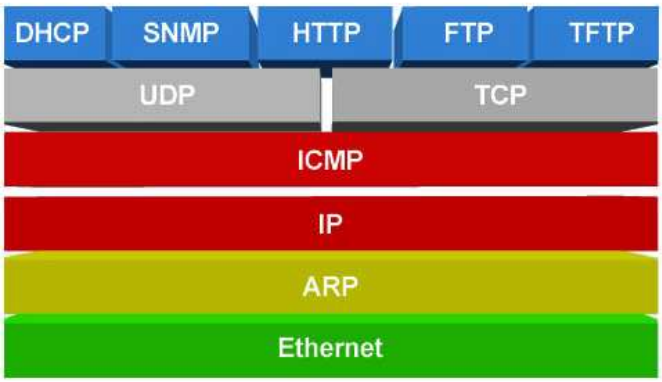
Réaliser un site WEB embraqué. PIC18 + ENC28J60


A partir de la note d'application Microchip AN833 et de la pile TCPIP « TCPIPStack 4.02 » disponibles sur www.microchip.com

Matériel : un PIC18F4620 (ou PIC18F2620)
un PICTAIL Ethernet ou son équivalent
ou : un PIC18Fx7J60 (avec contrôleur Ethernet intégré)


Logiciel : PILE TCP/IP MICROCHIP configurée pour le matériel ci-dessus et accompagnant ce document.

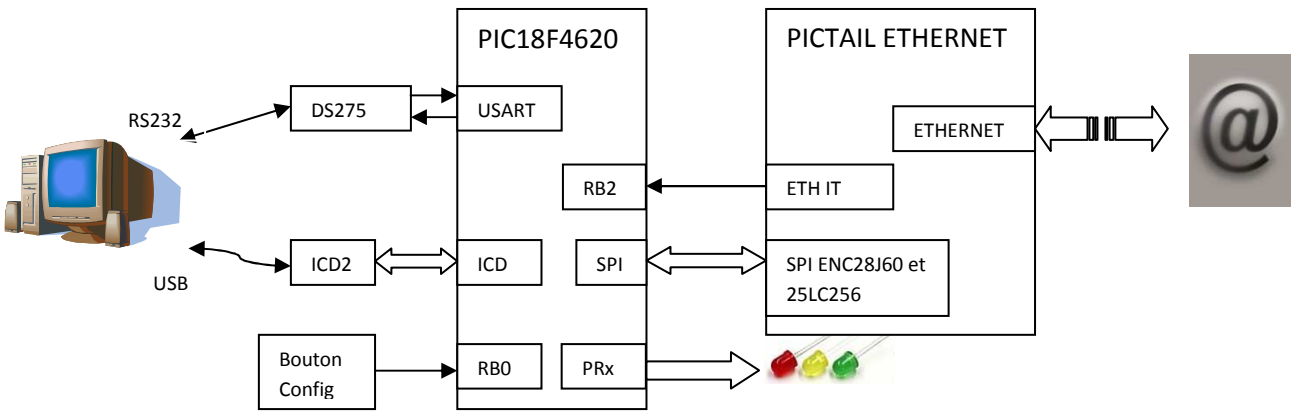
## 1) Présentation de la pile et protocoles TCP/IP supportés :



