TÜRK MÜHENDİS VE MİMAR ODALARI BİRLİĞİ
ELEKTRİK MÜHENDİSLERİ ODASI
İSTANBUL ŞUBESİ
2005-2006 ÖĞRETİM YILI
PROJE YARIŞMASI

İNTERNET TABANLI EV OTOMASYONU İÇİN WEB SUNUCU TASARIMI

Ali Uğur EROL
Haluk ECE

Proje Yöneticisi: Yrd. Doç. Dr. Kayhan GÜLEZ

İstanbul, 2006

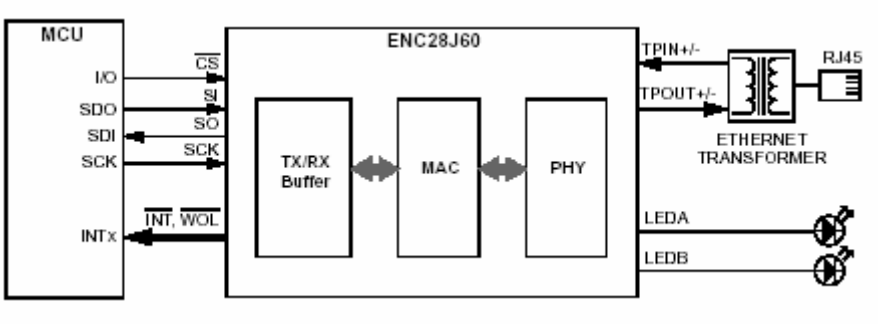
1. GİRİŞ

İnternet, birçok bilgisayar sisteminin birbirine bağlı olduğu, dünya çapında yaygın olan ve sürekli büyüyen bir iletişim ağıdır. Aynı zamanda, insanların her geçen gün gittikçe artan "üretilen bilgiyi saklama / paylaşma ve ona kolayca ulaşma" istekleri sonrasında ortaya çıkmış bir teknolojidir. Bu teknoloji yardımıyla pek çok alandaki bilgilere insanlar kolay, ucuz, hızlı ve güvenli bir şekilde erişebilmektedir.

Günümüz teknolojisinin geldiği noktada bina yönetim sistemleri, yalnızca büyük boyutlu binaları değil, her türlü yaşam alanını akıllı hale getirebilme özelliği sunuyor. Bilgi teknolojileri günlük yaşantımızın vazgeçilmez bir parçası olmuş ve özellikle son yıllarda internetin sağladığı olanaklar ile bilgiye ulaşmak ve bilgiyi paylaşmak oldukça kolaylaşmıştır. Bunun yanında teknolojinin bu denli değişmesi insanların günlük hayatlarında bazı değişikliklere neden olmuştur. Bununla birlikte ev otomasyon sistemleri de yavaş yavaş hayatımıza girmektedir. Güvenlik kameraları, otomatik kapılar, bahçe sulama sistemleri, sıcaklık ve ısı kontrolleri bu sistemlerin en basit örneklerini oluştururlar. Bu tür sistemlerin gelişmesi içim, uygulanabilirliğin artırılması ve maliyetinin düşürülmesi gerekiyor. Teknolojinin gelişmesi sürdükçe bunların da istenilen ve ulaşılabilen konumlara geleceği öngörülmektedir.

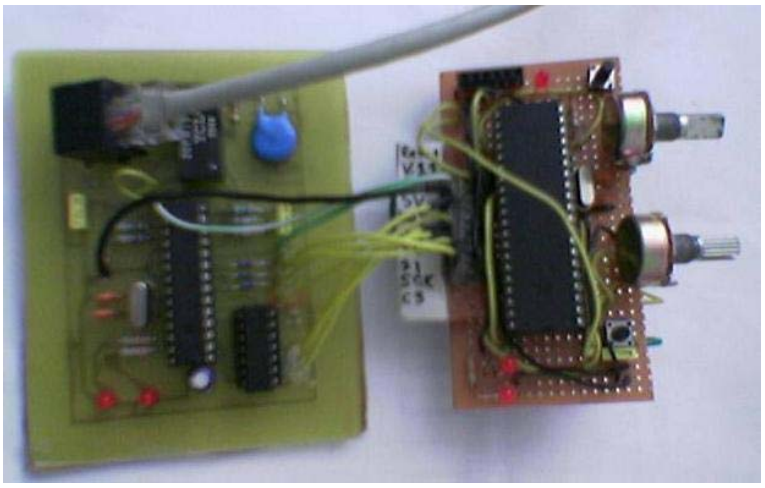
Yapılan çalışmada, gelişmiş ev kontrol sistemleri, internet teknolojileri kullanılarak düşük maliyetlerle tasarlanmış ve uygulanmıştır. Sistem kendi başına çalışan bir web sunucu ile bilgisayar gibi maliyet gerektiren donanımdan bağımsız çalışmakta olup, ev otomasyon sistemi dünyanın herhangi bir noktasından web üzerinden kontrol edilebilmektedir. İkinci bölümde kullanılan teknolojiler ve sistemin çalışma mantığı anlatılmaktadır.

