

AVANT DE COMMENCER LIRE EN DETAIL L'APPLICATION MICROCHIP AN833.

(L'application MICROCHIP AN1120 est un excellent cours sur le protocole TCP-IP, qu'il est indispensable de connaître avant de fabriquer un site web)

<http://irp.nain-t.net/doku.php/start> site complet sur TCP-IP, http, le routage, les classes d'adresses ...

Réaliser un serveur WEB avec un PIC : voir document original complet ici :

http://www.aix-mrs.iufm.fr/formations/filieres/ge/data/PIC/PICC/indexPIC_C.htm)

Ce document est un complément de mise à jour lors de l'usage d'un P18F46xx ou P18F26xx avec la nouvelle pile TCP-IP V4.55

Tout d'abord télécharger et installer la « pile TCPIP » de Microchip (actuellement V4.55)


C:\Microchip Solutions\Microchip\TCPIP Stack ne doit pas être modifié.

C:\Microchip Solutions\Microchip\Include\TCPIP Stack contient les « header » et ne doit pas être modifié.

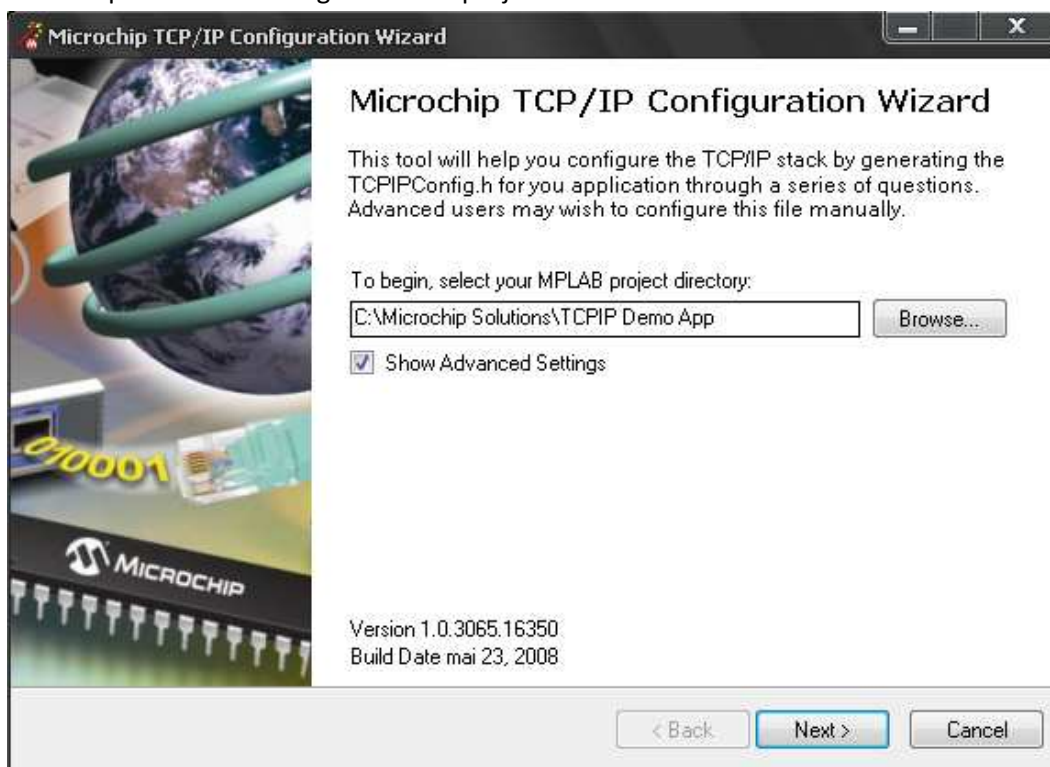
C:\Microchip Solutions\TCPIP Demo App contient un exemple de serveur web avec un projet MPLAB pour C18. C'est ce projet qui sert de base pour le projet final. Il doit être au préalable modifié pour un P18F46xx ou P18F26xx (ici un P18F4620), tel quel le code ne tient pas dans la mémoire de 64KO de ce microcontrôleur. Ce document explique comment configurer la partie « matériel » et les parties « logiciel » en C18 sur le PIC et en HTML sur le serveur WEB.