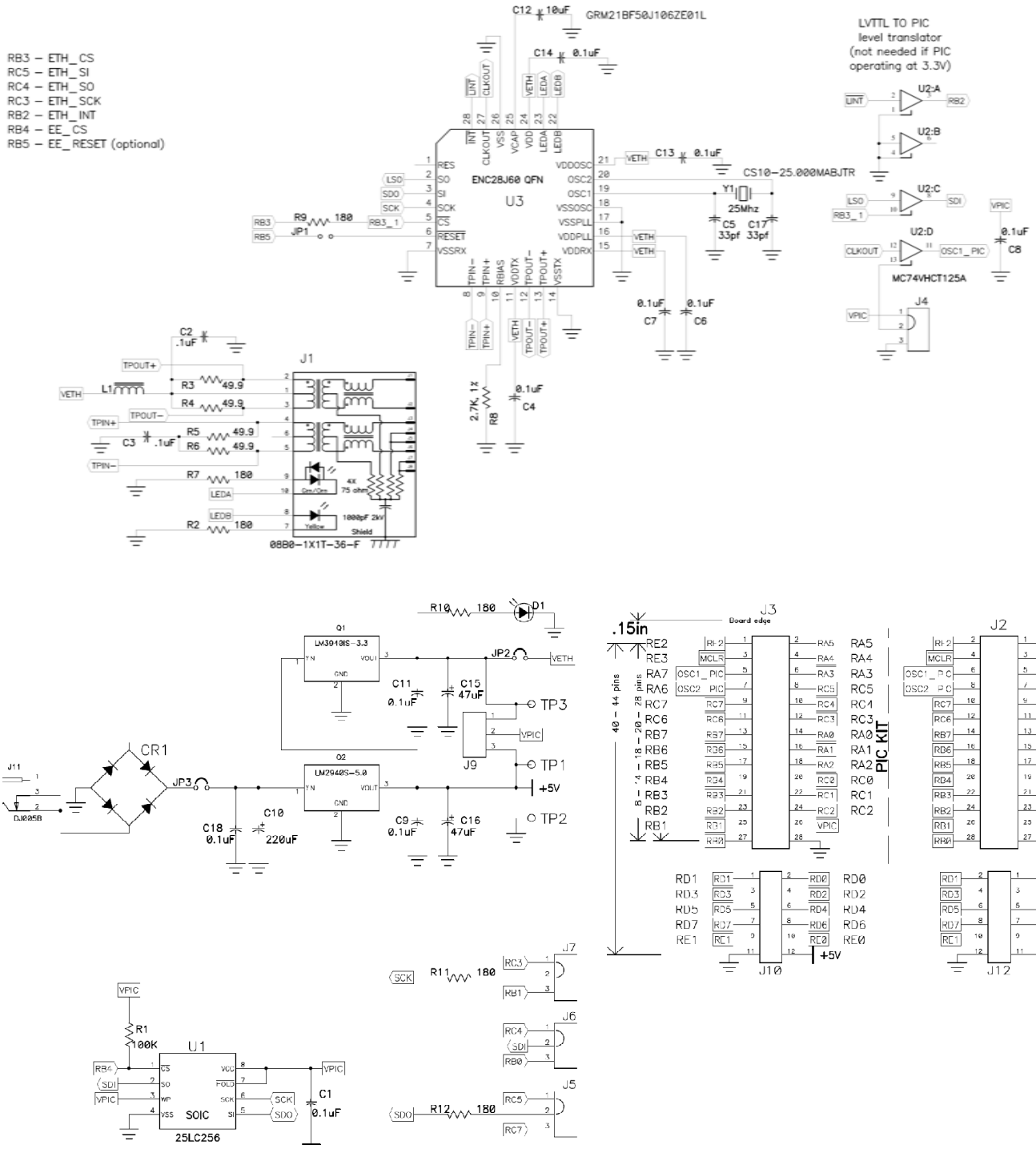

Microchip propose une pile logicielle TCP/IP libre de droits et optimisée pour les PIC18, PIC24 et les dsPIC®. La pile est une suite des programmes qui fournissent des services à toutes les applications basées sur TCP/IP. Les utilisateurs n'ont pas besoin de connaitre les complexités et caractéristiques de TCP/IP afin de l'employer. La pile est divisée en couches, chacune accédant à des services d'une ou plusieurs autres couches directement au-dessous d'elle. Certaines couches de TCP/IP sont dites « vivent », elles réagissent automatiquement quand un service est demandé ou lors de l'arrivée de paquet. La pile est de conception modulaire et est écrite en langage C.

