In the embedded networking course we worked on connecting an ethernet module (W5100) to the PSoC board. The goal was to to make the PSoC serve a web page through the module and have it do something with user input. My project consisted of a simple page with 2 buttons that switched a led on or off.

The software is quite crude yet mostly works. It was mostly written as I went instead of being designed from the ground up. There probably are some quirks in the code that works with the module's memory buffer. Sometimes the page is not served properly and instead there'll be some HTML syntax either missing or substituted with garbage characters. This likely indicates that the module's memory is being accessed in the wrong place at times. I didn't have time (read: couldn't be arsed) to investigate further since it worked most of the time.

The module is configured by sending and receiving instructions through the SPI interface. There were some timing issues with the SPI where it might've jammed if the module was polled too fast or something. Some delays scattered in the code are to alleviate that. I don't even know if they're really necessary.

First the W5100 module is initialized and configured to act as a server with its given ip address, subnet mask and port. Then the program flow is very simple: It waits until something writes to the port it's listening on. It then serves a web page with 2 submit buttons on it. When the page is served the program closes the connection and starts listening to a new connection again. After the initial connection if the user presses either of the buttons on the page, the browser opens a connection to the server again, this time transmitting a query string dictated by the submit button. The program reads the browser query and detecting a string 'q=' it parses the variable after the '=' sign either switching the led on or off depending on the input.

Using an ethernet module facing the internet could enable one to create all sorts of interesting contraptions. It could be used to send email warnings in conjunction with an alarm system. Another possible use would be a control interface to lights or pretty much any machinery.

```
//----------------------------------------------------------------
-------------
// C main line
//----------------------------------------------------------------
-------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User
Modules
#include <string.h>

char parseData(unsigned char data[], unsigned int size);
void receiveData(unsigned char data[], unsigned int size);
void writeHTTPHeader(void);
void writeHTMLPage(void);
void W5100_init(void);
unsigned int W5100_getStartAddress(void);
void W5100_openSocket(void);
void W5100_closeSocket(void);
void W5100_sendData(void);
void W5100_write(unsigned char addr_l, unsigned char addr_h,
unsigned char data);
unsigned char W5100_read(unsigned char addr_l, unsigned char
addr_h);
void W5100_writeString(char msg[], unsigned int size);
unsigned int W5100_read2(unsigned int addr);
void W5100_setPort(unsigned char port);
void W5100_write2(unsigned int addr, unsigned int data);
void spi_write(unsigned char b);
unsigned char spi_read(void);
void us_wait(unsigned char us);
void clearLCD(unsigned int mode);

//socket control codes
#define cSOCK_OPEN 0x01
#define cSOCK_CLOSE 0x10
#define cSOCK_LISTEN 0x02
#define cSOCK_SEND 0x20

//socket status codes
#define sSOCK_INIT 0x13
#define sSOCK_LISTEN 0x14
#define sSOCK_ESTABLISHED 0x17

char html0[] = "<HTML><FORM METHOD=GET ACTION=\"\">\n\r";
char html2[] = "<INPUT TYPE=HIDDEN VALUE=w NAME=\"q\">\n\r";
char html3[] = "<INPUT TYPE=SUBMIT>\n\r";
char html4[] = "</FORM>\n\r";
char html5[] = "<FORM METHOD=GET ACTION=\"\">\n\r";
char html6[] = "<INPUT TYPE=HIDDEN VALUE=s NAME=\"q\">\n\r";
char html7[] = "<INPUT TYPE=SUBMIT>\n\r";
char html8[] = "</FORM></HTML>\n\r";



void main(void)
{
	int i;
	char result = 0;
	unsigned int temp, getreq;
	unsigned char rData, dataSize;

	unsigned char data[100];
	unsigned char derp[2];
	CSelect_Start();
	CSelect_On();

	M8C_EnableGInt ;
	SleepTimer_Start();
	SleepTimer_SetInterval(SleepTimer_64_HZ);
	SleepTimer_EnableInt();
```

```
LCD_Start();
PWM8_WritePulseWidth(124);
PWM8_DisableInt();
PWM8_Stop();

SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);

W5100_init();

//currently waits for received data
//checks for 'q=w' or 'q=s' string inside the request
//submits html page. forcibly closes socket and starts over
while(1) {
     W5100_openSocket();

     //read socket status (S0_SR)
     rData = W5100_read(0x04, 0x03);