TCPIPConfig.exe et MPFS2.EXE se trouvent dans C:\Microchip Solutions\Microchip\TCPIP Stack\Utilities

Configuration de TCPIPconfig.h

 Microchip propose un utilitaire TRES pratique pour configurer les options TCP-IP : TCPIPConfig.exe
La pile TCP-IP est proposée par défaut avec toutes les options ce qui entraîne une occupation ROM d'environ 96 KO !!! Incompatible avec un P18F4620 ne disposant que de 64KO. Il est nécessaire de désactiver les options qui ne seront pas utiles.

Avant de lancer TCPIPconfig.exe effectuer une sauvegarde du dossier « TCPIP Demo App », c'est dans ce dossier que se fait la configuration du projet.



Ne conserver que les services suivants :

WEB server, obligatoire, c'est le serveur web

MPFS file system, obligatoire pour la gestion des fichiers du serveur

DHCP client, en cas de configuration automatique par un serveur DHCP)

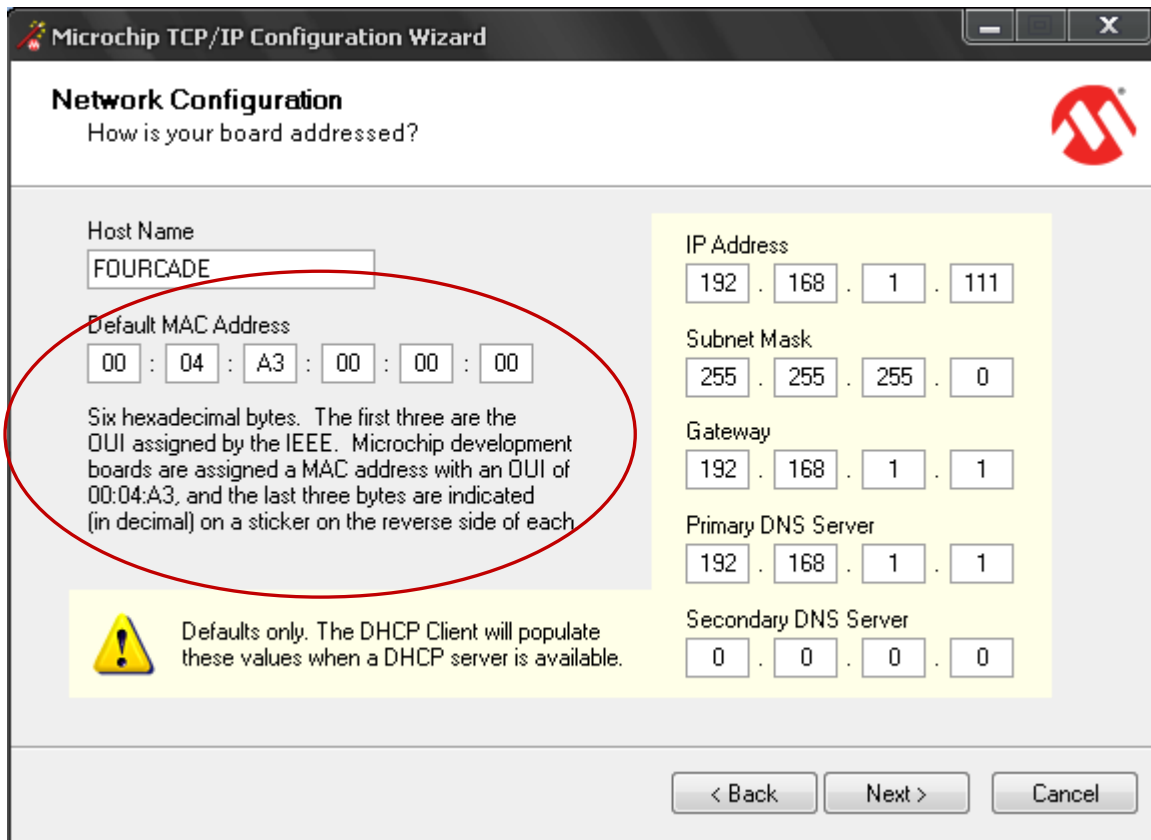
ICMP server, si l'on souhaite disposer d'une réponse au ping.