2. SİSTEMİN ÇALIŞMA PRENSİBİ



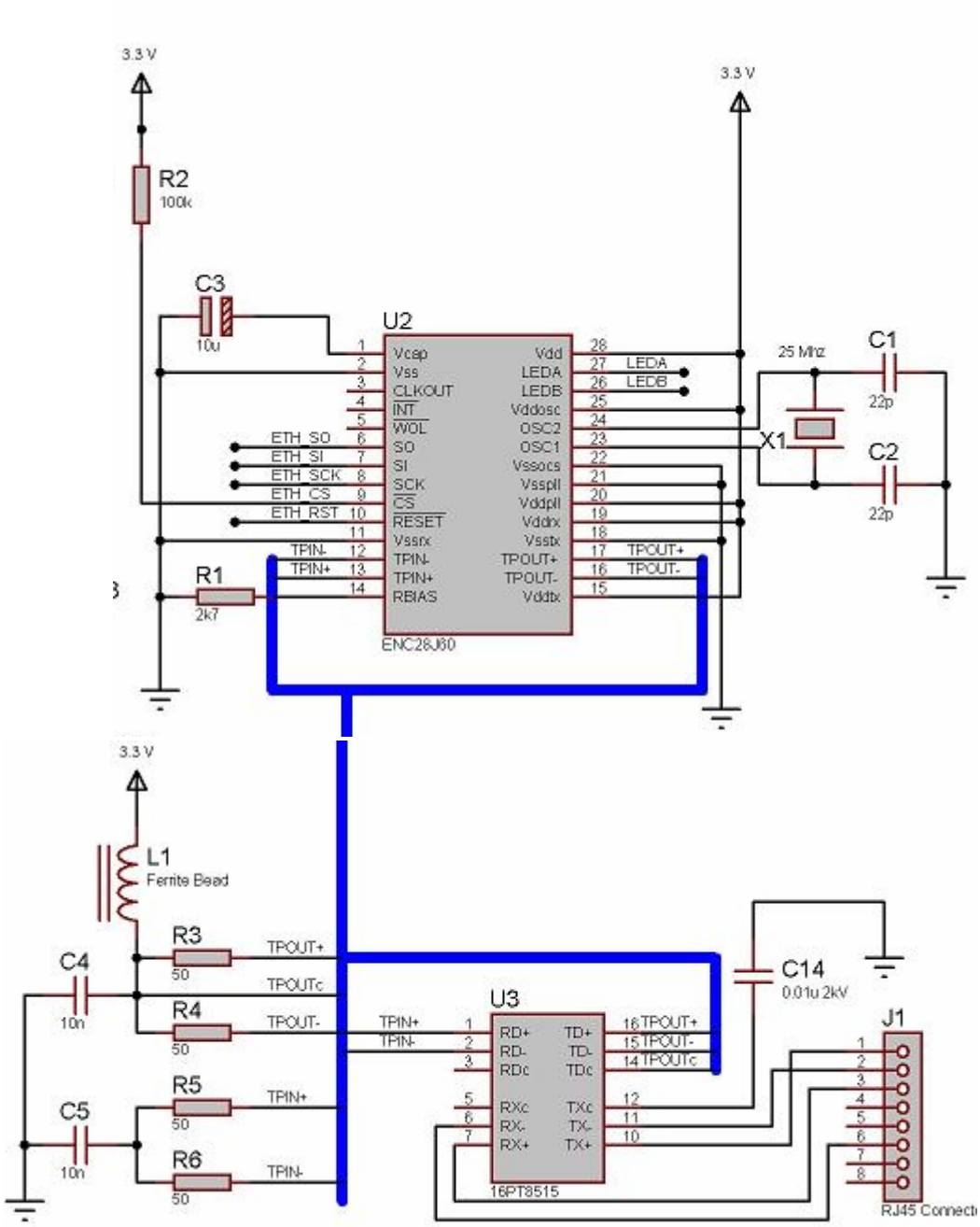
Şekil-2.1 Devrenin Prensip Şeması

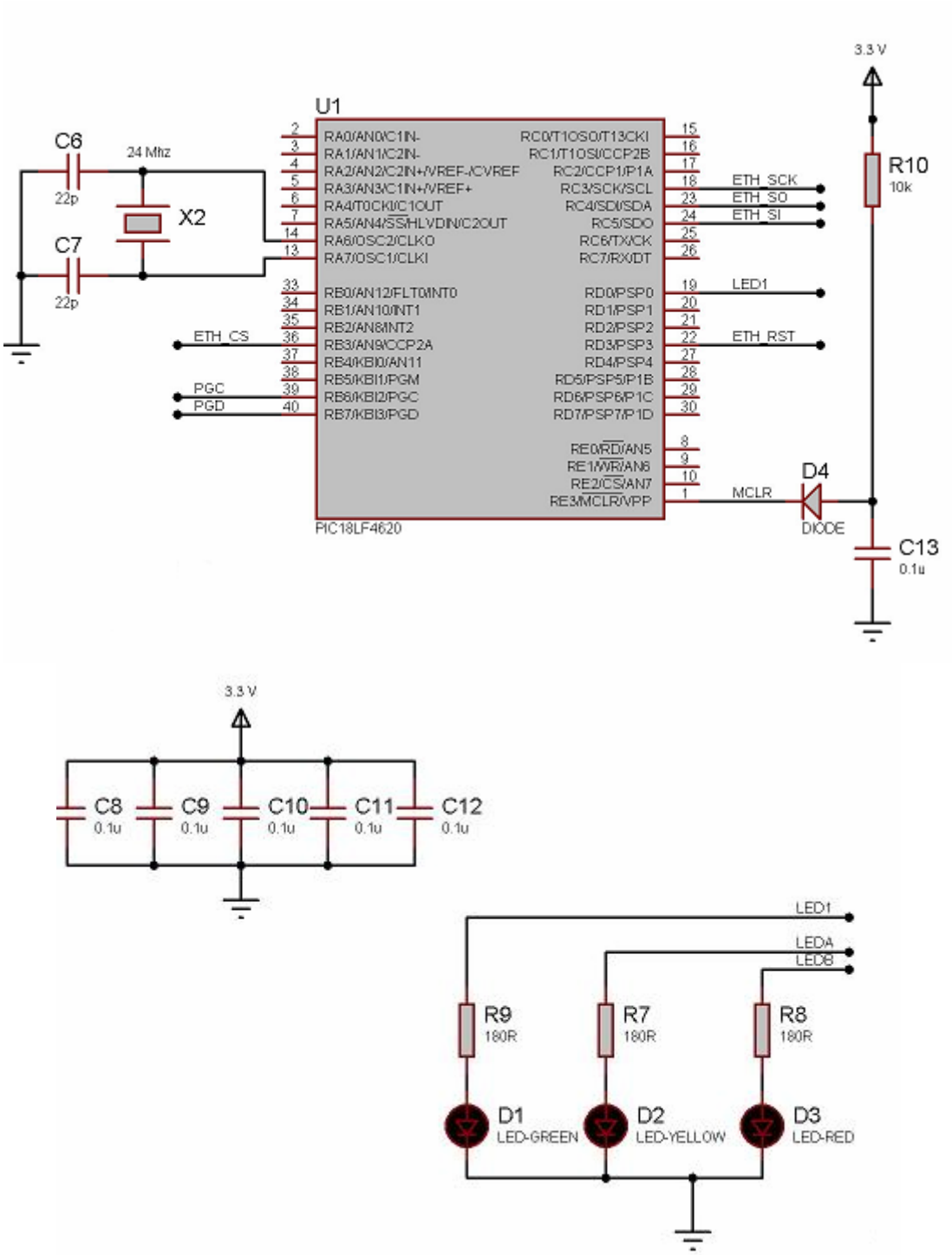
Devrede bulunan mikrodenetleyicinin I/O larından gelen bilgiler Ethernet kontrolcüsü ile internet ortamına aktartılmaktadır. Mikrokontrolcü olarak Pic 18f4620 seçilmiştir. Bu kontrolcü 8-bit data yazmaçlarına sahip, 13 adet analog girişi ve bir çok dijital giriş çıkışa sahip olması göz önünde bulundurularak seçilmiştir. Ayrıca bu mikrodenetleyicinin 64 Kilobyte program hafızasına sahip olması içinde gömülü bulunan web sunusunun rahatça kullanılmasına olanak sağlamaktadır. Ev otomasyonunun internet bağlantısı için tasarlanan devrede çeşitli cihazlar uzaktan bir web browser üzerinden sistemin web sunucusuna bağlanarak kontrol edilebilmektedir. Devrenin prensip şeması şekil-2.1 de gösterildiği üzere basit bir yapıya sahiptir. pic18f4620 mikrokontrolcüsüne bağlı Ethernet kontrolcüsü ve kontrolcüye bağlanan birkaç eleman üzerinden ethernet bağlantısı sağlanmaktadır. Devrede Microchip ürünü ENC28J60 Ethernet kontrolcüsü kullanılmıştır. 28 bacaklı olan bu entegre kullanımı kolay, herhangi bir mikrokontrolcü ile seri arayüzle haberleşebilir olmakla birlikte bu sistemin temel elemanı olmaktadır.



