web browser.

The 'html2c' Utility

To converting HTML content to file array, use the 'html2c.exe' utility provided with the CMX package. The 'html2c.exe' utility outputs a '.c' file and a '.h' file with the same name as the HTML content file. These source and header files declare and initialize a 'char' array in FLASH. The file arrays can represent binary or ASCII files.

Adding File Arrays to the Project

Once the source and header files for the file arrays have been created, incorporate them into the project by adding the '.c' file to the *Project Build List* and *#include*-ing the '.h' file at the top of 'main.c'.

When the file arrays have been added to the project, the final step is to create an entry in the file system for each file array by calling the *mn_vf_set_entry()* function as shown in Figure 3. In this example, 'index.html', 'image1.gif', and 'image2.gif' can be downloaded from a web browser by typing the proper URL into the address bar.

Starting the HTTP Server

The last function call in 'main.c' should start the HTTP server as follows:

```
mn_server();
```

Once the HTTP server is started, it takes control of program execution. Code that needs to be executed after the server has started must reside in a CGI script, in one of the special blank functions defined in 'callback.c', or in an interrupt service routine.

Figure 3. Adding files to the Virtual File System

```
#include "index.h"      // index.c is in the project build list
#include "image1.h"     // image1.c is in the project build list
#include "image2.h"     // image2.c is in the project build list
...
void main()
{
    ...
    // sample virtual file entries
    mn_vf_set_entry((byte *)"index.html", INDEX_SIZE, index_html, VF_PTYPE_FLASH);
    mn_vf_set_entry((byte *)"image1.gif", IMAGE1_SIZE, image1_gif, VF_PTYPE_FLASH);
    mn_vf_set_entry((byte *)"image2.gif", IMAGE2_SIZE, image2_gif, VF_PTYPE_FLASH);
    ...
}
```

A “Hello World” Application

This example project demonstrates developing a basic web-enabled application. The OMF-51 object file for this example is included as part of the reference design package. It is recommended that you run this project from the provided object file first to get a feeling for what lies ahead. We recommend following the steps in Appendix A to set up the “uWeb” server or the “Hello World” server from their object files before continuing.

Starting the “Hello World” Application From Scratch

Create a new Silicon Labs IDE project and make the necessary changes to CMX Micronet™ outlined in the previous sections. The “Hello World” project includes ‘main.c’, the CMX source files, STARTUP.A51, and a C source file named ‘index.c’, which contains the file array for the primary web page.

Description of ‘main.c’ for the “Hello World” Project

The ‘main.c’ source file contains all user written code in the ‘Hello World’ project. The *#include* statements at the top of the file and their significance to the project are listed in Table 5 .

Table 5. Header File Descriptions

Header File	Purpose
stdio.h	Used for UART communication
micronet.h	Defines all CMX-provided functions.
sfrdefs.h	Defines the Silicon Labs-specific SFRs not already defined by the CMX stack
index.h	Defines the size of the ‘index_html’ file array

The three global variables listed in Table 6 are all located in FLASH. Each of the variables has an *xdata* pointer associated with it to allow writing to FLASH memory.

Table 6. Global Variables in ‘main.c’

first_time
ip_address[4]
mac_address[6]

The <first_time> variable can be written one time after every FLASH download because it is initialized to 0xFF. It is used as a flag to indicate that the IP and MAC addresses for the device have already been assigned and are stored in the Scratchpad area of FLASH.

The <ip_address> and <mac_address> variables are stored in the Scratchpad area of FLASH. This Scratchpad area allows the device to erase and rewrite the variables as many times as desired without interfering with program code. The two Scratchpad area variables generate memory overlap warnings by the linker because the Scratchpad shares the first 256 bytes of the 64KB *CODE* address space with program code. 0Please see the C8051F12x datasheet for more information about the Scratchpad area.

The purpose of the *main()* routine is to initialize the system, to add file arrays to the Virtual File System, and to start the HTTP Web Server. Once started, the HTTP Web Server takes control of the processor. The *while(1)* loop at the end of ‘main.c’ should not be reached unless there is an error starting the server and may be replaced with error handling code. Please refer to the CMX Micronet documentation for error codes returned from the *mn_server()* function.

Creating HTML Content

HTML content consists of HTML documents, images, Java applets, or any binary file suitable for

downloading from an embedded web server. Since the 'Hello World' server only has one HTML page, the only tool needed to create it is a text editor.

Figure 4 shows the contents of 'index.html'. After it has been created in a text editor and saved in the 'HTML' subdirectory, it needs to be converted to a file array using the 'html2c.exe' utility. Open a command window and type in the commands listed in Figure 5. These commands assume the project directory is located at 'C:\Projects\HelloWorld' and that a copy of 'html2c.exe' is in the HTML folder..

Figure 5. Example Command Prompt for Converting an HTML file to 'C' Source and Header Files

```
prompt> cd\  
prompt> cd Projects  
prompt> cd HelloWorld  
prompt> cd HTML  
prompt> html2c index.html
```

The commands in Figure 5 will create 'index.c' and 'index.h'. The 'index.c' source file should be added to the *Project Build List* and the 'index.h' header file should be *#include*-ed at the top of 'main.c'. You should now be able to build the project and test the embedded web server.

Figure 4. The Contents of 'index.html'

```
<html>  
<head><title>Hello World</title><head>  
  
<body bgcolor="green" text="white" link="yellow" vlink="red" alink="blue">  
  
<h1>Hello World!</h1>  <br><br>  
  
<b>This page is served from a C8051F124 and uses the CMX Micronet TCP/IP stack. </b>  
  
</body>  
</html>
```

Appendix A—Setting Up the Embedded Web Server Demo

Materials Needed

- C8051F12x Development Kit
- AB2 Ethernet Daughter Card
- Crossover Cable (or standard Ethernet cable and access to a network).
- A PC with a free serial port. (first time only)

Preparing the Hardware

1. Connect the AB2 Ethernet Card to the C8051F12x Target Board's 96-pin connector.
2. Connect the RJ-45 connector on the AB2 Ethernet card to a PC or laptop using the crossover cable or connect both the PC and the AB2 board to the same Ethernet network using standard Ethernet cables. A crossover cable allows a direct connection between the PC and the embedded system.
3. Connect the power supply to the C8051F12x Target Board.

Downloading Object Code to FLASH

1. Connect the EC2 Serial Adapter to the C8051F12x using the 10-pin ribbon cable.
2. Connect the EC2 Serial Adapter to the COM1 port on the PC. If COM1 is not available, then any free COM port may be used.
3. Download the 'uWEB_124_1.hex' or the 'HELLOWORLD_124_1.hex' object files to FLASH using the Silicon Labs IDE or the command line FLASH programming utility available from the [Silicon Labs website](#).
4. Disconnect the EC2 Serial Adapter from the C8051F12x Target Board and cycle power after the download is complete.