## 2) Connexions électriques

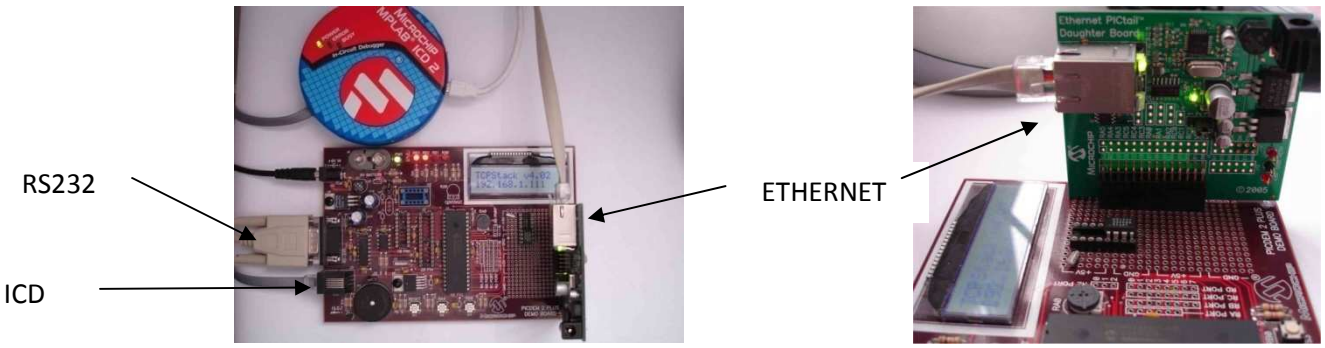
Le matériel comporte un PIC18F4620 utilisant son quartz intégré, un interface ICD, un interface RS232, avec un circuit de transfert de niveau type MAX232 ou DS275 (sans condensateurs externes), (une carte PICDEM2+ convient), ainsi qu'un interface Ethernet du même type que la carte « PICTAIL ETHERNET » , références AC164121 , prix 30€». Quelques LEDs suivant les options choisis et au moins un bouton (sur RB0) permettant de lancer le mode configuration. Moyennant quelques modifications à la configuration de la « PILE TCP/IP », le P18F4620 et le PICTAIL ETHERNET peuvent être remplacés par un P18Fx7J60.
Le logiciel proposé est configuré en fonction des connexions de la carte « PICTAIL ETHERNET », (CS, SI, SO etc…). La liaison ICD servant à programmer le PIC, la liaison RS232 permettant avec le logiciel « hyper terminal » de configurer le serveur WEB (voir page 4).




PICTAIL ETHERNET : schéma structurel




Exemple de câblage sur PICDEM2+ :