Şekil 2.2 Devrenin Geliştirme Aşamasındaki Görünümü

Devre Şeması:





Şekil 2.3 Devre Şeması

Yazılım:

Devrede kullanılan mikrodenetleyicisinin yazılımı C dilinde yazılmış olup Microchip firmasının PIC18f4620 entegresi kullanılmıştır. Yazılım geliştirme aracı olarak MPLAB programı kullanıldık. Derleyici olarak MCC18 kullanıldı. Derlenen programın bilgisayar ortamından mikrodenetleyicisinin program hafızasına aktarılma işlemi PIC START PLUS programlayıcısı ile gerçekleşmiştir.

Yazılımın temelleri Microchip firmasının ENC28J60 entegresinin sistem tasarımcılarının geliştirmesi için yayınlamış olduğu kaynak kodlara dayanmaktadır.

Kaynak Kodları:

```
// Buradan
#define THIS_IS_STACK_APPLICATION


#include <string.h>

/*
 * These headers must be included for required defs.
 */
#include "StackTsk.h"
#include "Tick.h"
#include "MAC.h"
#include "Helpers.h"

#if defined(STACK_USE_DHCP)
#include "DHCP.h"
#endif

#if defined(STACK_USE_HTTP_SERVER)
#include "HTTP.h"
#endif

#include "MPFS.h"

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
#include "FTP.h"
#endif

#if defined(USE_LCD)
#include "XLCD.h"
#endif

#if defined(STACK_USE_ANNOUNCE)
#include "Announce.h"
#endif

#if defined(MPFS_USE_EEPROM)
#include "XEEPROM.h"
#endif


#include "Delay.h"

#define STARTUP_MSG "MCHPStack 3.02"

ROM char const StartupMsg[] = STARTUP_MSG;

#if (defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)) &&
defined(USE_LCD)
ROM char const DHCPMsg[] = "DHCP/Gleaning...";
```

```
#endif

ROM char const SetupMsg[] = "Board Setup...";


APP_CONFIG AppConfig;

BYTE myDHCPBindCount = 0;
#if defined(STACK_USE_DHCP)
    extern BYTE DHCPBindCount;
#else
    /*
     * If DHCP is not enabled, force DHCP update.
     */
    BYTE DHCPBindCount = 1;
#endif

/*
 * Set configuration fuses
 */
#if defined(MCHP_C18) && defined(__18F4620)
#pragma config OSC=HSPLL, FCMEN=OFF, IESO=OFF
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config LVP=OFF
#elif defined(HITECH_C18)
__CONFIG(1, UNPROTECT & 0x36FF);    // Fail-safe clock monitor disable,
oscillator switch over disabled, HS_PLL
__CONFIG(2, PWRTDIS & BORDIS & WDTDIS);
#endif

/*
 * Private helper functions.
 * These may or may not be present in all applications.
 */
static void InitAppConfig(void);
static void InitializeBoard(void);
static void ProcessIO(void);

BOOL StringToIPAddress(char *str, IP_ADDR *buffer);
void NotifyRemoteUser(void);
void USARTPutROMString(ROM char const *str);
static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD);
//--------->static void SetConfig(void);

#if defined(MPFS_USE_EEPROM)
static BOOL DownloadMPFS(void);
static void SaveAppConfig(void);
#else
      #define SaveAppConfig()
#endif

// NOTE: Several PICs, including the PIC18F4620 revision A3 have a RETFIE
FAST/MOVFF bug
// The interruptlow keyword is used to work around the bug when using C18
#if defined(MCHP_C18)
    #pragma interruptlow HighISR //save=section(".tmpdata")
    void HighISR(void)
#elif defined(HITECH_C18)
    #if defined(STACK_USE_SLIP)
        extern void MACISR(void);
    #endif
    void interrupt HighISR(void)
#endif
{
    TickUpdate();

#if defined(STACK_USE_SLIP)
    MACISR();
#endif
```

```
}

#if defined(MCHP_C18)
#pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code /* return to default code section */
#endif



/*
*  Main
*/
void main(void)
{
    static TICK t = 0;

    /*
     * Initialize any application specific hardware.
     */

      InitializeBoard();

      /*
     * Initialize all stack related components.
     * Following steps must be performed for all applications using
     * PICmicro TCP/IP Stack.
     */
    TickInit();

    /*
     * Following steps must be performed for all applications using
     * PICmicro TCP/IP Stack.
     */
    MPFSInit();

    /*
     * Initialize Stack and application related NV variables.
     */
    InitAppConfig();

    /*
     * This implementation, initiates Board setup process if RB0
     * is detected low on startup.
     */


    StackInit();


#if defined(STACK_USE_HTTP_SERVER)
    HTTPInit();
#endif

#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
    FTPInit();
#endif


#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
    if ( AppConfig.Flags.bIsDHCPEnabled )
    {
#if defined(USE_LCD)
        XLCDGoto(1, 0);
        XLCDPutROMString(DHCPMsg);
#endif
    }
```

```
    else
    {
        /*
         * Force IP address display update.
         */
        myDHCPBindCount = 1;
#if defined(STACK_USE_DHCP)
        DHCPDisable();
#endif
    }
#endif

    while(1)
    {
        /*
         * Blink SYSTEM LED every second.
         */
        if ( TickGetDiff(TickGet(), t) >= TICK_SECOND/2 )
        {
            t = TickGet();
            LATD0 ^= 1;
        }

        /*
         * This task performs normal stack task including checking
         * for incoming packet, type of packet and calling
         * appropriate stack entity to process it.
         */
        StackTask();


#if defined(STACK_USE_HTTP_SERVER)
        /*
         * This is a TCP application.  It listens to TCP port 80
         * with one or more sockets and responds to remote requests.
         */
        HTTPServer();
#endif


#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
        FTPServer();
#endif

        /*
         * In future, as new TCP/IP applications are written, it
         * will be added here as new tasks.
         */

         /*
          * Add your application speicifc tasks here.
          */
        ProcessIO();    //Sorun çıkarabilirrrrrrrrrrrrrrrrrr


        /*
         * For DHCP information, display how many times we have renewed the
IP
         * configuration since last reset.
         */
        if ( DHCPBindCount != myDHCPBindCount )
        {
            myDHCPBindCount = DHCPBindCount;

                  USARTPutROMString(NewIP);
            DisplayIPValue(&AppConfig.MyIPAddr, FALSE);    // Print to
USART
                  USARTPutROMString(CRLF);
#if defined(STACK_USE_ANNOUNCE)
                  AnnounceIP();
```

```
#endif

#if defined(USE_LCD)
            DisplayIPValue(&AppConfig.MyIPAddr, TRUE);

            if ( AppConfig.Flags.bIsDHCPEnabled )
            {
                XLCDGoto(1, 14);
                if ( myDHCPBindCount < 0x0a )
                    XLCDPut(myDHCPBindCount + '0');
                else
                    XLCDPut(myDHCPBindCount + 'A');
            }
#endif
        }

    }
}

#if defined(USE_LCD)
//                                1234567890123456
ROM char const blankLCDLine[] = "                ";
#endif


static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD)
{
    char IPDigit[8];

#ifdef USE_LCD
    if ( bToLCD )
    {
        /*
         * Erase second line.
         */
        XLCDGoto(1, 0);
        XLCDPutROMString(blankLCDLine);

    }

    /*
     * Rewrite the second line.
     */
    XLCDGoto(1, 0);
#endif

    itoa(IPVal->v[0], IPDigit);
#ifdef USE_LCD
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
#endif
    {
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[1], IPDigit);
#ifdef USE_LCD
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
#endif
    {
```

```
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[2], IPDigit);
#ifdef USE_LCD
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
#endif
    {
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[3], IPDigit);
#ifdef USE_LCD
    if ( bToLCD )
        XLCDPutString(IPDigit);
    else
#endif
        USARTPutString((BYTE*)IPDigit);
}


static char AN0String[8];
static char AN1String[8];

static void ProcessIO(void)
{
    WORD_VAL ADCResult;

    /*
     * Select AN0 channel, Fosc/64 clock
     * Works for both compatible and regular A/D modules
     */
      ADCON0      = 0b10000001;

    /*
     * Wait for acquisition time.
     * Here, rather than waiting for exact time, a simple wait is
     * used.  Real applications requiring high accuracy should
     * calculate exact acquisition time and wait accordingly.
     */
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );

    /*