Programming the IP and MAC Addresses

The first time the device is run after a FLASH download, it will automatically go into "change IP and MAC address mode". It will also enter this mode if the SW2 (P3.2) button is held down while the reset button is pressed and released.

When the device enters this mode, the green LED will start blinking. At this point the device can be controlled by a UART terminal communicating at 9600 Baud 8-N-1. When prompted, enter the IP address chosen for the embedded system. Please see Appendix B for guidelines on choosing an IP address. Also when prompted, enter the MAC address (IA) found on the AB2 Ethernet Card. This address will start with 00-0B-3C-xx-yy-zz. This procedure should be repeated once after every FLASH download.

Accessing the Web Server

1. If your embedded system is connected directly to a network, then go to Step #2. If you are using a crossover cable, then your PC must have a static IP address in order to recognize the embedded system. On Windows PCs, you may access the Internet Protocol properties by right-clicking on *My Network Places* and selecting *Properties*. Right-click on *Local Area Connection* and select *Properties* again. Select *Internet Protocol (TCP/IP)* and click *Properties* yet another time. Specify a static IP address for the PC, as shown in Appendix B, and leave all other inputs at their default values. When the operating system prompts you to add a Subnet mask, click *OK* and accept the default mask it provides.
2. Open an instance of your favorite browser and type the IP address of the embedded system

into the address bar and press the 'Enter' key.
For example,

```
http://10.10.10.163/
```

where 10.10.10.163 is the IP
address of the embedded system.

Note: Depending on the network, it may take
30 seconds or more for the network to detect
the server. This delay can be minimized by
using a crossover cable.



Appendix B—Determining an IP Address for the Embedded

Web Server

The following text outlines a few guidelines to follow when choosing an IP address for the embedded system.

- 1. Obtain information about the PC you are trying to connect to. The key pieces of information you need are the IP address and the Subnet mask. If you are using a crossover cable, you may choose any IP address for your PC as long as the Subnet mask allows it to recognize the embedded system.
- 2. The IP address chosen for the embedded system must match the PC's IP address in all bit locations where the Subnet mask is a '1' in order for the PC to recognize the embedded system. Otherwise, the PC will send it's request outside the local network.

IP Address Selection Example

The example in Figure 6 shows the IP address and Subnet mask of the PC we want to connect to the embedded web server. Since the first 24 bits of the Subnet mask are '1', the first 24 bits of the embedded web server's IP address must match the first 24 bits of the PC's IP address. The valid range of IP addresses for the embedded web server is from 10.10.10.0 to 10.10.10.254 with the exception of 10.10.10.80 since this address is already taken by the PC. 10.10.10.255 is reserved because it is the Broadcast Address for this network. An IP address is considered a broadcast address if all bits which are '0' in the Subnet mask are '1' in the IP address.

Figure 6. IP Address Selection Example

PC IP Address

10	10	10	80	(decimal)
0000 1010	0000 1010	0000 1010	0101 0000	(binary)

PC Subnet Mask

255	255	255	0	(decimal)
1111 1111	1111 1111	1111 1111	0000 0000	(binary)

Embedded Web Server IP Address

10	10	10	163	(decimal)
0000 1010	0000 1010	0000 1010	1010 0011	(binary)

Appendix C—Example ‘main.c’ Source File for the ‘Hello World’ Project

```
//-----
// main.c
//-----
//
// AUTH: FB
// DATE: 4 OCT 02
//
// Target: C8051F12x
// Tool chain: KEIL C51
//
// Description:
// This is an example of how to set up and use the CMX Micronet HTTP
// Web Server.
//

//-----
// Includes
//-----
#include <stdio.h>                // for printf
#include "micronet.h"             // for all CMX provided routines
#include "sfrdefs.h"             // for all 8-bit and 16-bit sfr
                                // definitions not found in the standard
                                // 8051 implementations.

// include the header file for each HTML content file below
#include "HTML\index.h"          // header file for 'index.html'

//-----
// 16-bit SFR Definitions for 'F12x
//-----

sfr16 DP      = 0x82;           // data pointer
sfr16 ADC0     = 0xbe;           // ADC0 data
sfr16 ADC0GT   = 0xc4;           // ADC0 greater than window
sfr16 ADC0LT   = 0xc6;           // ADC0 less than window
sfr16 RCAP2    = 0xca;           // Timer2 capture/reload
sfr16 RCAP3    = 0xca;           // Timer3 capture/reload
sfr16 RCAP4    = 0xca;           // Timer4 capture/reload
sfr16 TMR2     = 0xcc;           // Timer2
sfr16 TMR3     = 0xcc;           // Timer3
sfr16 TMR4     = 0xcc;           // Timer4
sfr16 DAC0     = 0xd2;           // DAC0 data
sfr16 DAC1     = 0xd2;           // DAC1 data
sfr16 PCA0CP5  = 0xe1;           // PCA0 Module 5 capture
sfr16 PCA0CP2  = 0xe9;           // PCA0 Module 2 capture
sfr16 PCA0CP3  = 0xeb;           // PCA0 Module 3 capture
sfr16 PCA0CP4  = 0xed;           // PCA0 Module 4 capture
sfr16 PCA0     = 0xf9;           // PCA0 counter
sfr16 PCA0CP0  = 0xfb;           // PCA0 Module 0 capture
sfr16 PCA0CP1  = 0xfd;           // PCA0 Module 1 capture

//-----
// Global CONSTANTS
//-----
#define INTCLK      24500000      // Internal oscillator frequency in Hz
```

```
#define SYSCLK      49000000          // Output of PLL derived from (INTCLK*2)
#define BAUDRATE    9600              // Baud rate of UART in bps

sbit SW2 = P3^7;                      // SW2='0' means switch pressed
sbit LED = P1^6;                      // LED='1' means ON

//-----
// Global VARIABLES
//-----
unsigned char code first_time = 0xFF;  // may be written once after
                                       // each download because it is
                                       // initialized to 0xFF

char xdata* data ptr_first_time = &first_time;

unsigned char code ip_address[4] _at_ 0x0000; // located in Scratchpad area
char xdata* data ptr_ip_address = &ip_address;

unsigned char code mac_address[6] _at_ 0x0004; // located in Scratchpad area
char xdata* data ptr_mac_address = &mac_address;

//-----
// Function PROTOTYPES
//-----
void main (void);
void SYSCLK_Init (void);
void PORT_Init (void);
void UART1_Init (void);
void EMIF_Init (void);
void ipconfig (void);

//-----
// MAIN Routine
//-----
void main (void)
{
    WDTCN = 0xde;                      // Disable watchdog timer
    WDTCN = 0xad;

    // initialize the C8051F12x
    PORT_Init ();
    SYSCLK_Init ();
    UART1_Init ();

    EMIF_Init ();

    // initialize the IP and MAC addresses and disable UART1
    ipconfig();

    // Set SFR page to LEGACY_PAGE
    SFRPAGE = LEGACY_PAGE;

    // initialize the CMX Micronet variables
    mn_init();

    // Add files to the Virtual File System. Make sure you have allocated
    // enough slots in the Virtual File system for the number of files
```

```

// you add. More slots can be allocated in 'mnconfig.h'
mn_vf_set_entry((byte *)"index.html", INDEX_SIZE, index_html, VF_PTYPE_FLASH);

// start the HTTP Server
mn_server();

while(1);                                // This point in code should never
                                          // be reached unless an error occurs

}

//-----
// Initialization Routines
//-----

//-----
// SYSCLK_Init
//-----
//
// This routine initializes the system clock to use the internal oscillator
// at 24.5 MHz multiplied by two using the PLL.
//
void SYSCLK_Init (void)
{
    int i;                                // software timer

    char SFRPAGE_SAVE = SFRPAGE;          // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;                // set SFR page

    OSCICN = 0x83;                         // set internal oscillator to run
                                          // at its maximum frequency

    CLKSEL = 0x00;                         // Select the internal osc. as
                                          // the SYSCLK source

    //Turn on the PLL and increase the system clock by a factor of M/N = 2
    SFRPAGE = CONFIG_PAGE;

    PLL0CN = 0x00;                         // Set internal osc. as PLL source
    SFRPAGE = LEGACY_PAGE;
    FL0SCL = 0x10;                         // Set FLASH read time for 50MHz clk
                                          // or less

    SFRPAGE = CONFIG_PAGE;
    PLL0CN |= 0x01;                         // Enable Power to PLL
    PLL0DIV = 0x01;                         // Set Pre-divide value to N (N = 1)
    PLL0FLT = 0x01;                         // Set the PLL filter register for
                                          // a reference clock from 19 - 30 MHz
                                          // and an output clock from 45 - 80 MHz

    PLL0MUL = 0x02;                         // Multiply SYSCLK by M (M = 2)

    for (i=0; i < 256; i++) ;              // Wait at least 5us
    PLL0CN |= 0x02;                         // Enable the PLL
    while(!(PLL0CN & 0x10));                // Wait until PLL frequency is locked
    CLKSEL = 0x02;                         // Select PLL as SYSCLK source

    SFRPAGE = SFRPAGE_SAVE;                // Restore SFR page
}