DNS client, si l'on souhaite pouvoir accéder à un serveur par son nom

Announce service, permet de localiser le serveur Microchip par l'utilitaire Microchip Ethernet Discoverer.exe

NetBIOS Name service, pour pouvoir utiliser les noms NetBIOS

UART support, si l'on souhaite pouvoir configurer le serveur par UART (ou utiliser l'UART)

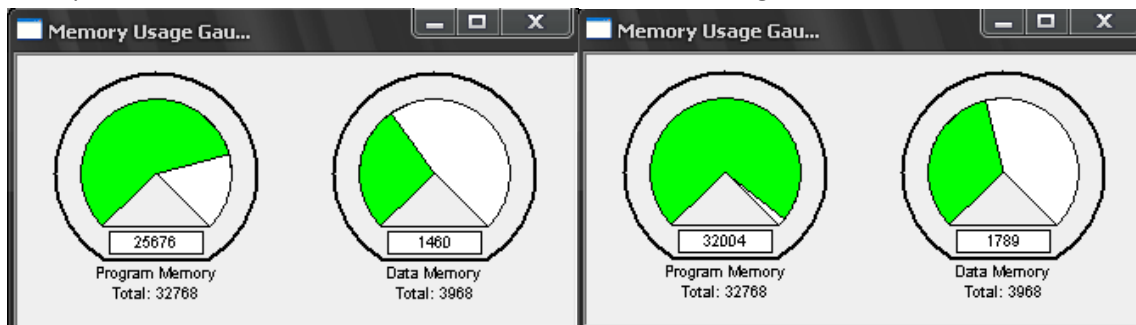


Attention, compléter correctement cette page.

Choisir ensuite la configuration du serveur WEB HPPT CLASSIC ou HTTP2. HTTP2 est plus performant et plus simple à exploiter mais occupe beaucoup plus de place en ROM .

Exemple avec HTTP CLASSIC

Même configuration mais avec HTTP2



Le choix mémoire pour les pages html est généralement EXTERNAL EEPROM (mémoire EEPROM SPI)

En cas d'utilisation de HTTP2, et de manque de place en ROM ne valider que « POST support », afin de bénéficier des fonctionnalités POST HTML/JAVA sans consommer trop de ROM.

Laisser « TCP socket configuration » en automatique

Décocher si nécessaire Use TX cheksum afin de gagner « un peu » de place en ROM

Modifications de maindemo.c

Configuration communications avec **osc 8MHz** (interne/externe), code à placer dans **maindemo.c** rechercher la fonction **static void InitializeBoard(void)**

La configuration automatique de l'USART posant des problèmes, il est préférable de remplacer cette partie du code par :

```
OSCCONbits.IRCF2=1;
OSCCONbits.IRCF1=1;
OSCCONbits.IRCF0=1;
OSCTUNEbits.PLLEN=1; // PLL (x4)
OSCCONbits.SCS1=0; // 1 pour osc int 0 pour quartz
ADCON1=0x0F; SPBRG = 51; // configure la vitesse (BAUD) 9600
SPBRGH= 0x06;
BAUDCONbits.BRG16=0; // SPBRGH = 0 pour 9600
TXSTA = 0b00100000;
RCSTA = 0b10010000; // active l'USART
TXSTAbits.BRGH=0; // High Speed
```

Modifications de hardware.h

Le fichier **hardware.h** contient la définition du matériel sur PICDEM2+ et P18F4620

Tout d'abord, rechercher et décommenter : **#define YOUR_BOARD**

```
// Choose which hardware profile to compile for here. See
// the hardware profiles below for meaning of various options.
```

...

```
#define YOUR_BOARD
```

introduire ensuite le code suivant après la ligne : **#elif defined (YOUR_BOARD)**