     * First convert AN0 channel.
     * AN0 is already setup as an analog input.
     */
    ADCON0_GO   = 1;

    /*
     * Wait until conversion is done.
     */
    while( ADCON0_GO );

    /*
     * Save the result.
     */
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    /*
     * Convert 10-bit value into ASCII String.
     */
```

```
    itoa(ADCResult.Val, AN0String);

    /*
     * Now, convert AN1 channel.
     *
     * In PICDEM.net board, RA2 thru RA7 should be digital or else
     * LED, LCD and NIC would not operate correctly.
     * Since there is no mode where only AN0 and AN1 be analog inputs
     * while rests are digial pins, we will temperoraily switch
     * select a mode where RA2 becomes analog input while we do
     * conversion of RA1.  Once conversion is done, we will convert
     * RA2 back to digital pin.
     */
#if defined(USE_COMPATIBLE_AD)
      // Change AN1 to be an analog input
    ADCON1      = 0b11001101;  //değişti

    // Select AN1 channel.
    ADCON0      = 0b10001001;
#else
    // Select AN1 channel.
      ADCON0            = 0b00000101;
#endif


    /*
     * Wait for acquisition time.
     * Here, rather than waiting for exact time, a simple wait is
     * used.  Real applications requiring high accuracy should
     * calculate exact acquisition time and wait accordingly.
     */
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );

    /*
     * Start the conversion.
     */
    ADCON0_GO   = 1;

    /*
     * Wait until it is done.
     */
    while( ADCON0_GO );

    /*
     * Save the result.
     */
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;

    /*
     * Convert 10-bit value into ASCII String.
     */
    itoa(ADCResult.Val, AN1String);

    /*
     * Reset RA2 pin back to digital output.
     */
#if defined(USE_COMPATIBLE_AD)
    ADCON1      = 0b11001110;       // RA0 as analog input.
#endif
}

/*
 * CGI Command Codes.
 */
#define CGI_CMD_DIGOUT      (0)
#define CGI_CMD_LCDOUT      (1)          // Obsolete.  No LCD present.
#define CGI_CMD_RECONFIG     (2)
```

```
/*
 * CGI Variable codes. - There could be 00h-FFh variables.
 * NOTE: When specifying variables in your dynamic pages (.cgi),
 *       use the hexadecimal numbering scheme and always zero pad it
 *       to be exactly two characters.  Eg: "%04", "%2C"; not "%4" or
"%02C"
 */
#define VAR_LED_D5          (0x00)
#define VAR_LED_D6          (0x01)
#define VAR_ANAIN_AN0       (0x02)
#define VAR_ANAIN_AN1       (0x03)
#define VAR_DIGIN_RB5       (0x04)
#define VAR_STROUT_LCD      (0x05) // Obsolete.  No LCD present.
#define VAR_MAC_ADDRESS     (0x06)
#define VAR_SERIAL_NUMBER   (0x07)
#define VAR_IP_ADDRESS      (0x08)
#define VAR_SUBNET_MASK     (0x09)
#define VAR_GATEWAY_ADDRESS (0x0A)
#define VAR_DHCP          (0x0B)    // Use this variable when the web page
is updating us
#define VAR_DHCP_TRUE       (0x0B) // Use this variable when we are
generating the web page
#define VAR_DHCP_FALSE      (0x0C) // Use this variable when we are
generating the web page


/*****************************************************************
 * Function:        void HTTPExecCmd(BYTE** argv, BYTE argc)
 *
 * PreCondition:    None
 *
 * Input:           argv        - List of arguments
 *                  argc        - Argument count.
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        This function is a "callback" from HTTPServer
 *                  task.  Whenever a remote node performs
 *                  interactive task on page that was served,
 *                  HTTPServer calls this functions with action
 *                  arguments info.
 *                  Main application should interpret this argument
 *                  and act accordingly.
 *
 *                  Following is the format of argv:
 *                  If HTTP action was : thank.htm?name=Joe&age=25
 *                      argv[0] => thank.htm
 *                      argv[1] => name
 *                      argv[2] => Joe
 *                      argv[3] => age
 *                      argv[4] => 25
 *
 *                  Use argv[0] as a command identifier and rests
 *                  of the items as command arguments.
 *
 * Note:            THIS IS AN EXAMPLE CALLBACK.
 *****************************************************************/
#if defined(STACK_USE_HTTP_SERVER)

ROM char const COMMANDS_OK_PAGE[] = "COMMANDS.CGI";
ROM char const CONFIG_UPDATE_PAGE[] = "CONFIG.CGI";

// Copy string with NULL termination.
#define COMMANDS_OK_PAGE_LEN  (sizeof(COMMANDS_OK_PAGE))
#define CONFIG_UPDATE_PAGE_LEN  (sizeof(CONFIG_UPDATE_PAGE))

ROM char const CMD_UNKNOWN_PAGE[] = "INDEX.HTM";
```

```
// Copy string with NULL termination.
#define CMD_UNKNOWN_PAGE_LEN    (sizeof(CMD_UNKNOWN_PAGE))

void HTTPExecCmd(BYTE** argv, BYTE argc)
{
    BYTE command;
    BYTE var;
    BYTE CurrentArg;

    WORD_VAL TmpWord;

    /*
     * Design your pages such that they contain command code
     * as a one character numerical value.
     * Being a one character numerical value greatly simplifies
     * the job.
     */
    command = argv[0][0] - '0';

    /*
     * Find out the cgi file name and interpret parameters
     * accordingly
     */
    switch(command)
    {
    case CGI_CMD_DIGOUT:      // ACTION=0
        /*
         * This DIGOUTS.CGI.  Any arguments with this file
         * must be about controlling digital outputs.
         */

        /*
         * Identify the parameters.
         * Compare it in upper case format.
         */
        var = argv[1][0] - '0';

        switch(var)
        {
        case VAR_LED_D5:      // NAME=0
            /*
             * This is "D5".
             * Toggle D5.
             */
            LATD5 ^= 1;
            break;

        case VAR_LED_D6:      // NAME=1
            /*
             * This is "D6".
             * Toggle it.
             */
            LATD6 ^= 1;
            break;
         }

         memcpypgm2ram((void*)argv[0],
              (ROM void*)COMMANDS_OK_PAGE, COMMANDS_OK_PAGE_LEN);
         break;
//    case CGI_CMD_LCDOUT:    // ACTION=1
//        /*
//         * Not implemented.
//         */
//        break;

      case CGI_CMD_RECONFIG:  // ACTION=2
            // Loop through all variables that we've been given
            CurrentArg = 1;
            while(argc > CurrentArg)
            {
```

```
                // Get the variable identifier (HTML "name"), and
                // increment to the variable's value
                TmpWord.byte.MSB = argv[CurrentArg][0];
                TmpWord.byte.LSB = argv[CurrentArg++][1];
            var = hexatob(TmpWord);

            // Make sure the variable's value exists
            if(CurrentArg >= argc)
                break;

            // Take action with this variable/value
            switch(var)
            {
            case VAR_SERIAL_NUMBER:
                AppConfig.SerialNumber.Val = atoi(argv[CurrentArg]);
                AppConfig.MyMACAddr.v[4] =
AppConfig.SerialNumber.byte.MSB;
                AppConfig.MyMACAddr.v[5] =
AppConfig.SerialNumber.byte.LSB;
                break;

            case VAR_IP_ADDRESS:
            case VAR_SUBNET_MASK:
            case VAR_GATEWAY_ADDRESS:
                {
                      DWORD TmpAddr;

                      // Convert the returned value to the 4 octect
                      // binary representation
                        if(!StringToIPAddress(argv[CurrentArg],
(IP_ADDR*)&TmpAddr))
                            break;

                            // Reconfigure the App to use the new values
                        if(var == VAR_IP_ADDRESS)
                        {
                              // Cause the IP address to be rebroadcast
                              // through Announce.c or the RS232 port
since
                              // we now have a new IP address
                              if(TmpAddr != *(DWORD*)&AppConfig.MyIPAddr)
                                    DHCPBindCount++;

                                // Set the new address