```

```
//-----  
// PORT_Init  
//-----  
//  
// Configure the Crossbar and GPIO ports  
//  
void PORT_Init (void)  
{  
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page  
  
    SFRPAGE = CONFIG_PAGE;           // set SFR page  
  
    XBR0    = 0x00;  
    XBR1    = 0x00;  
    XBR2    = 0x44;                   // Enable crossbar, weak pull-ups,  
                                     // and UART1  
  
    POMDOUT |= 0x01;                 // Set TX1 pin to push-pull  
    P1MDOUT |= 0x40;                 // Set P1.6(LED) to push-pull  
  
    // all pins used by the external memory interface are in push-pull mode  
    P4MDOUT = 0xFF;  
    P5MDOUT = 0xFF;  
    P6MDOUT = 0xFF;  
    P7MDOUT = 0xFF;  
    P4 = 0xC0;                       // /WR, /RD, are high, RESET is low  
    P5 = 0x00;  
    P6 = 0x00;                       // P5, P6 contain the address lines  
    P7 = 0xFF;                       // P7 contains the data lines  
  
    SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page  
}  
  
//-----  
// EMIF_Init  
//-----  
//  
// Configure the External Memory Interface for Split-Mode to support both  
// on-chip and off-chip access.  
//  
void EMIF_Init (void){  
  
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page  
  
    SFRPAGE = LEGACY_PAGE;  
  
    EMI0CF = 0xF7;                   // Split-Mode, non-multiplexed  
                                     // on P4 - P7  
  
    EMI0TC = 0xB7;                   // This value may be modified  
                                     // according to SYSCLK to meet the  
                                     // timing requirements for the CS8900A  
                                     // For example, EMI0TC should be >= 0xB7  
                                     // for a 100 MHz SYSCLK.  
    SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page  
}
```

```

//-----
// UART1_Init
//-----
//
// Configure the UART1 using Timer1, for <baudrate> and 8-N-1.
//
void UART1_Init (void)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = UART1_PAGE;
    SCON1  = 0x10;                  // SCON1: mode 0, 8-bit UART, enable RX

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |= 0x20;                  // TMOD: timer 1, mode 2, 8-bit reload

    if (SYSCLK/BAUDRATE/2/256 < 1) {
        TH1 = -(SYSCLK/BAUDRATE/2);
        CKCON |= 0x10;              // T1M = 1; SCA1:0 = xx
    } else if (SYSCLK/BAUDRATE/2/256 < 4) {
        TH1 = -(SYSCLK/BAUDRATE/2/4);
        CKCON &= ~0x13;            // Clear all T1 related bits
        CKCON |= 0x01;             // T1M = 0; SCA1:0 = 01
    } else if (SYSCLK/BAUDRATE/2/256 < 12) {
        TH1 = -(SYSCLK/BAUDRATE/2/12);
        CKCON &= ~0x13;            // T1M = 0; SCA1:0 = 00
    } else {
        TH1 = -(SYSCLK/BAUDRATE/2/48);
        CKCON &= ~0x13;            // Clear all T1 related bits
        CKCON |= 0x02;             // T1M = 0; SCA1:0 = 10
    }

    TL1 = TH1;                      // initialize Timer1
    TR1 = 1;                        // start Timer1

    SFRPAGE = UART1_PAGE;
    TI1 = 1;                        // Indicate TX1 ready

    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}

//-----
// ipconfig
//-----
//
// Configure the IP address of the device through the serial port.
//
void ipconfig()
{
    char input_str[20];
    unsigned char data temp_char[6];
    char c;
    long i;
    bit ok_flag;
    bit EA_SAVE;                    // Preserve Current Interrupt Status
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR Page