La démo utilise 8 LED et 8 boutons, il n'y a que 4 LED et 2 boutons sur PICDEM2+ d'où une répétition des définitions. Il est souhaitable de disposer au moins d'un bouton afin de passer en mode configuration après le RESET et d'une LED afin de visualiser l'activité du serveur WEB .

```
// Define your own board hardware profile here
// #define CLOCK_FREQ (3200000) // en Hz pour PICDEM2+ avec PIC18F4620, horloge
interne F MAX (4x8MHz)
/*****/
/*****/
// PICDEM2+ Ethernet PICTail
// I/O pins
#define LED0_TRIS (TRISBbits.TRISB1)
#define LED0_IO (PORTBbits.RB1)
#define LED1_TRIS (TRISBbits.TRISB1)
#define LED1_IO (PORTBbits.RB1)
#define LED2_TRIS (TRISBbits.TRISB1)
#define LED2_IO (PORTBbits.RB1)
#define LED3_TRIS (TRISBbits.TRISB3)
#define LED3_IO (PORTBbits.RB1)
```

```

#define LED4_TRIS          (TRISBbits.TRISB1)          // 1 LED seulement
#define LED4_IO            (PORTBbits.RB1)
#define LED5_TRIS          (TRISBbits.TRISB1)
#define LED5_IO            (PORTBbits.RB1)
#define LED6_TRIS          (TRISBbits.TRISB1)
#define LED6_IO            (PORTBbits.RB1)
#define LED7_TRIS          (TRISBbits.TRISB1)
#define LED7_IO            (PORTBbits.RB1)

#define S2_TRIS (TRISAbits.TRISA0)
#define S2_IO (PORTAbits.RA0)
#define S3_TRIS (TRISBbits.TRISB0)
#define S3_IO (PORTBbits.RB0)
#define BUTTON0_TRIS      S3_TRIS // Button0
#define BUTTON0_IO        S3_IO
#define BUTTON1_TRIS      S2_TRIS // Button1
#define BUTTON1_IO        S2_IO
#define BUTTON2_TRIS      S2_TRIS
#define BUTTON2_IO        S2_IO
#define BUTTON3_TRIS      S2_TRIS
#define BUTTON3_IO        S2_IO

// ENC28J60 I/O pins (brochage identique au PICTAIL ethernet)
//-----
//----- Pour PICDEM2+ -----
//-----
#define ENC_CS_TRIS        (TRISBbits.TRISB3)
#define ENC_CS_IO          (LATBbits.LATB3)
#define ENC_SCK_TRIS       (TRISCbits.TRISC3)
#define ENC_SDI_TRIS       (TRISCbits.TRISC4)
#define ENC_SDO_TRIS       (TRISCbits.TRISC5)
#define ENC_SPI_IF         (PIR1bits.SSPIF)
#define ENC_SSPBUF          (SSPBUF)
#define ENC_SPISTAT        (SSPSTAT)
#define ENC_SPISTATbits    (SSPSTATbits)
#define ENC_SPICON1        (SSPCON1)
#define ENC_SPICON1bits    (SSPCON1bits)
#define ENC_SPICON2        (SSPCON2)

// 25LC256 I/O pins
#define EEPROM_CS_TRIS     (TRISBbits.TRISB4)
#define EEPROM_CS_IO       (LATBbits.LATB4)
#define EEPROM_SCK_TRIS    (TRISCbits.TRISC3)
#define EEPROM_SDI_TRIS    (TRISCbits.TRISC4)
#define EEPROM_SDO_TRIS    (TRISCbits.TRISC5)
#define EEPROM_SPI_IF      (PIR1bits.SSPIF)
#define EEPROM_SSPBUF      (SSPBUF)
#define EEPROM_SPICON1     (SSPCON1)
#define EEPROM_SPICON1bits (SSPCON1bits)
#define EEPROM_SPICON2     (SSPCON2)
#define EEPROM_SPISTAT     (SSPSTAT)
#define EEPROM_SPISTATbits (SSPSTATbits)
// La définition du LCD est inutile sur une carte sans LCD
// Elle entraîne automatiquement la génération du code des fonctions LCD
// LCD Module I/O pins
#define FOUR_BIT_MODE
typedef struct
{
    unsigned char data : 4; // Bits 0 through 3
    unsigned char : 4; // Bits 4 through 7
} LCD_DATA;
#define LCD_DATA_TRIS      (((volatile LCD_DATA*)&TRISD)->data)

```

```
#define LCD_DATA_IO          (((volatile LCD_DATA*)&LATD)->data)
#define LCD_RD_WR_TRIS      (TRISAbits.TRISA2)
#define LCD_RD_WR_IO        (LATABits.LATA2)
#define LCD_RS_TRIS         (TRISAbits.TRISA3)
#define LCD_RS_IO           (LATABits.LATA3)
#define LCD_E_TRIS          (TRISAbits.TRISA1)
#define LCD_E_IO            (LATABits.LATA1)
/*****/
#define LED_GET()           (LATB)
#define LED_PUT(a)          (LATB = (a))
/*****/
```