     if(rData == sSOCK_INIT) {
          LCD_Position(0,0);
          LCD_PrCString("Status: init");
     } else {
          W5100_closeSocket();
          LCD_Position(0,0);
          LCD_PrCString("Status: dead");
     }

     //socket control: listen
     W5100_write(0x04, 0x01, cSOCK_LISTEN);

     //if socket status = listen
     rData = W5100_read(0x04, 0x03);
     if(rData == sSOCK_LISTEN) {
          LCD_Position(1,0);
          LCD_PrCString("Status: listen");
     } else {
          W5100_closeSocket();
          LCD_Position(1,0);
          LCD_PrCString("Status: dead");
     }

     clearLCD(2);
     LCD_Position(0,0);

     if(result == 2) {
          LCD_PrCString("pwm stopped");
     } else if (result == 1) {
          LCD_PrCString("pwm running");
     }

     //wait until socket status = SOCK_ESTABLISHED
```

```
		do {
			rData = W5100_read(0x04, 0x03);
			clearLCD(1);
			LCD_Position(1,0);
			LCD_PrHexByte(rData);
			SleepTimer_SyncWait(8, SleepTimer_WAIT_RELOAD);
		} while(rData != sSOCK_ESTABLISHED);

		clearLCD(2);

		LCD_Position(0,0);
		LCD_PrCString("Status:");
		LCD_Position(1,0);
		LCD_PrCString("established");

		rData = 0;
		do {
			//read socket 0 received buffer size
			temp = W5100_read2(0x0426);
			SleepTimer_SyncWait(64, SleepTimer_WAIT_RELOAD);
		} while(temp == 0);

		clearLCD(1);
		receiveData(data, temp);

		result = parseData(data, 15);

		writeHTTPHeader();
		writeHTMLPage();

		W5100_closeSocket();

	}

	clearLCD(1);
	LCD_Position(1,0);
	LCD_PrCString("closed");

	while(1) {

	}

}

char parseData(unsigned char data[], unsigned int size) {
	int i, found;
	found = 0;
	for(i = 0; i < size; i++) {
		if(data[i] == 'q') {
			if(data[i+1] == '=') {
				found=1;
			}
```

```
            }
            if(found == 1) {
                  break;
            }
      }

      if(found == 1) {
            i+=2;
            if(data[i] == 'w') {
                  LCD_Position(0,0);
                  PWM8_Start();
                  return 1;
            } else if(data[i] == 's') {
                  LCD_Position(0,0);
                  PWM8_Stop();
                  return 2;
            }
      }
      return 0;
}

void receiveData(unsigned char data[], unsigned int size) {
      unsigned char rData;
      unsigned char test[2];
      int i, addr, offset, upperSize, leftSize;

      //get read pointer register S0_RX_RD
      addr = W5100_read2(0x0428);
      LCD_Position(1,0);
      LCD_PrHexInt(addr);
      //calculate offset S0_RX_RD & gS0_RX_MASK
      offset = (addr & 0x07FF);

      //start address =  gS0_RX_BASE + offset
      addr = 0x6000 + offset;

      if( (offset + size) > (0x6000 + 1) ) {
            upperSize = 0x6000 + 1 - offset;
            LCD_Position(1,5);
            LCD_PrHexInt(upperSize);
            for(i = 0; i < upperSize; i++) {
                  data[i] = W5100_read(addr >> 8, addr);
                  addr++;
            }

            leftSize = size - upperSize;
            LCD_Position(1,10);
            LCD_PrHexInt(leftSize);
            addr = 0x6000;
            for(i = upperSize; i < (upperSize + leftSize); i++) {
                  data[i] = W5100_read(addr >> 8, addr);
                  addr++;
            }

        } else {
            LCD_Position(1,5);
            LCD_PrHexInt(size);

            /*for(i = 0; i < size; i++) {
                data[i] = W5100_read(addr >> 8, addr);
                addr++;
            }*/

            test[1] = '\0';
            LCD_Position(0,0);
            for(i = 0; i < 15; i++) {
                data[i] = W5100_read(addr >> 8, addr);
                test[0] = data[i];
                addr++;
                LCD_PrString(test);
                SleepTimer_SyncWait(2, SleepTimer_WAIT_RELOAD);
            }
        }


        //S0_RX_RD += get_size
        addr = W5100_read2(0x0428);
        LCD_Position(1,10);
        LCD_PrHexInt(addr);
        addr += size;
        W5100_write2(0x0428, addr);

        //socket control: receive
        W5100_write(0x04, 0x01, 0x40);
    }


    void writeHTMLPage(void) {
        W5100_writeString(html0, 35);
        W5100_writeString(html2, 38);
        W5100_writeString(html3, 21);
        W5100_writeString(html4, 9);
        W5100_writeString(html5, 29);
        W5100_writeString(html6, 38);
        W5100_writeString(html7, 21);
        W5100_writeString(html8, 16);
    }

    void writeHTTPHeader(void) {
        char header1[] = "HTTP/1.1 200 OK\n\r";
        char header2[] = "Content-Type: text/html\n\r";
        char header3[] = "Content-Length: 207\n\r";