```

```
// prompt for the IP and MAC addresses if this is the first time the program
// is run after a FLASH download or if the SW2(P1.6) button is pressed
if (first_time || !SW2 ) {

    SFRPAGE = UART1_PAGE;

    RI1 = 0;
    while (1) {

        printf("Press any key to continue\n");

        for ( i = 0; i < SYSCLK/100; i++) {
            if(RI1) break;
        }

        if (RI1) break;
        LED = ~LED;
    }
    RI1 = 0;

    ok_flag = 0;
    do {
        SFRPAGE = UART1_PAGE;
        printf("\n\nEnter the IP address (eg. 10.10.10.163) >");
        gets(input_str, sizeof(input_str));
        sscanf(input_str, "%bu.%bu.%bu.%bu", &temp_char[0], &temp_char[1],
            &temp_char[2], &temp_char[3]);

        // check if IP address is entered correctly and write in FLASH
        printf("\n\nIs %bu.%bu.%bu.%bu correct?", temp_char[0], temp_char[1],
            temp_char[2], temp_char[3]);

        c = getchar();
        if (c == 'y' || c == 'Y') {

            SFRPAGE = LEGACY_PAGE;

            // erase scratchpad area and write the ip address to FLASH
            EA_SAVE = EA;                // preserve current interrupt state
            EA = 0;                      // disable interrupts
            FLSCCL |= 0x01;              // enable FLASH write/erase
            PSCTL = 0x07;                // MOVX erases scratchpad FLASH

            *ptr_ip_address = 0;          // initiate erase

            PSCTL = 0x05;                // MOVX writes scratchpad FLASH

            ptr_ip_address[0] = temp_char[0]; // write the first byte
            ptr_ip_address[1] = temp_char[1]; // write the second byte
            ptr_ip_address[2] = temp_char[2]; // write the third byte
            ptr_ip_address[3] = temp_char[3]; // write the fourth byte

            PSCTL = 0x00;                // MOVX writes target XRAM
            FLSCCL &= ~0x01;             // disable FLASH write/erase
            EA = EA_SAVE;                // restore interrupts
            ok_flag = 1;

            SFRPAGE = UART1_PAGE;
```



```

        printf("\nIP address successfully programmed.\n");
    }
} while( !ok_flag);

ok_flag = 0;
do {

    SFRPAGE = UART1_PAGE;

    printf("\n\nEnter the MAC address (IA) (eg. 00-0B-3C-xx-yy-zz) >");
    gets(input_str, sizeof(input_str));
    sscanf(input_str, "%bX-%bX-%bX-%bX-%bX-%bX", &temp_char[0], &temp_char[1],
                                                    &temp_char[2], &temp_char[3],
                                                    &temp_char[4], &temp_char[5]);

    // check if IP address is entered correctly and write in FLASH
    printf("\n\nIs %bX-%bX-%bX-%bX-%bX-%bX correct?", temp_char[0], temp_char[1],
                                                    temp_char[2], temp_char[3],
                                                    temp_char[4], temp_char[5]);

    c = getchar();
    if(c == 'y' || c == 'Y'){

        SFRPAGE = LEGACY_PAGE;

        // write the MAC address to FLASH
        EA_SAVE = EA;                // preserve current interrupt state
        EA = 0;                      // disable interrupts
        FLSCCL |= 0x01;              // enable FLASH write/erase
        PSCTL = 0x05;               // MOVX writes scratchpad FLASH

        ptr_mac_address[0] = temp_char[0]; // write the first byte
        ptr_mac_address[1] = temp_char[1]; // write the second byte
        ptr_mac_address[2] = temp_char[2]; // write the third byte
        ptr_mac_address[3] = temp_char[3]; // write the fourth byte
        ptr_mac_address[4] = temp_char[4]; // write the fifth byte
        ptr_mac_address[5] = temp_char[5]; // write the sixth byte

        PSCTL = 0x01;               // MOVX writes FLASH byte

        *ptr_first_time = 0x00;      // clear the first_time flag
                                    // Note: this flag is not in
                                    // the scratchpad area.

        PSCTL = 0x00;               // MOVX writes target XRAM
        FLSCCL &= ~0x01;            // disable FLASH write/erase
        EA = EA_SAVE;               // restore interrupts

        ok_flag = 1;

        SFRPAGE = UART1_PAGE;
        printf("\nMAC address successfully programmed.\n");
    }
} while( !ok_flag);

}

// Disable Timer1
SFRPAGE = TIMER01_PAGE;

```

```
TR1 = 0; // Stop Timer1
TMOD = 0x00; // Restore the TMOD register to
// its reset value
CKCON = 0x00; // Restore the CKCON register to
// its reset value

// Disable UART1
SFRPAGE = UART1_PAGE;
SCON1 = 0x00; // Disable UART1

SFRPAGE = LEGACY_PAGE;

// Copy the IP and MAC address from the scratchpad area to the CMX variables
// located in RAM.
EA_SAVE = EA; // preserve current interrupt state
EA = 0; // disable interrupts
PSCTL = 0x04; // enable reads from the scratchpad

// read the IP and MAC address from FLASH into their appropriate arrays in memory
ip_src_addr[0] = ip_address[0];
ip_src_addr[1] = ip_address[1];
ip_src_addr[2] = ip_address[2];
ip_src_addr[3] = ip_address[3];
eth_src_hw_addr[0] = mac_address[0];
eth_src_hw_addr[1] = mac_address[1];
eth_src_hw_addr[2] = mac_address[2];
eth_src_hw_addr[3] = mac_address[3];
eth_src_hw_addr[4] = mac_address[4];
eth_src_hw_addr[5] = mac_address[5];

PSCTL = 0x00; // disable reads from the scratchpad
EA = EA_SAVE; // restore interrupts

SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
```