                            memcpy((void*)&AppConfig.MyIPAddr,
(void*)&TmpAddr, sizeof(AppConfig.MyIPAddr));
                        }
                        else if(var == VAR_SUBNET_MASK)
                            memcpy((void*)&AppConfig.MyMask,
(void*)&TmpAddr, sizeof(AppConfig.MyMask));
                        else if(var == VAR_SUBNET_MASK)
                            memcpy((void*)&AppConfig.MyGateway,
(void*)&TmpAddr, sizeof(AppConfig.MyGateway));
                    }
                break;

            case VAR_DHCP:
                if(AppConfig.Flags.bIsDHCPEnabled)
                {
                      if(!(argv[CurrentArg][0]-'0'))
                      {
                            AppConfig.Flags.bIsDHCPEnabled = FALSE;
                      }
                  }
                  else
                {
                      if(argv[CurrentArg][0]-'0')
                      {
                       MY_IP_BYTE1 = 0;
                       MY_IP_BYTE2 = 0;
```

```
                        MY_IP_BYTE3 = 0;
                        MY_IP_BYTE4 = 0;

                              AppConfig.Flags.bIsDHCPEnabled = TRUE;
//                              stackFlags.bits.bInConfigMode = TRUE;
                              //DHCPReset();
                            }
                        }
                    break;
            }

                    // Advance to the next variable (if present)
                    CurrentArg++;
        }

            // Save any changes to non-volatile memory
            SaveAppConfig();


            // Return the same CONFIG.CGI file as a result.
        memcpypgm2ram((void*)argv[0],
             (ROM void*)CONFIG_UPDATE_PAGE, CONFIG_UPDATE_PAGE_LEN);
            break;

    default:
        memcpypgm2ram((void*)argv[0],
              (ROM void*)CMD_UNKNOWN_PAGE, CMD_UNKNOWN_PAGE_LEN);
        break;
    }

}
#endif


/*****************************************************************************
 * Function:        WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
 *
 * PreCondition:    None
 *
 * Input:           var         - Variable Identifier
 *                  ref         - Current callback reference with
 *                                respect to 'var' variable.
 *                  val         - Buffer for value storage.
 *
 * Output:          Variable reference as required by application.
 *
 * Side Effects:    None
 *
 * Overview:        This is a callback function from HTTPServer() to
 *                  main application.
 *                  Whenever a variable substitution is required
 *                  on any html pages, HTTPServer calls this function
 *                  8-bit variable identifier, variable reference,
 *                  which indicates whether this is a first call or
 *                  not.  Application should return one character
 *                  at a time as a variable value.
 *
 * Note:            Since this function only allows one character
 *                  to be returned at a time as part of variable
 *                  value, HTTPServer() calls this function
 *                  multiple times until main application indicates
 *                  that there is no more value left for this
 *                  variable.
 *                  On begining, HTTPGetVar() is called with
 *                  ref = HTTP_START_OF_VAR to indicate that
 *                  this is a first call.  Application should
 *                  use this reference to start the variable value
 *                  extraction and return updated reference.  If
 *                  there is no more values left for this variable
 *                  application should send HTTP_END_OF_VAR.  If
```

```
 *                  there are any bytes to send, application should
 *                  return other than HTTP_START_OF_VAR and
 *                  HTTP_END_OF_VAR reference.
 *
 *                  THIS IS AN EXAMPLE CALLBACK.
 *                  MODIFY THIS AS PER YOUR REQUIREMENTS.
 *****************************************************************/
#if defined(STACK_USE_HTTP_SERVER)
WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
{
      // Temporary variables designated for storage of a whole return
      // result to simplify logic needed since one byte must be returned
      // at a time.
      static BYTE VarString[20];
      static BYTE VarStringLen;
      BYTE *VarStringPtr;

      BYTE i;
      BYTE *DataSource;

    /*
     * First of all identify variable.
     */
    switch(var)
    {

    case VAR_LED_D5:
        *val = LATD5 ? '1':'0';
        break;

    case VAR_LED_D6:
        *val = LATD6 ? '1':'0';
        break;

    case VAR_ANAIN_AN0:
        *val = AN0String[(BYTE)ref];
        if ( AN0String[(BYTE)ref] == '\0' )
            return HTTP_END_OF_VAR;

        (BYTE)ref++;
        return ref;

    case VAR_ANAIN_AN1:
        *val = AN1String[(BYTE)ref];
        if ( AN1String[(BYTE)ref] == '\0' )
            return HTTP_END_OF_VAR;

        (BYTE)ref++;
        return ref;

    case VAR_DIGIN_RB5:
        *val = PORTB_RB0 ? '1':'0';
        break;


    case VAR_MAC_ADDRESS:
        if ( ref == HTTP_START_OF_VAR )
        {
            VarStringLen = 2*6+5;  // 17 bytes: 2 for each of the 6 address
bytes + 5 octet spacers

              // Format the entire string
            i = 0;
            VarStringPtr = VarString;
            while(1)
            {
                  *VarStringPtr++ = btohexa_high(AppConfig.MyMACAddr.v[i]);
                  *VarStringPtr++ = btohexa_low(AppConfig.MyMACAddr.v[i]);
                  if(++i == 6)
                        break;
```

```
            *VarStringPtr++ = '-';
        }
    }

        // Send one byte back to the calling function (the HTTP Server)
        *val = VarString[(BYTE)ref];

    if ( (BYTE)++ref == VarStringLen )
        return HTTP_END_OF_VAR;

    return ref;

case VAR_SERIAL_NUMBER:
    if ( ref == HTTP_START_OF_VAR )
    {
          // Obtain the serial number.  For this demo, we will call
          // the two low bytes of our MAC address (required to be
          // organization assigned) our board's serial number
          itoa(AppConfig.SerialNumber.Val, VarString);
        VarStringLen = strlen(VarString);
    }

        // Send one byte back to the calling function (the HTTP Server)
        *val = VarString[(BYTE)ref];

        // If this is the last byte to be returned, return
        // HTTP_END_OF_VAR so the HTTP server won't keep calling this
        // application callback function
    if ( (BYTE)++ref == VarStringLen )
        return HTTP_END_OF_VAR;

    return ref;

case VAR_IP_ADDRESS:
case VAR_SUBNET_MASK:
case VAR_GATEWAY_ADDRESS:
  // Check if ref == 0 meaning that the first character of this
  // variable needs to be returned
    if ( ref == HTTP_START_OF_VAR )
    {
          // Decide which 4 variable bytes to send back
          if(var == VAR_IP_ADDRESS)
              DataSource = (BYTE*)&AppConfig.MyIPAddr;
            else if(var == VAR_SUBNET_MASK)
              DataSource = (BYTE*)&AppConfig.MyMask;
            else if(var == VAR_GATEWAY_ADDRESS)
              DataSource = (BYTE*)&AppConfig.MyGateway;

          // Format the entire string
          VarStringPtr = VarString;
          i = 0;
          while(1)
          {
                itoa((WORD)*DataSource++, VarStringPtr);
                VarStringPtr += strlen(VarStringPtr);
                if(++i == 4)
                    break;
                *VarStringPtr++ = '.';
            }
            VarStringLen = strlen(VarString);
    }

        // Send one byte back to the calling function (the HTTP Server)
        *val = VarString[(BYTE)ref];

        // If this is the last byte to be returned, return
        // HTTP_END_OF_VAR so the HTTP server won't keep calling this
        // application callback function
    if ( (BYTE)++ref == VarStringLen )
        return HTTP_END_OF_VAR;
```

```
        return ref;

    case VAR_DHCP_TRUE:
    case VAR_DHCP_FALSE:
      // Check if ref == 0 meaning that the first character of this