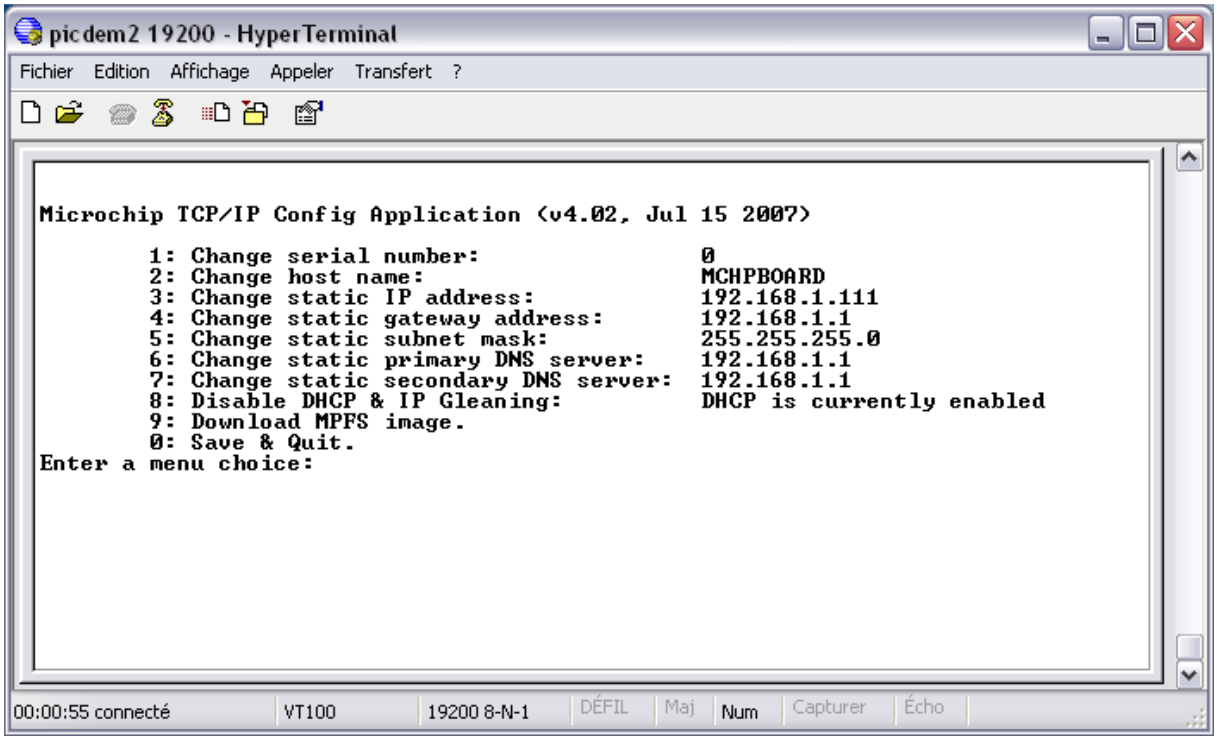
### 3) Essayer la demo

*Le logiciel accompagnant ce document est déjà configuré comme ci-dessous.*

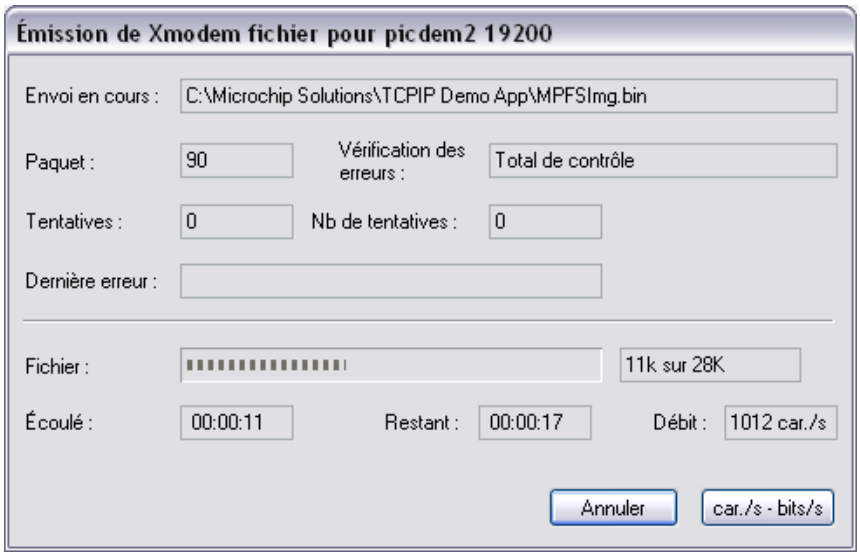
L'execution de « TCPIPStack 4.02.exe », crée un dossier « Microchip Solutions » contenant deux dossiers : « Microchip » et « TCTIP demo app ».

Ouvrir le projet TCPIP-demo-app-c18.mcw avec MPLAB, les fichiers du projet sont les suivants :

Après compilation et programmation du PIC18F4620, lancer le programme en appuyant sur S3(RB0).
La fenêtre de configuration apparaît dans l'HyperTerminal (19200,N,8,1).




Vérifier la conformité des paramètres avec celle du réseau puis télécharger le fichier de demo MPFS.img dans l'EEPROM en utilisant le protocole XMODEM.




Après téléchargement, appeler le serveur WEB à l'aide de l'internet explorer.


ENC28J60


Fichier Edition Affichage Favoris Outils ?
Précédente
Rechercher
Favoris
Adresse
http://192.168.1.11/
OK
Lycée M.M.Fourcade 13120 Gardanne
BTS Systèmes Electroniques
ACCUEIL
Caractéristiques
Architecture
La Pile TCP/IP
Bienvenue!
Pile TCP/IP version: v4.02
Compilée le : Jul 15 2007 18:51:58
Ce site montre les possibilités d'un serveur hébergé dans un PIC18Fxxxx associé à un interface ETHERNET Microchip ENC28J60.
Version modifiée pour PICDEM2+ avec PIC18F4620
CD 07/2007
Interface ETHERNET PICTAIL avec mémoire EEPROM 25LC256 hebergeant le site
RB3-ETH-CS ENC28J60 Chip Select
RC5-ETH-SI SPI
RC4-ETH-SO SPI
RC3-ETH-SCK SPI
RB2-ETH-INT IT ENC28J60
RB4-EE-CS EEPROM ChipSelect
RB5-EE-RESET
Le cadre ci dessous permet de lire :
- la valeur de la tension sur l'entrée AN0 du PIC
- l'état de la LED RB1 sur le PORTB
- l'état du bouton S3 (RB0)
- d'envoyer un message sur l'afficheur LCD du KIT
Si le navigateur ne permet d'afficher les images regardez cette page
Actions
Basculer LEDs: LED2 LED1
Ecrire sur LCD: ECRIRE
ETAT
Pot0: 298
Buttons: 0 0 0 1
LEDs: 1 1 1 1 1 1 1 1