Appendix D—Example Ethernet Transmit and Receive Routines

```
//-----
// Ethernet_Routines.c
//-----
//
// AUTH: FB
// DATE: 7 OCT 02
//
// Target: C8051F12x
// Tool chain: KEIL C51
//
// Description: This is an example of how to send and receive packets using the
//              CS8900A Ethernet Controller in 8-bit polled mode.
//
//              This program periodically sends Ethernet Packets and captures
//              all incoming packets. The incoming packets are displayed on
//              a UART terminal at a baud rate of 115200.
//
//              To connect the device directly to a PC, a crossover Ethernet
//              cable is needed. If using a hub or a switch, then a normal
//              Ethernet cable may be used.
//
//-----
// Includes
//-----
#include <c8051f120.h>           // SFR declarations
#include <stdio.h>              // printf()

//-----
// 16-bit SFR Definitions for 'F12x
//-----

sfr16 DP      = 0x82;          // data pointer
sfr16 ADC0    = 0xbe;          // ADC0 data
sfr16 ADC0GT  = 0xc4;          // ADC0 greater than window
sfr16 ADC0LT  = 0xc6;          // ADC0 less than window
sfr16 RCAP2   = 0xca;          // Timer2 capture/reload
sfr16 RCAP3   = 0xca;          // Timer3 capture/reload
sfr16 RCAP4   = 0xca;          // Timer4 capture/reload
sfr16 TMR2    = 0xcc;          // Timer2
sfr16 TMR3    = 0xcc;          // Timer3
sfr16 TMR4    = 0xcc;          // Timer4
sfr16 DAC0    = 0xd2;          // DAC0 data
sfr16 DAC1    = 0xd2;          // DAC1 data
sfr16 PCA0CP5 = 0xe1;          // PCA0 Module 5 capture
sfr16 PCA0CP2 = 0xe9;          // PCA0 Module 2 capture
sfr16 PCA0CP3 = 0xeb;          // PCA0 Module 3 capture
sfr16 PCA0CP4 = 0xed;          // PCA0 Module 4 capture
sfr16 PCA0    = 0xf9;          // PCA0 counter
sfr16 PCA0CP0 = 0xfb;          // PCA0 Module 0 capture
sfr16 PCA0CP1 = 0xfd;          // PCA0 Module 1 capture

//-----
```

```
// Data Structures and Type Definitions
//-----
typedef union MACADDR {                // The 48-bit Ethernet MAC address
    unsigned int Int[3];
    unsigned char Char[6];
} MACADDR;

typedef union ULONG {                  // Byte Addressable Unsigned Long
    unsigned long Long;
    unsigned int Int[2];
    unsigned char Char[4];
} ULONG;

typedef union UINT {                  // Byte Addressable Unsigned Int
    unsigned int Int;
    unsigned char Char[2];
} UINT;

//-----
// Global CONSTANTS and VARIABLES
//-----
#define SYSCLK          49000000      // SYSCLK frequency in Hz
#define BAUDRATE        115200       // Baud rate of UART in bps

sbit ETH_RESET = P4^5;                // CS8900A reset pin

#define TRANSMIT_CMDH    0x00         // Transmit Command (High and Low
#define TRANSMIT_CMDL    0xC0         // bytes)

sbit LED = P1^6;                      // LED='1' means ON
sbit SW2 = P3^7;                      // SW2='0' means switch pressed

MACADDR MYMAC;                        // The 48-bit MAC address for
                                     // the Ethernet Controller
MACADDR BROADCAST;                   // A broadcast destination address
                                     // for sending packets

#define BASE_ADDRESS 0xC000

// CS8900A Internal PacketPage Register Addresses
#define IPPREG_PRODUCT_ID    0x0000
#define IPPREG_BASE_ADDRESS  0x0020
#define IPPREG_LineCTL       0x0112
#define IPPREG_RxCTL         0x0104
#define IPPREG_RxCFG         0x0102
#define IPPREG_BufCFG        0x010A
#define IPPREG_BufEvent      0x012C
#define IPPREG_TxEvent       0x0128
#define IPPREG_RxEvent       0x0124
#define IPPREG_IA            0x0158
#define IPPREG_BusST         0x0138
#define IPPREG_TestCTL       0x0118
#define IPPREG_LineST        0x0134
#define IPPREG_SelfST        0x0136

// CS8900A PacketPage Register Bit Definitions
#define TxBidErr    0x0080
#define RxOK        0x0100
#define Rdy4TxNow   0x0100
```

```

#define TxUnderrun    0x0200
#define TxOK          0x0100
#define INITD         0x0080
#define SerTxON       0x0080
#define SerRxON       0x0040
#define PromiscuousA  0x0080
#define RxOKA         0x0100
#define MulticastA    0x0200
#define IndividualA   0x0400
#define BroadcastA    0x0800
#define CRCErrorA     0x1000
#define RuntA         0x2000
#define ExtradataA    0x4000

//-----
// CS8900A Direct Access Register Definitions
//-----
volatile unsigned char xdata DATA0L      _at_ (BASE_ADDRESS + 0x0000);
volatile unsigned char xdata DATA0H      _at_ (BASE_ADDRESS + 0x0001);
volatile unsigned char xdata DATA1L      _at_ (BASE_ADDRESS + 0x0002);
volatile unsigned char xdata DATA1H      _at_ (BASE_ADDRESS + 0x0003);
volatile unsigned char xdata TxCMDL        _at_ (BASE_ADDRESS + 0x0004);
volatile unsigned char xdata TxCMDH        _at_ (BASE_ADDRESS + 0x0005);
volatile unsigned char xdata TxLENGTHL     _at_ (BASE_ADDRESS + 0x0006);
volatile unsigned char xdata TxLENGTHH     _at_ (BASE_ADDRESS + 0x0007);
volatile unsigned char xdata ISQL          _at_ (BASE_ADDRESS + 0x0008);
volatile unsigned char xdata ISQH          _at_ (BASE_ADDRESS + 0x0009);
volatile unsigned char xdata PACKETPAGE_POINTERL _at_ (BASE_ADDRESS + 0x000A);
volatile unsigned char xdata PACKETPAGE_POINTERH _at_ (BASE_ADDRESS + 0x000B);
volatile unsigned char xdata PACKETPAGE_DATA0L _at_ (BASE_ADDRESS + 0x000C);
volatile unsigned char xdata PACKETPAGE_DATA0H _at_ (BASE_ADDRESS + 0x000D);
volatile unsigned char xdata PACKETPAGE_DATA1L _at_ (BASE_ADDRESS + 0x000E);
volatile unsigned char xdata PACKETPAGE_DATA1H _at_ (BASE_ADDRESS + 0x000F);

//-----
// Function PROTOTYPES
//-----
void main (void);
void SYSCLK_Init (void);
void PORT_Init (void);
void UART1_Init (void);
void EMIF_Init (void);

void CS8900A_Reset(void);
void CS8900A_Init(void);

unsigned long PACKETPAGE_ReadID();
unsigned int PACKETPAGE_Read (unsigned int register_address);
void PACKETPAGE_Write(unsigned int register_address, unsigned int output_data);

void CS8900A_RxPoll(void);

void Receive_Frame(void);
void Send_Frame(char* buffer, int length, MACADDR* address);

//-----
// MAIN Routine
//-----
void main (void)