        char header4[] = "Connection: close\n\r";
        char header5[] = "\n\r";
```

```
     W5100_writeString(header1, 17);
     W5100_writeString(header2, 25);
     W5100_writeString(header3, 21);
     W5100_writeString(header4, 19);
     W5100_writeString(header5, 2);
}

void W5100_init(void) {
     int i;
     unsigned char initData[18] = {
                         0xC0, 0xA8, 0x01, 0x01, //gateway =
192.168.1.1
                         0xFF, 0xFF, 0xFF, 0x00, //subnet mask
= 255.255.255.0
                         0x00, 0x11, 0x22, 0x33, 0x44,
0x55, //mac = 00:11:22:33:44:55
                         0xC0, 0xA8, 0x01, 0x58}; //ip =
192.168.1.88

     unsigned char addrH[18] = {
                         0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00};

     unsigned char addrL[18] = {
                         0x01, 0x02, 0x03, 0x04,
                         0x05, 0x06, 0x07, 0x08,
                         0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E,
                         0x0F, 0x10, 0x11, 0x12};

     for(i = 0; i < 18; i++) {
          W5100_write( addrH[i], addrL[i], initData[i] );
     }

     //tcp mode (S0_MR = 0x01)
     W5100_write(0x04, 0x00, 0x01);

     //listen to port 80 (S0_PR = 0x50)
     W5100_setPort(0x50);
}

unsigned int W5100_read2(unsigned int addr) {
     int temp = W5100_read(addr >> 8, addr);
     temp = temp << 8;
     addr++;
     temp += W5100_read(addr >> 8, addr);

     return temp;
}
```

```
void W5100_write2(unsigned int addr, unsigned int data) {
     W5100_write(addr >> 8, addr, data >> 8);
     addr++;
     W5100_write(addr >> 8, addr, data);
}


void W5100_writeString(char msg[], unsigned int size) {
     int i, addr, offset,  leftSize, upperSize;
     //get available memory position for writing
     //start address S0_TX_WR
     addr = W5100_read2(0x0424);

     //get offset S0_TX_WR & gS0_TX_MASK (0x07FF)
     offset = (addr & 0x07FF);

     //start address = gS0_TX_BASE (0x4000) + offset
     addr = offset + 0x4000;

     //if overflow in tx buffer
     if( (offset + size) > (0x4000 + 1) ) {

          upperSize = 0x07FF + 1 - offset;

          for(i = 0; i < upperSize; i++) {
               W5100_write(addr >> 8, addr, msg[i]);
               addr++;
          }

          leftSize = size - upperSize;

          for(i = upperSize; i < (upperSize + leftSize); i++) {
               W5100_write(addr >> 8, addr, msg[i]);
               addr++;
          }

     } else { //if everything fits in the buffer continuously

          for(i = 0; i < size; i++) {
               W5100_write(addr >> 8, addr, msg[i]);
               addr++;
          }

     }



     //read address & increment by data size
     addr = W5100_read2(0x0424);
     addr += size;

     //write S0_TX_WR back
     W5100_write2(0x0424, addr);

     W5100_sendData();
}

void W5100_sendData(void) {
     W5100_write(0x04, 0x01, cSOCK_SEND);
}


void W5100_closeSocket(void) {
     W5100_write(0x04, 0x01, cSOCK_CLOSE);
}

void us_wait(unsigned char us) {
    do {
        asm("nop");
    } while (--us);
}

void spi_write(unsigned char b) {
     while(!(SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY));
     SPIM_SendTxData(b);
     while(!(SPIM_bReadStatus() & SPIM_SPIM_SPI_COMPLETE));
}

unsigned char spi_read(void) {
     unsigned char b = 0x00;
     b = SPIM_bReadRxData();
     return b;
}

void W5100_write(unsigned char addr_h, unsigned char addr_l,
unsigned char data) {
     CSelect_Off();
     spi_write(0xf0);
     spi_write(addr_h);
     spi_write(addr_l);
     spi_write(data);
     CSelect_On();
}

unsigned char W5100_read(unsigned char addr_h, unsigned char
addr_l) {
     unsigned char data;
     CSelect_Off();
     spi_write(0x0f);
     spi_write(addr_h);
     spi_write(addr_l);
     spi_write(0x00);
     CSelect_On();
     data = spi_read();
     return data;
}

void W5100_openSocket(void) {
     W5100_write(0x04, 0x01, cSOCK_OPEN);
}

void W5100_setPort(unsigned char port) {
     W5100_write2(0x0404, port);
}

void clearLCD(unsigned int mode) {
     switch(mode) {
          case 0:
               LCD_Position(0, 0);
               LCD_PrCString("                ");
               break;

          case 1:
               LCD_Position(1, 0);
               LCD_PrCString("                ");
               break;

          case 2:
               LCD_Position(0, 0);
               LCD_PrCString("                ");
               LCD_Position(1, 0);
               LCD_PrCString("                ");
               break;

          default:
               break;
     }
}
```