      // variable needs to be returned
        if ( ref == HTTP_START_OF_VAR )
        {
              if((var == VAR_DHCP_TRUE) ^ AppConfig.Flags.bIsDHCPEnabled)
                  return HTTP_END_OF_VAR;

            VarStringLen = 7;
            VarString[0] = 'c';
            VarString[1] = 'h';
            VarString[2] = 'e';
            VarString[3] = 'c';
            VarString[4] = 'k';
            VarString[5] = 'e';
            VarString[6] = 'd';
        }

            *val = VarString[(BYTE)ref];

        if ( (BYTE)++ref == VarStringLen )
            return HTTP_END_OF_VAR;

        return ref;
    }

    return HTTP_END_OF_VAR;
}
#endif


#if defined(STACK_USE_FTP_SERVER) && defined(MPFS_USE_EEPROM)
ROM char const FTP_USER_NAME[]    = "ftp";
#undef FTP_USER_NAME_LEN
#define FTP_USER_NAME_LEN   (sizeof(FTP_USER_NAME)-1)

ROM char const FTP_USER_PASS[]    = "microchip";
#define FTP_USER_PASS_LEN   (sizeof(FTP_USER_PASS)-1)

BOOL FTPVerify(char *login, char *password)
{
    if ( !memcmppgm2ram(login, (ROM void*)FTP_USER_NAME, FTP_USER_NAME_LEN)
)
    {
        if ( !memcmppgm2ram(password, (ROM void*)FTP_USER_PASS,
FTP_USER_PASS_LEN) )
            return TRUE;
    }
    return FALSE;
}
#endif




/*****************************************************************
 * Function:        void InitializeBoard(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
```

```
 * Overview:        Initialize board specific hardware.
 *
 * Note:            None
 *******************************************************************/
static void InitializeBoard(void)
{
      // Set up analog features of PORTA
#if defined(USE_COMPATIBLE_AD)
    ADCON1  = 0b11001110;       // RA0 as analog input, Right justified
#else
      ADCON0 = 0b00000001;    // ADON, Channel 0
      ADCON1 = 0b00001101;    // Vdd/Vss is +/-REF, AN0 and AN1 are analog
      ADCON2 = 0b10000110;    // Right justify, no ACQ time, Fosc/64
#endif

#if defined(USE_LCD)
    TRISA   = 0x03;

    // LCD is enabled using RA5.
    PORTA_RA5 = 0;          // Disable LCD.
#else
    TRISA   = 0x23;
#endif

    // Turn off the LED's.
    TRISD = 0x00;
    LATD = 0x00;

    // Enable internal pull-ups.
    INTCON2_RBPU = 0;

#if defined(USE_LCD)
    XLCDInit();
    XLCDGoto(0, 0);
    XLCDPutROMString(StartupMsg);
#endif

    TXSTA = 0b00100000;     // Low BRG speed
    RCSTA = 0b10010000;
    SPBRG = SPBRG_VAL;

    T0CON = 0;
    INTCON_GIEH = 1;
    INTCON_GIEL = 1;
}

/*********************************************************************
 * Function:        void InitAppConfig(void)
 *
 * PreCondition:    MPFSInit() is already called.
 *
 * Input:           None
 *
 * Output:          Write/Read non-volatile config variables.
 *
 * Side Effects:    None
 *
 * Overview:        None
 *
 * Note:            None
 ********************************************************************/
static void InitAppConfig(void)
{
#if defined(MPFS_USE_EEPROM)
    BYTE c;
    BYTE *p;
#endif

    /*
     * Load default configuration into RAM.
```

```
     */
    AppConfig.MyIPAddr.v[0]     = MY_DEFAULT_IP_ADDR_BYTE1;
    AppConfig.MyIPAddr.v[1]     = MY_DEFAULT_IP_ADDR_BYTE2;
    AppConfig.MyIPAddr.v[2]     = MY_DEFAULT_IP_ADDR_BYTE3;
    AppConfig.MyIPAddr.v[3]     = MY_DEFAULT_IP_ADDR_BYTE4;

    AppConfig.MyMask.v[0]       = MY_DEFAULT_MASK_BYTE1;
    AppConfig.MyMask.v[1]       = MY_DEFAULT_MASK_BYTE2;
    AppConfig.MyMask.v[2]       = MY_DEFAULT_MASK_BYTE3;
    AppConfig.MyMask.v[3]       = MY_DEFAULT_MASK_BYTE4;

    AppConfig.MyGateway.v[0]    = MY_DEFAULT_GATE_BYTE1;
    AppConfig.MyGateway.v[1]    = MY_DEFAULT_GATE_BYTE2;
    AppConfig.MyGateway.v[2]    = MY_DEFAULT_GATE_BYTE3;
    AppConfig.MyGateway.v[3]    = MY_DEFAULT_GATE_BYTE4;

    AppConfig.MyMACAddr.v[0]    = MY_DEFAULT_MAC_BYTE1;
    AppConfig.MyMACAddr.v[1]    = MY_DEFAULT_MAC_BYTE2;
    AppConfig.MyMACAddr.v[2]    = MY_DEFAULT_MAC_BYTE3;
    AppConfig.MyMACAddr.v[3]    = MY_DEFAULT_MAC_BYTE4;
    AppConfig.MyMACAddr.v[4]    = MY_DEFAULT_MAC_BYTE5;
    AppConfig.MyMACAddr.v[5]    = MY_DEFAULT_MAC_BYTE6;

#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
    AppConfig.Flags.bIsDHCPEnabled = TRUE;
#else
    AppConfig.Flags.bIsDHCPEnabled = FALSE;
#endif

#if defined(MPFS_USE_EEPROM)
    p = (BYTE*)&AppConfig;


    XEEBeginRead(EEPROM_CONTROL, 0x00);
    c = XEERead();
    XEEEndRead();

    /*
     * When a record is saved, first byte is written as 0x55 to indicate
     * that a valid record was saved.
     */
    if ( c == 0x55 )
    {
        XEEBeginRead(EEPROM_CONTROL, 0x01);
        for ( c = 0; c < sizeof(AppConfig); c++ )
            *p++ = XEERead();
        XEEEndRead();
    }
    else
        SaveAppConfig();
#endif
}

#if defined(MPFS_USE_EEPROM)
static void SaveAppConfig(void)
{
    BYTE c;
    BYTE *p;

    p = (BYTE*)&AppConfig;
    XEEBeginWrite(EEPROM_CONTROL, 0x00);
    XEEWrite(0x55);
    for ( c = 0; c < sizeof(AppConfig); c++ )
    {
        XEEWrite(*p++);
    }

    XEEEndWrite();
}
#endif
```

```
BOOL StringToIPAddress(char *str, IP_ADDR *buffer)
{
    BYTE v;
    char *temp;
    BYTE byteIndex;

    temp = str;
    byteIndex = 0;

    while( v = *str )
    {
        if ( v == '.' )
        {
            *str++ = '\0';
            buffer->v[byteIndex++] = atoi(temp);
            temp = str;
        }
        else if ( v < '0' || v > '9' )
            return FALSE;

        str++;
    }

    buffer->v[byteIndex] = atoi(temp);

    return (byteIndex == 3);
}



#define MAX_USER_RESPONSE_LEN   (20)
void ExecuteMenuChoice(MENU_CMD choice)
{
    char response[MAX_USER_RESPONSE_LEN];
    IP_ADDR tempIPValue;
    IP_ADDR *destIPValue;

    USARTPut('\r');
    USARTPut('\n');
    USARTPutROMString(menuCommandPrompt[choice-'0'-1]);

    switch(choice)
    {
    case MENU_CMD_SERIAL_NUMBER:
        itoa(AppConfig.SerialNumber.Val, response);
        USARTPutString((BYTE*)response);
        USARTPut(')');
        USARTPut(':');
        USARTPut(' ');

        if ( USARTGetString(response, sizeof(response)) )
        {
            AppConfig.SerialNumber.Val = atoi(response);

            AppConfig.MyMACAddr.v[4] = AppConfig.SerialNumber.v[1];
            AppConfig.MyMACAddr.v[5] = AppConfig.SerialNumber.v[0];
        }
        else
            goto HandleInvalidInput;

        break;

    case MENU_CMD_IP_ADDRESS:
        destIPValue = &AppConfig.MyIPAddr;
        goto ReadIPConfig;

    case MENU_CMD_GATEWAY_ADDRESS:
        destIPValue = &AppConfig.MyGateway;
```

```
            goto ReadIPConfig;

        case MENU_CMD_SUBNET_MASK:
            destIPValue = &AppConfig.MyMask;

        ReadIPConfig:
            DisplayIPValue(destIPValue, FALSE);
            USARTPut(')');
            USARTPut(':');
            USARTPut(' ');

            USARTGetString(response, sizeof(response));

            if ( !StringToIPAddress(response, &tempIPValue) )
            {
HandleInvalidInput:
                USARTPutROMString(InvalidInputMsg);
                while( !USARTIsGetReady() );
                USARTGet();
            }
            else
            {
                destIPValue->Val = tempIPValue.Val;
            }
            break;


        case MENU_CMD_ENABLE_AUTO_CONFIG:
            AppConfig.Flags.bIsDHCPEnabled = TRUE;
            break;

        case MENU_CMD_DISABLE_AUTO_CONFIG:
            AppConfig.Flags.bIsDHCPEnabled = FALSE;
            break;

        case MENU_CMD_DOWNLOAD_MPFS:
#if defined(MPFS_USE_EEPROM)
            DownloadMPFS();
#endif
            break;

        case MENU_CMD_QUIT:
#if defined(MPFS_USE_EEPROM)
            SaveAppConfig();
#endif
            break;
        }
    }




    static void SetConfig(void)
    {
        MENU_CMD choice;

        do
        {
            USARTPutROMString(menu);
            choice = GetMenuChoice();
            if ( choice != MENU_CMD_INVALID )
                ExecuteMenuChoice(choice);
        } while(choice != MENU_CMD_QUIT);

    }


    #if defined(MPFS_USE_EEPROM)

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