```

```
{

char buffer[28] = {
    0x00, 0x01, 0x08, 0x00, 0x06, 0x04, 0x00, 0x02, 0x01, 0x23,
    0x45, 0x67, 0x89, 0x10, 0x0A, 0x0A, 0x0A, 0xA3, 0x00, 0x80,
    0xAD, 0x81, 0x85, 0x0B, 0x0A, 0x0A, 0x0A, 0x9E
};

unsigned long id;                                // holds the device ID

char buffer2[5] = "    ";

long k;                                           // counter

WDTCN = 0xde;                                    // Disable watchdog timer
WDTCN = 0xad;

SYSCLK_Init ();
PORT_Init ();
UART1_Init ();
EMIF_Init ();

CS8900A_Reset();                                // Reset the CS8900A

CS8900A_Init();                                  // Initialize for Rx and Tx

// Initialize the Global MAC addresses
MYMAC.Int[0] = 0x0123;                           // This address should be
MYMAC.Int[1] = 0x4567;                           // set to the MAC address
MYMAC.Int[2] = 0x8910;                           // on the AB2 Ethernet Card

BROADCAST.Int[0] = 0xffff;
BROADCAST.Int[1] = 0xffff;
BROADCAST.Int[2] = 0xffff;

id = PACKETPAGE_ReadID();                        // Read the device ID

while(1){

    // check event registers for incoming packets
    for(k = 0; k < 50000; k++) { CS8900A_RxPoll(); }

    // send an IEEE 802.3 Frame
    Send_Frame(buffer, sizeof(buffer), &BROADCAST);

}

} // end main

//-----
// Init Routines
//-----

//-----
// SYSCLK_Init
//-----
```

```

//
// This routine initializes the system clock to use the internal oscillator
// at 24.5 MHz multiplied by two using the PLL.
//
void SYSCLK_Init (void)
{
    int i;                                // software timer

    char SFRPAGE_SAVE = SFRPAGE;         // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;               // set SFR page

    OSCICN = 0x83;                       // set internal oscillator to run
                                         // at its maximum frequency

    CLKSEL = 0x00;                       // Select the internal osc. as
                                         // the SYSCLK source

    //Turn on the PLL and increase the system clock by a factor of M/N = 2
    SFRPAGE = CONFIG_PAGE;

    PLL0CN = 0x00;                       // Set internal osc. as PLL source
    SFRPAGE = LEGACY_PAGE;
    FLSEL = 0x10;                       // Set FLASH read time for 50MHz clk
                                         // or less

    SFRPAGE = CONFIG_PAGE;
    PLL0CN |= 0x01;                      // Enable Power to PLL
    PLL0DIV = 0x01;                      // Set Pre-divide value to N (N = 1)
    PLL0FLT = 0x01;                      // Set the PLL filter register for
                                         // a reference clock from 19 - 30 MHz
                                         // and an output clock from 45 - 80 MHz

    PLL0MUL = 0x02;                      // Multiply SYSCLK by M (M = 2)

    for (i=0; i < 256; i++) ;            // Wait at least 5us
    PLL0CN |= 0x02;                      // Enable the PLL
    while(!(PLL0CN & 0x10));              // Wait until PLL frequency is locked
    CLKSEL = 0x02;                      // Select PLL as SYSCLK source

    SFRPAGE = SFRPAGE_SAVE;              // Restore SFR page
}

//-----
// PORT_Init
//-----
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
    char SFRPAGE_SAVE = SFRPAGE;         // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;               // set SFR page

    XBR0 = 0x00;
    XBR1 = 0x00;
    XBR2 = 0x44;                       // Enable crossbar, weak pull-ups,
                                         // and UART1

    POMDOUT |= 0x01;                    // Set TX1 pin to push-pull

```

```
P1MDOUT |= 0x40;    // Set P1.6(LED) to push-pull

// all pins used by the external memory interface are in push-pull mode
P4MDOUT = 0xFF;
P5MDOUT = 0xFF;
P6MDOUT = 0xFF;
P7MDOUT = 0xFF;
P4 = 0xC0;          // /WR, /RD, are high, RESET is low
P5 = 0x00;
P6 = 0x00;          // P5, P6 contain the address lines
P7 = 0xFF;          // P7 contains the data lines

SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page
}

//-----
// EMIF_Init
//-----
//
// Configure the External Memory Interface for both on and off-chip access.
//
void EMIF_Init (void){

    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = LEGACY_PAGE;

    EMI0CF = 0xF7;    // Split-mode, non-multiplexed
                     // on P4 - P7

    EMI0TC = 0xB7;    // This constant may be modified
                     // according to SYSCLK to meet the
                     // timing requirements for the CS8900A
                     // For example, EMI0TC should be >= 0xB7
                     // for a 100 MHz SYSCLK.

    SFRPAGE = SFRPAGE_SAVE;    // Restore SFR page
}

//-----
// UART1_Init
//-----
//
// Configure the UART1 using Timer1, for <baudrate> and 8-N-1.
//
void UART1_Init (void)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = UART1_PAGE;
    SCON1 = 0x10;    // SCON1: mode 0, 8-bit UART, enable RX

    SFRPAGE = TIMER01_PAGE;
    TMOD  &= ~0xF0;
    TMOD  |= 0x20;    // TMOD: timer 1, mode 2, 8-bit reload

    if (SYSCLK/BAUDRATE/2/256 < 1) {
        TH1 = -(SYSCLK/BAUDRATE/2);
    }
}
```



```

    CKCON |= 0x10;                // T1M = 1; SCA1:0 = xx
} else if (SYSCLK/BAUDRATE/2/256 < 4) {
    TH1 = -(SYSCLK/BAUDRATE/2/4);
    CKCON &= ~0x13;              // Clear all T1 related bits
    CKCON |= 0x01;               // T1M = 0; SCA1:0 = 01
} else if (SYSCLK/BAUDRATE/2/256 < 12) {
    TH1 = -(SYSCLK/BAUDRATE/2/12);
    CKCON &= ~0x13;              // T1M = 0; SCA1:0 = 00
} else {
    TH1 = -(SYSCLK/BAUDRATE/2/48);
    CKCON &= ~0x13;              // Clear all T1 related bits
    CKCON |= 0x02;               // T1M = 0; SCA1:0 = 10
}

TL1 = TH1;                      // initialize Timer1
TR1 = 1;                        // start Timer1

SFRPAGE = UART1_PAGE;
TI1 = 1;                        // Indicate TX1 ready

SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}

//-----
// CS8900A_Reset
//-----
//
// This procedure resets the CS8900A using its reset pin (P4.5).
//
void CS8900A_Reset(void)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = TMR3_PAGE;
    TMR3CN = 0x00;                  // Disable Timer3
    TMR3CF &= ~0x18;                // Count System Clocks/12
    TMR3 = -(SYSCLK / 2500000);      // Load Timer3 with 400ns

    // reset pin (active high) is on P4.5
    ETH_RESET = 1;                  // Assert the Reset signal
    TR3 = 1;                        // Start Timer3
    while(!TF3);                    // wait for the Timer3 overflow flag.
    ETH_RESET = 0;                  // Take the CS8900A out of reset

    // wait at least 20ms for device to be ready
    TMR3CN = 0x00;                  // Disable Timer3, use system clock/12
    TMR3 = -(SYSCLK / 12 / 50);     // Load Timer3 with 20ms
    TR3 = 1;                        // Start Timer3
    while(!TF3);                    // wait for the Timer3 overflow flag.

    // check to see if the device is ready
    while( !(PACKETPAGE_Read(IPPREG_SelfST) & INITD) );

    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}