Toujours dans **hardware.h** et en cas d'utilisation de l'oscillateur interne il est nécessaire de configurer le microcontrôleur en inversant les commentaires après la ligne : **#elif defined(__18F4620)**

Remplacer 4620 par le numéro de votre PIC 18F

```
// PICDEM Z board
#pragma config OSC=HSPLL, WDT=OFF, MCLRE=ON, PBADEN=OFF, LVP=OFF // osc externe
// Pour un oscillateur interne commenter la ligne précédente et dé-commenter les trois
// lignes suivants.
//#pragma config OSC = INTIO67 //pour INTRC-OSC2 as RA6, OSC1 as RA7
//#pragma config WDT = OFF //pour watch dog timer disable
//#pragma config LVP = OFF //pour low voltage program disable
#endif
```

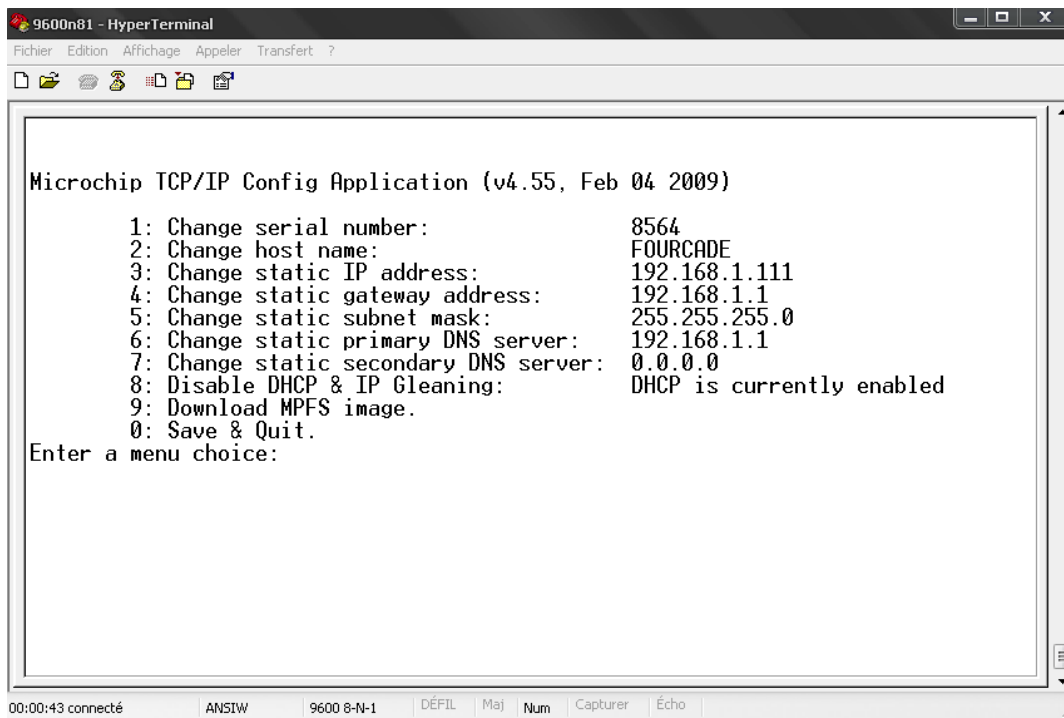
L'horloge de référence pour les temporisations (delay) doit être ajustée (4x8Mhz avec la PLL) dans

```
// Clock frequency value.
// This value is used to calculate Tick Counter value
#if defined(__18CXX)
    // All PIC18 processors
...
#else
#define GetSystemClock() (32000000uL) // Hz
```

Compiler le projet TCPIP Demo App-C18, exemple pour http CLASSIC

Des essais peuvent être réalisés avec une carte PICDEM2+ avec P18F4620 connecté à une carte PICTAIL Ethernet (voir document précédent pour le câblage).