## 4) Configuration du logiciel

Les fichiers du dossier « Microchip » **ne doivent pas être modifiés**, ils contiennent toutes les sources et utilitaires de la pile TCPIP de Microchip.
Le dossier « TCTIP demo app » contient un exemple complet de serveur WEB, en particulier les fichiers suivants qui doivent être adaptés à l'application (matériels et logiciels) :

| | |
|---|---|
| MainDemo.c | un exemple complet d'application |
| MPFSImg.c | fichier de data du site WEB (pour PIC24 et DSPIC) |
| HardwareProfile.h | description du matériel |
| *TCPIPCon*fig.h | description des application TCPIP activées |
| compile site.bat | batch de compilation du site WEB en MPFSImg.bin |
| TCPIP Demo App-C18.mcp | projet MPLAB |
| MPFSImg.bin | fichier binaire du site WEB téléchargeable dans l'EEPROM de la carte PICTAIL. |

**Les dossiers :**

| | |
|---|---|
| WebPages | site WEB du projet (sans arborescence) réalisé par exemple avec FrontPage. |
| Linker | les fichiers lkr |

Le fichier HardwareProfile.h décrit le matériel, il a été modifié et complété pour l'utilisation d'un PIC18F4620 (sur PICDEM2+ par exemple).
La ligne `#define YOUR_BOARD` ne doit pas être commentée.
Les lignes suivantes ont été ajoutées, elles décrivent le matériel de la carte PICDEM2+

```
// Define your own board hardware profile here
#define CLOCK_FREQ              (32000000)   //Hz (freq max interne d'un PIC18F4620)
// PICDEM2+  Ethernet PICtail
// I/O pins
    …
// ENC28J60 I/O pins
…
// 25LC256 I/O pins
…
// LCD Module I/O pins
#define FOUR_BIT_MODE
```

La gestion des E/S est automatique si elles sont définies. Par exemple la définition des connexions LCD entraine `#define USE_LCD`, ce « `define` » permet d'activer dans maindemo.c la gestion de l'afficheur LCD

*TCPIPCon*fig.h , ce fichier définit les services TCPIP utilisés , toutes les fonctions sont actives par défaut, il suffit de commenter celles qui sont inutiles, le programme compilé sera alors plus petit . Le programme une fois compilé fait environ 55KO, le PIC18F4620 ayant 64KO de ROM, il reste généralement assez de mémoire pour l'application. .

Configurer les lignes suivantes en fonction du réseau utilisé (IP par défaut, masque de sous réseau etc…)

```
#define MY_DEFAULT_HOST_NAME                "PIC_FOURCADE"

#define MY_DEFAULT_MAC_BYTE1            (0x00)
#define MY_DEFAULT_MAC_BYTE2            (0x04)
#define MY_DEFAULT_MAC_BYTE3            (0xA3)
#define MY_DEFAULT_MAC_BYTE4            (0x00)
#define MY_DEFAULT_MAC_BYTE5            (0x00)
#define MY_DEFAULT_MAC_BYTE6            (0x00)

#define MY_DEFAULT_IP_ADDR_BYTE1        (192ul)
#define MY_DEFAULT_IP_ADDR_BYTE2        (168ul)
#define MY_DEFAULT_IP_ADDR_BYTE3        (1ul)
#define MY_DEFAULT_IP_ADDR_BYTE4        (111ul)

#define MY_DEFAULT_MASK_BYTE1           (255ul)
#define MY_DEFAULT_MASK_BYTE2           (255ul)
#define MY_DEFAULT_MASK_BYTE3           (255ul)
#define MY_DEFAULT_MASK_BYTE4           (0ul)
```