```
 * Function:         BOOL DownloadMPFS(void)
 *
 * PreCondition:     MPFSInit() is already called.
 *
 * Input:            None
 *
 * Output:           TRUE if successful
 *                   FALSE otherwise
 *
 * Side Effects:     This function uses 128 bytes of Bank 4 using
 *                   indirect pointer.  This requires that no part of
 *                   code is using this block during or before calling
 *                   this function.  Once this function is done,
 *                   that block of memory is available for general use.
 *
 * Overview:         This function implements XMODEM protocol to
 *                   be able to receive a binary file from PC
 *                   applications such as HyperTerminal.
 *
 * Note:             In current version, this function does not
 *                   implement user interface to set IP address and
 *                   other informations.  User should create their
 *                   own interface to allow user to modify IP
 *                   information.
 *                   Also, this version implements simple user
 *                   action to start file transfer.  User may
 *                   evaulate its own requirement and implement
 *                   appropriate start action.
 *
 ********************************************************************/
#define XMODEM_SOH      0x01
#define XMODEM_EOT      0x04
#define XMODEM_ACK      0x06
#define XMODEM_NAK      0x15
#define XMODEM_CAN      0x18
#define XMODEM_BLOCK_LEN 128

static BOOL DownloadMPFS(void)
{
    enum SM_MPFS
    {
        SM_MPFS_SOH,
        SM_MPFS_BLOCK,
        SM_MPFS_BLOCK_CMP,
        SM_MPFS_DATA,
    } state;