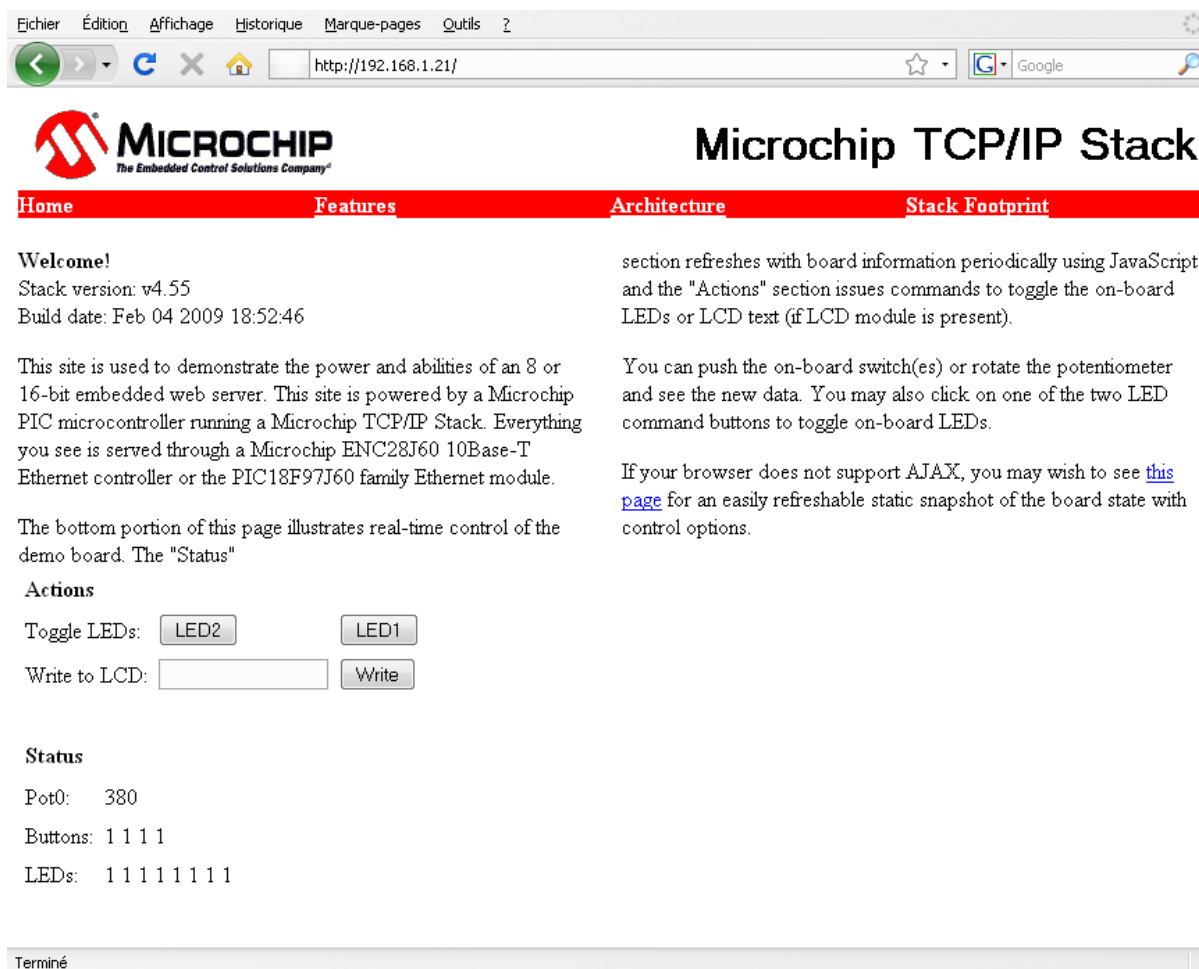
Pour télécharger le site web dans l'EEPROM appuyer sur S3 avant le RESET, cela lance les communications sur l'USART (9600,N,8,1 avec Q=8MHz):



Le choix 9 permet de télécharger le site web dans l’EEPROM de la carte PICTAIL ETHERNET (fichier MPFSimg.bin avec un serveur HTTP CLASSIC) avec le protocole XMODEM.