MainDemo.c est une application complète (et complexe) de serveur WEB.
Les lignes suivantes ont été modifiées afin d'activer l'horloge interne du PIC18F4620 ainsi que la PLL, la fréquence interne étant alors de 8MHz.

```
#elif defined(__18F4620)      // ou 18F2620
#pragma config OSC = INTIO67, WDT=OFF, MCLRE=ON, PBADEN=OFF, LVP=OFF, XINST=OFF

InitializeBoard(void)a été modifié comme suit :
#if defined(__18F4620)        // config Q interne et vitesse MAX
       OSCCONbits.IRCF2=1;
       OSCCONbits.IRCF1=1;
       OSCCONbits.IRCF0=1;    // Base de temps 8MHz
       OSCTUNEbits.PLLEN=1;   // PLL active (x4)
       OSCCONbits.SCS1=1;     // osc interne
#endif
```

### 5) Algotithme de maindemo.c

`InitializeBoard();` initialise l'horloge, les E/S ,l'UART, le CAN, autorise les interruptions,

`TickInit();` initialise la base de temps de la machine d'état, utilise **TIMER0 !!!**

`MPFSInit();` initialise la gestion des fichier au format MICROCHIP (en EEPROM interne ou externe)

`InitAppConfig();` initialise les adresses paramètre Ethernet (ip, masque , DNS, routeur etc…) gestion de l'EEPROM externe.

Teste du bouton 0, si enfoncé plus de 4S, l'EEPROM est effacée, si moins de 4S, il y a appel de SetConfig() pour configuration par RS232.

Initialisation des services TCP/IP .

Debut de la boucle While(1) ;

Clignotement de la LED0 toutes les secondes

`StackTask();` valide la prochaine tâche TCPIP à effectuer

Exécute les services activés HTTP, FTP, SNMP, DHCP, client, serveur, TELNET …

Verifie que l'adresse IP n'a pas changé (par DHCP)

`ProcessIO();` c'est ici que l'on place le programme de l'application

Dans l'exemple, il n'y a qu'une conversion analogique numérique, (effectuée en boucle) le résultat étant rangé en ASCII dans AN0String

```
uitoa(*((WORD*)(&ADRESL)), AN0String);
```

Fin de la boucle While(1) ;

### 6) Créer des CGI en langage HTML

CGI : Common Gateway Interface (interface de passerelle commune). Les fichiers CGI sont écrits en langage HTML, ils ont la particularité de pouvoir envoyer des commandes au serveur ou de récupérer des données dynamiques.

Fonctions JAVASCRIPT proposées dans l'exemple de Microchip dans la version 4.02


Ces fonctions permettent une manipulation aisée des CGI, elles évitent la création de « FRAME » (voir page 8) mais alourdissent un peu le fichier HTML.

`function GetServerFile(FileName, AssignTo) :`
GetServerFile appelle un fichier cgi, **ce dernier contient le numéro de la variable recherchée**, le résultat est placé dans le dernier paramètre

AssignTo représente la balise ou doit être placée la réponse.
ex : `GetServerFile('Version.cgi','txtStackVersion')` envoie une requête http avec le numéro contenu dans le fichier Version.cgi et place la réponse en `<span id="txtStackVersion">`.

`function GetXmlHttpObject(handler) :` (utilisée par GetServerFile)
cette fonction vérifie la compatibilité du navigateur avec activex (exécution de cgi)

`function StateChanged() :` (utilisée par GetServerFile)
envoie une demande d'état (status) et met à jour la balise
`<span id="txtAutoUpdateStatus">Loading...</span>`

**Exemple : récupérer et afficher un texte**

Après la balise body

Fichier cgi contenant le numéro de la commande

Variable résultat

```
<body
bgcolor="white"
onload="GetServerFile('Version.cgi','txtStackVersion');
GetServerFile('BuildDate.cgi','txtBuildDate');
GetServerFile('Status.cgi','txtAutoUpdateStatus');">
```

A l'endroit ou le message doit être affiché :

```
Pile TCP/IP version: <span id="txtStackVersion">inconnue</span><br>
```

Le fichier BuildDate.cgi contient uniquement la ligne « 0x17 » qui fait référence à la variable `#define VAR_STACK_DATE (0x17),` le traitement par HTTPGetVar retournera la variable système MICROCHIP __DATE__.