    BYTE c;
    MPFS handle;
    BOOL lbDone;
    BYTE blockLen;
    BYTE lResult;
    BYTE tempData[XMODEM_BLOCK_LEN];
    TICK lastTick;
    TICK currentTick;

    state = SM_MPFS_SOH;
    lbDone = FALSE;

    handle = MPFSFormat();

    /*
     * Notify the host that we are ready to receive...
     */
    lastTick = TickGet();
    do
    {
        /*
         * Update tick here too - just in case interrupt is not used.
         */
```

```
    TickUpdate();

    currentTick = TickGet();
    if ( TickGetDiff(currentTick, lastTick) >= (TICK_SECOND/2) )
    {
        lastTick = TickGet();
        USARTPut(XMODEM_NAK);

        /*
         * Blink LED to indicate that we are waiting for
         * host to send the file.
         */
        LATA2 ^= 1;
    }

} while( !USARTIsGetReady() );


while(!lbDone)
{
    /*
     * Update tick here too - just in case interrupt is not used.
     */
    TickUpdate();

    if ( USARTIsGetReady() )
    {
        /*
         * Toggle LED as we receive the data from host.
         */
        LATA2 ^= 1;
        c = USARTGet();
    }
    else
    {
        /*
         * Real application should put some timeout to make sure
         * that we do not wait forever.
         */
        continue;
    }

    switch(state)
    {
    default:
        if ( c == XMODEM_SOH )
        {
            state = SM_MPFS_BLOCK;
        }
        else if ( c == XMODEM_EOT )
        {
            /*
             * Turn off LED when we are done.
             */
            LATA2 = 1;

            MPFSClose();
            USARTPut(XMODEM_ACK);
            lbDone = TRUE;
        }
        else
            USARTPut(XMODEM_NAK);

        break;

    case SM_MPFS_BLOCK:
        /*
         * We do not use block information.
         */
        lResult = XMODEM_ACK;
```

```
            blockLen = 0;
            state = SM_MPFS_BLOCK_CMP;
            break;

        case SM_MPFS_BLOCK_CMP:
            /*
             * We do not use 1's comp. block value.
             */
            state = SM_MPFS_DATA;
            break;

        case SM_MPFS_DATA:
            /*
             * Buffer block data until it is over.
             */
            tempData[blockLen++] = c;
            if ( blockLen > XMODEM_BLOCK_LEN )
            {
                /*
                 * We have one block data.
                 * Write it to EEPROM.
                 */
                MPFSPutBegin(handle);

                lResult = XMODEM_ACK;
                for ( c = 0; c < XMODEM_BLOCK_LEN; c++ )
                    MPFSPut(tempData[c]);

                handle = MPFSPutEnd();

                USARTPut(lResult);
                state = SM_MPFS_SOH;
            }
            break;

        }

    }


/*
 * This small wait is required if SLIP is in use.
 * If this is not used, PC might misinterpret SLIP
 * module communication and never close file transfer
 * dialog box.
 */
#if defined(STACK_USE_SLIP)
    {
        BYTE i;
        i = 255;
        while( i-- );
    }
#endif
    return TRUE;
}

#endif


#if defined(USE_LCD)
void XLCDDelay15ms(void)
{
    DelayMs(15);
}
void XLCDDelay4ms(void)
{
    DelayMs(4);
}

void XLCDDelay100us(void)
```

```
{
    INTCON_GIEH = 0;
    Delay10us(1);
    INTCON_GIEH = 1;
}
#endif
```

// Buraya

3. SONUÇ

Bu çalışmada aynı bilgisayar ağındaki sistemlerin web arayüzü ile kontrolü sağlanabildiği gibi artık oldukça yaygınlaşmış durumdaki internet üzerinden de uzaktaki bir sistemden bilgi alınması ve bu sistemin kontrolü rahatça sağlanabilmektedir. Sistemin avantajları arasında kendisi için kullanıcıyla arasında maliyetli bir özel bağlantı sağlanması yerine hali hazırda bulunan internet bağlantısının kullanılması ve bu internet bağlantısı üzerinden kendisine bağlı cihazların kontrolünde herhangi bir bilgisayar gereksinimi duymamasıdır ve bu sayede maliyet oldukça makul boyutlarda tutulabilmektedir. Sistemin avantajları ve uygulanabilir alanlara örnekler aşağıda listelenmiştir.

Avantajları

- Düşük maliyet
- Kendi üzerinde web sunucusu bulundurması ve kullanıcının bu sunucuya rahatça ulaşabilmesi
- Alternatif sistemlere göre hızlı olması
- İnternete bağlanabilen herhangi bir cihazdan (bilgisayar, cep telefonu, cep bilgisayarı vb.) bağlanılabilme özelliği
- Bilgisayardan bağımsız kendi başına çalışan bir sistem olması
- Ethernet arayüzü olan herhangi bir cihazla entegre olabilme özelliği
- Bilgisayar kontrollü bir sisteme göre 1/10 malitette olması
- Kolay kullanım (Herhangi bir web göstericisi (Internet explorer , Firefox vs.) üzerinden erişilebilirlik.
- Sistemdeki ufak değişikliklerle sadece ev otomasyonu değil hertürlü bilgi alış-verişi gereken sistemlere kolayca uyarlanabilmesi.

Uygulama alanları

- Günümüzde gelişmekte olan akıllı ev aletlerinin kontrolü ile
    - Kullanıcının ev ısısını eve geldiğinde arzu ettiği sıcaklıkta olması için uzaktan kontrol edebileceği ısıtma-soğutma sistemleri
    - İçindeki yiyecek-içeceklerin listesini görebileceğimiz akıllı buzdolabından bilgi alabilme
    - Kullanıcının eve girdiğinde yıkama programını bitirmek üzere çalışabilen çamaşır makinesinin prrogramlarını ve çalışma zamanını ayarlayabilme
    - Kendi kendine istenilen odanın temizliğini yapabilen elektrik süpürgelerinin
    - Gelen telefon aramalarının listesine web sayfası üzerinden uzaktan erişilebilmesi
    - Zile basıldığında kameradan fotoğraf çekilip kullanıcının eve gelen kişilerin listesini görebilmesi

  gibi cihazların kolayca web arayüzü üzerinden kontrolü sağlanabilecek ev otomasyonu sistemleri

- Güvenlik firmalarının müşterilerinin mülklerinde istenilen bölgeleri uzaktan kontrol edebilme, izleyebilme sistemleriyle entegresi
- Su , doğalgaz, elektrik tesisat bilgilerinin ve harcamalarının uzaktan erişilebilmesi

Kaynaklar

- www.microchip.com
- ENC28J60 datasheet
- PIC18f4620 datasheet