Relancer ensuite le serveur (RESET ou choix 0)

Connecter vous sur l’adresse IP du serveur (ici 192.168.1.111) ou l’adresse fournie par le DHCP et visible sur l’afficheur LCD du KIT à l’aide d’un navigateur



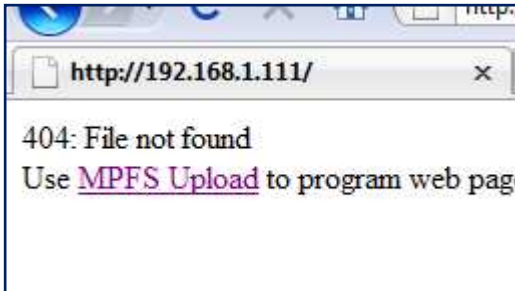
Compiler le projet TCPIP Demo App-C18, exemple pour HTTP2

Des essais peuvent être réalisés avec une carte PICDEM2+ avec P18F4620 connecté à une carte PICTAIL Ethernet. (voir document précédent pour le câblage).

La pile HTTP2 offre des possibilités et performances **nettement supérieures à HTTP CLASSIC**.

ReConfigurer le projet avec TCPIPConfig.exe en **validant HTTP2** et uniquement l'option « POST » afin de limiter la taille du code à moins de 64KO.

Lors du lancement le site demande une mise à jour en ligne, télécharger le fichier MPFSImg2.bin



Le site WEB est alors opérationnel, les nombreuses possibilités peuvent être testées dans la démo.

Programmation de l'application liens http<->C18 (serveur HTTP2)

Mise à jour de variables : méthode POST

Afin de comprendre le principe de cette méthode de mise à jour, nous allons créer un site web **beaucoup plus simple** que celui de la démo et **affichant la tension sur l'entrée AN0** d'un PIC18F4620 (potentiomètre RAO du KIT PICDEM2+).

- 1) Recopier le dossier **WevPage2** en **WebPage_Application**
- 2) Dans ce dernier ne conserver que les fichiers :
index.html
mchp.js
status.xml
- 3) A l'aide d'un éditeur html (KompoZer par exemple) éditer index.html comme ceci :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <script src="/mchp.js" type="text/javascript"></script>
</head>
<body>
Tension sur AN0:
<span id="pot0"
  style="font-size: 150px; text-align: center; font-weight: bold; color:
  rgb(0, 0, 0);">?</span>mV<br>

<script type="text/javascript"><!--
// Parses the xmlResponse from status.xml and updates the status box
function updateStatus(xmlData) {
// Check if a timeout occurred
if(!xmlData)
```

```

{
document.getElementById('display').style.display = 'none';
document.getElementById('loading').style.display = 'inline';
return;
}
// Update the POT value
document.getElementById('pot0').innerHTML = getXMLValue(xmlData, 'pot0'); }
setTimeout("newAJAXCommand('status.xml', updateStatus, true)",500);
//--> </script>
</body>
</html>

```

Ce fichier inclut le strict « minimum »

le fichier JAVASCRIPT `mchps.js` qui contient les fonctions java de communications avec la pile TCP-IP

L’affichage de la `variable locale pot0` et ses paramètres de police, taille etc...

Le script de la fonction `updateStatus`, qui appelle régulièrement (ici toutes les 500mS) la fonction `document.getElementById`, cette dernière demande une mise à jour des variables contenues dans le fichier `status.xml`

- 4) Placer dans le fichier `status.xml` les variables à mettre à jour :

```

<response>
<pot0>~pot~</pot0>
</response>

```

Ici `pot0` (du fichier html) renvoie à la fonction C18 `void HTTPPrint_pot(void)` du fichier `CustomHTTPApp.C` par la déclaration `~pot~`.

Le `~pot~` aura pour effet d’appeler une fonction dont le nom sera `void HTTPPrint_pot(void)`

Les fonctions `HTTPPrint_xxx(void)` sont déclarées dans le fichier `HTTPPrint.h`

Exemple :