```

```
//-----  
// CS8900A_Init  
//-----  
//  
// This function configures the CS8900A for transmitting and receiving. It  
// also assigns the CS8900A the MAC address it should respond to.  
//  
void CS8900A_Init(void)  
{  
    // set which frames will be accepted by the CS8900A  
    // the RxOKA bit must be set for the device to operate properly  
    PACKETPAGE_Write(IPPREG_RxCTL, RxOKA + PromiscuousA);  
  
    // assign the Ethernet MAC address  
    PACKETPAGE_Write(IPPREG_IA, MYMAC.Int[0]);  
    PACKETPAGE_Write(IPPREG_IA + 2, MYMAC.Int[1]);  
    PACKETPAGE_Write(IPPREG_IA + 4, MYMAC.Int[2]);  
  
    // enable transmit and receive  
    PACKETPAGE_Write(IPPREG_LineCTL, SerTxON | SerRxON);  
}  
  
//-----  
// Function Definitions  
//-----  
  
//-----  
// PACKETPAGE_Read  
//-----  
//  
// This function returns the contents of a PacketPage Register  
//  
// Note: Read high byte first and Write low byte first when communicating with  
//       the CS8900A.  
//  
unsigned int PACKETPAGE_Read (unsigned int IPPREG_address)  
{  
    UINT register_address;  
    UINT retval;  
  
    register_address.Int = IPPREG_address;  
  
    // specify register address and set autoincrement off  
    PACKETPAGE_POINTERL = register_address.Char[1];  
    PACKETPAGE_POINTERH = (register_address.Char[0] & 0x7F);  
  
    // read lower 16-bits most significant byte first  
    retval.Char[0] = PACKETPAGE_DATA0H;  
    retval.Char[1] = PACKETPAGE_DATA0L;  
  
    // return the data register contents  
    return retval.Int;  
}  
  
//-----  
// PACKETPAGE_Write
```

```

//-----
//
// This function writes a 16-bit value to a PacketPage Register.
//
// Note: Read high byte first and write low byte first when communicating with
//       the CS8900A
//
void PACKETPAGE_Write(unsigned int IPPREG_address, unsigned int output_data)
{
    UINT register_address;
    UINT dat;

    register_address.Int = IPPREG_address;
    dat.Int = output_data;

    // specify register address and set autoincrement off
    PACKETPAGE_POINTERL = register_address.Char[1];
    PACKETPAGE_POINTERH = (register_address.Char[0] & 0x7F);

    //write the data to the data ports
    PACKETPAGE_DATA0L = dat.Char[1];
    PACKETPAGE_DATA0H = dat.Char[0];
}

//-----
// PACKETPAGE_ReadID
//-----
//
// This function returns the contents of the Product Identification Code
// register. This is a 32-bit register at location 0 in the PacketPage memory.
//
// This register identifies the device as a CS8900A and does not change.
//
unsigned long PACKETPAGE_ReadID ()
{
    ULONG retval;

    retval.Int[0] = PACKETPAGE_Read(0x0000);
    retval.Int[1] = PACKETPAGE_Read(0x0002);

    return retval.Long;
}

//-----
// CS8900A_RxPoll
//-----
//
// This function polls the CS8900A for the Receive OK event.
//
void CS8900A_RxPoll(void)
{
    unsigned int event;

    event = PACKETPAGE_Read(IPPREG_RxEvent);

    if(event & RxOK){
        Receive_Frame();
        return;
    }
}

```

```
    }

}

//-----
// RECEIVE_FRAME
//-----
//
// This function Receives a frame from the CS8900A Data Ports and displays it
// on a Hyperterminal window.
//
void Receive_Frame(void)
{
    UINT status;
    UINT length;
    UINT dat;

    int i;

    status.Char[0] = DATA0H;
    status.Char[1] = DATA0L;

    length.Char[0] = DATA0H;
    length.Char[1] = DATA0L;

    SFRPAGE = UART1_PAGE;

    printf("\n\n New Packet: %d bytes", length.Int);

    printf("\n Destination: ");

    for ( i = 0; i < 3; i++) {

        dat.Char[0] = DATA0L;
        dat.Char[1] = DATA0H;

        printf("%04X", dat.Int);

        length.Int -= 2;
    }

    printf("\n Source:      ");

    for ( i = 0; i < 3; i++) {

        dat.Char[0] = DATA0L;
        dat.Char[1] = DATA0H;

        printf("%04X", dat.Int);

        length.Int -= 2;
    }

    printf("\n Data: ");

    while( ((int) length.Int) > 0 ){
```

```

    dat.Char[0] = DATA0L;
    dat.Char[1] = DATA0H;

    printf("%04X", dat.Int);

    length.Int -= 2;

}

}

//-----
// Send_Frame
//-----
//
// This function sends an IEEE 802.3 frame to the CS8900A. Upon entry, there
// should be valid data in array <buffer>.
//
//          48-bit   48-bit   16-bit   0-1500 bytes
// -----
// | Preamble | SFD | Dest | Source | Length of | Data Field | Pad | FCS |
// |          |    | Addr | Addr  | data field |          |    | (CRC) |
// -----
// supplied by |          supplied by the host (TxLength) | supplied by
// CS8900A     |          | CS8900A
void Send_Frame(char* buffer, int length, MACADDR* dest_address_ptr)
{
    UINT len;
    int status;
    int i;

    // issue a transmit command
    TxCMDL = TRANSMIT_CMDL;
    TxCMDH = TRANSMIT_CMDH;

    // bid for buffer space
    // data field length + dest field (6) + source field (6) + length field (2)
    len.Int = length + 14;
    TxLENGTHL = len.Char[1];
    TxLENGTHH = len.Char[0];

    // error check
    if (PACKETPAGE_Read(IPPREG_BusST) & TxBidErr) while (1);

    // wait for CS8900A Tx ready
    do {
        status = PACKETPAGE_Read(IPPREG_BusST);
    } while (!(status & Rdy4TxNow));

    // write the destination address field
    for (i = 0; i < 6; i+=2 ){
        DATA0L = dest_address_ptr->Char[i];
        DATA0H = dest_address_ptr->Char[i+1];
    }
}