```

void HTTPPrint(DWORD callbackID)
{
    switch(callbackID)
    {
        case 0x0000001c:
            HTTPPrint_pot();
            break;
        default:
            // Output notification for undefined values
            TCPPutROMArray(sktHTTP, (ROM BYTE*)"!DEF", 4);
    }
}

```

Il faut bien sur modifier `HTTPPrint.h` en fonction des besoins de mise à jour, ce qui complique les choses.

- 5) **MPFS2.EX s’occupe de tout cela !!!**

Il compile le site, met à jour le fichier `HTTPPrint.h` et si la gestion des noms NETBIOS est active, il télécharge éventuellement le site.

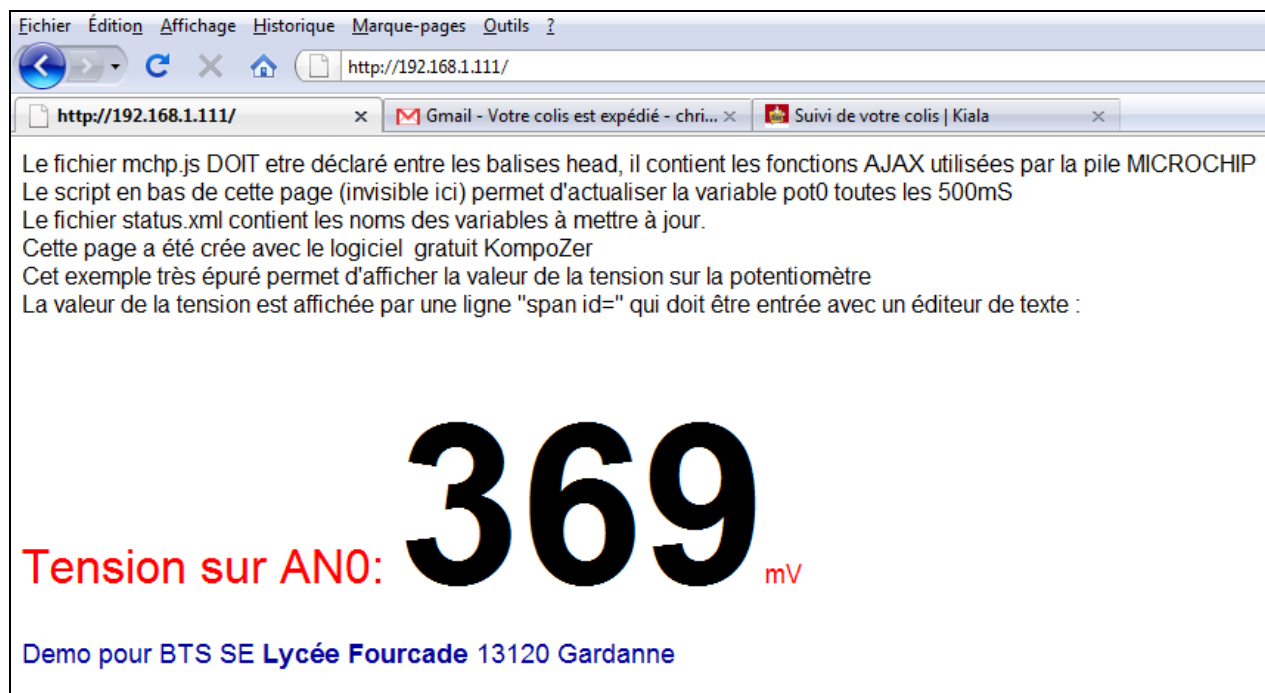
6) Mise à jour de l'application dans CustomHTTPApp.C

Dans l'exemple proposé on affiche la tension sur AN0, il faut créer une fonction

void HTTPPrint_pot(void) comme suit :

```
void HTTPPrint_pot(void)
{
    BYTE AN0String[8];
    WORD ADval;
    // Lance et attend la fin de conversion sur AN0
    ADCON0bits.GO = 1;
    while(ADCON0bits.GO);
    // conversion pour obtenir des mV
    ADval = (WORD)(((long)(ADRES)*500)/1024);
    // convertit la valeur dans une chaine ASCII
    utoa(ADval, AN0String);
    // envoie cette chaine vers le client HTTP
    TCPPutString(sktHTTP, AN0String);
}
```

7) Compiler, télécharger, exécuter et voilà ...



Matériel utilisé : PICDEM2+ avec P18F4620 et PICTAIL Ethernet

Cablage décrit ici :

http://www.aix-mrs.iufm.fr/formations/filieres/ge/data/PIC/PICC/indexPIC_C.htm)

Projet compilé avec MPLAB 8.20, et la pile Microchip TCP-IP V4.55

(<http://www.microchip.com>)

Pages WEB réalisées avec KompoZer : <http://kompozer.net/>