```

```
// write the source address field
for (i = 0; i < 6; i+=2 ){

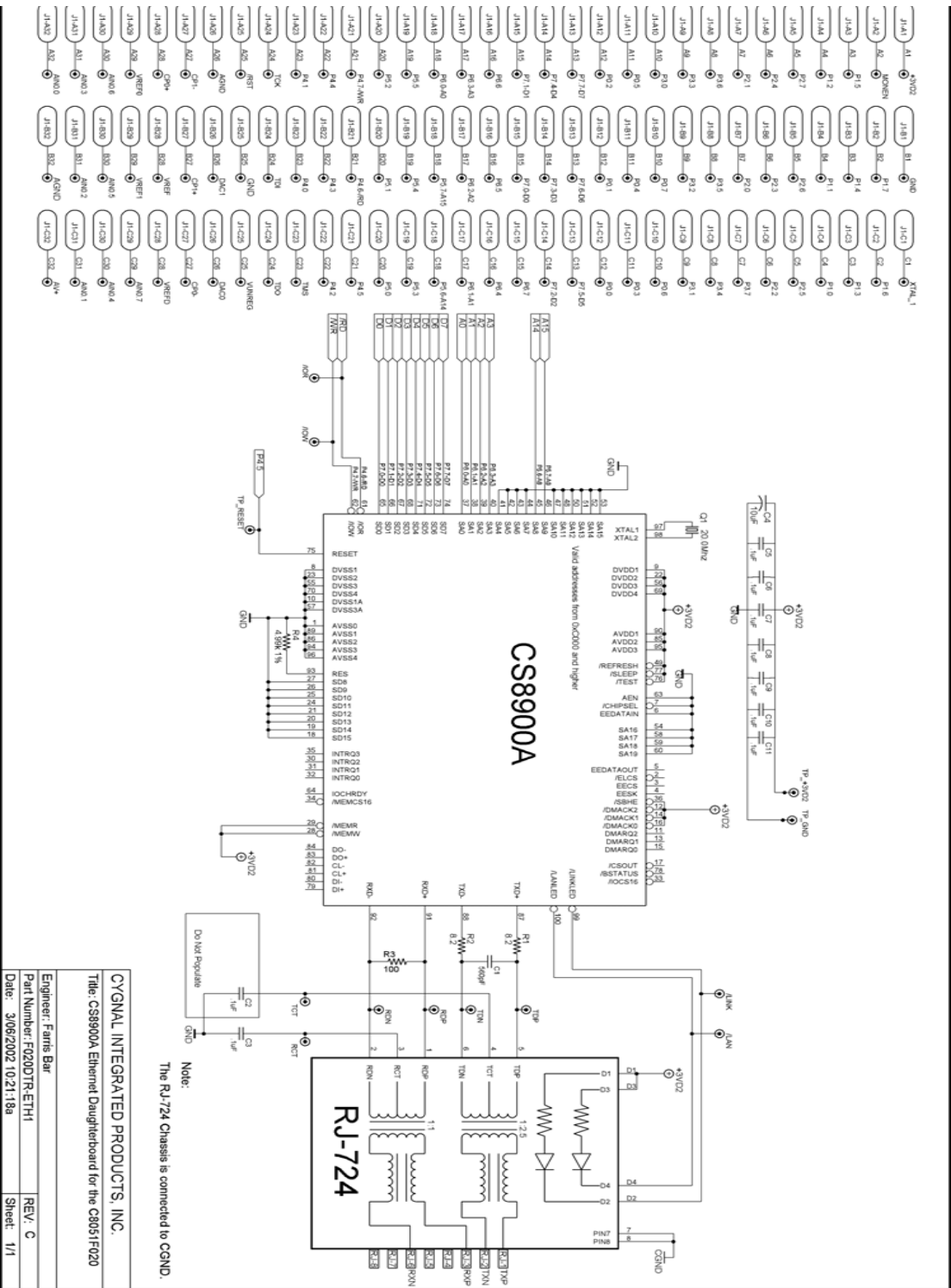
    DATA0L = MYMAC.Char[i];
    DATA0H = MYMAC.Char[i+1];
}

// write the data length field
len.Int = length;
DATA0L = len.Char[0];
DATA0H = len.Char[1];

// write the data field
// The CS8900A automatically transmits after the last byte is written
i = 0;
while (i < length){
    DATA0L = buffer[i];
    i++;

    if (i < length){
        DATA0H = buffer[i];
        i++;
    }
}
}
```

Appendix E—AB2 Ethernet Daughter Card Schematic



Appendix F—Bill of Materials for AB2 Ethernet Daughter Card

Qty	Part Number	Manufacturer	Description
1	RJ724 –L1	Trans-Power	RJ-45 Connector with integrated magnetics and LEDs. Contact: Trans-Power (www.trans-power.com) or Tyco Electronics (www.tycoelectronics.com) 1-800-468-2023 P/N 1-1605752-1
1	PCN10A-96P-2.54DS 650473-5	Hirose Electric OR AMP/TYCO Electronics	96-Pin DIN connector MALE Plug Right Angle, 3Row, Standard
1	CS8900A	Cirrus Logic	100 Pin TQFP
1			Testpoint
2			Rubber Feet

Qty	Value	Package	Notes
-----	-------	---------	-------

Capacitors

8	0.1 uF	0805	X7R 50/100V
1	10 uF	3216	Tant TE Series 6.3V Panasonic PCS1106CT-ND or eq.
1	560 pF	0805	C0G/Ceramic NPO 50V

Resistors

2	8.2 OHM	0805	MTFLM 5% 1/10W
1	100 OHM	0805	MTFLM 1% 1/10W
1	4.99K +/- 1% OHM	0805	MTFLM 1% 1/10W

Crystal

1	20.0 MHz		HC-49/VA 20PF Parallel
---	----------	--	------------------------

Notes:

Relevant Devices

This application note applies to the following devices:

C8051F120, C8051F121, C8051F122, C8051F123, C8051F124, C8051F125, C8051F126, and C8051F127.

Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032
Email: productinfo@silabs.com
Internet: www.silabs.com

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Silicon Laboratories:](#)

[AB2](#)