Le fichier Status.cgi contient :

```
<table cellpadding="3">
       <tr><td>Pot0:</td><td>%02</td></tr><tr>
       <td>Buttons:</td>
       <td>%0F %0E %0D %04</td></tr><tr>
       <td>LEDs:</td> <td>%15 %14 %13 %12 %11 %10 %01 %00</td></tr>
</table>
```

Son appel affichera sur la balise « txtAutoUpdateStatus » l'état du potentiomètre Pot0 et des LEDs.

« onload » s'exécute une fois lors de l'ouverture de la page.

Il est possible d'appeler à partir d'un fichier HTML un fichier cgi, cela provoquera son execution. Un lien vers index.cgi existe dans le fichier index.htm. (version démo V4.02)

index.cgi (ce fichier est appellé depuis index.htm)

```
<html>
    <body>
           <b>Microchip TCP/IP Stack</b><br>
           Stack version: %16<br>
           Build date: %17<br>
           <br>
           <form method="get" action="0">
                  <b>Actions</b><br>
                  Toggle LEDs:<br>
                  <input type="submit" name="1" value="BASCULER LED2"></input>
```

Entraine l'appel de **HTTPExecCmd**


```
                <input type="submit" name="0" value="BASCULER LED1"></input>
            </form>
            <form method="get" action="1">
                Write to LCD:
                <input type="text" name="3" size="16"></input>
                <input type="submit" value="Write"></input>
            </form><br>
            <b>Status</b><br>
            Pot0: %02<br>
            Buttons: %0F %0E %0D %04<br>
            LEDs: %15 %14 %13 %12 %11 %10 %01 %00<br>
    </body>
</html>
```

Possibilité d'afficher une image en fonction d'une variable :
img src=LED%02.gif on peut créer par exemple  : LED0.gif et  LED1.gif, la valeur de %02 définira le fichier gif à afficher.

Possibilité de changer un lien en fonction d'une variable :
<a href=%02.htm>LinkName</a>

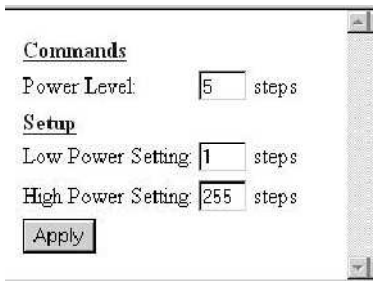
### 7) Gérer les commandes CGI.

La fonction `HTTPServer();` reçoit et transmet ces informations sur le port 80 (http)
Chaque commande ou paramètre possède un numéro, qui sera le même dans l'application serveur et dans les pages web (CGI)
Les requêtes avec plusieurs paramètres sont émises par une commande HTML « form » traitées par `void HTTPExecCmd(BYTE** argv, BYTE argc)`
Exemple : valeurs retournées par un clic sur le bouton "BASCULER LED1"
"http:192.168.0.2/0?0=BASCULER+LED1"
Il faudra : décoder la commande « 0 » avec pour valeur « BASCULER+LED1 », l'exécuter et éventuellement retourner un acquittement à l'adresse d'origine

```
// CGI Command Codes
#define CGI_CMD_DIGOUT      (0)
#define CGI_CMD_LCDOUT      (1)
#define CGI_CMD_RECONFIG     (2)

// Codes des Variables CGI. - 00h-FFh
// NOTE: dans les pages web (.cgi), utiliser la notation hexadécimale sur deux
caractères ex: "%04", "%2C"; pas "%4" ou "%02C"
#define VAR_LED0                    (0x00) // LED Outputs
#define VAR_LED1                    (0x01)
#define VAR_LED2                    (0x10)
#define VAR_LED3                    (0x11)
#define VAR_LED4                    (0x12)
…
```

Exemple :
<FORM METHOD=GET action=command.cgi>
<table> ...
<tr><td>Power Level:</td>
<td><input type=text size=2 maxlength=1 name=P value=%07></td><tr> ...
<tr><td><input type=submit name=B value=Apply></td></tr> ...




Message émit par l'explorer lorsque le bouton Apply est enfoncé :
10.10.5.110/command.cgi?P=5&L=1&H=255&B=Apply

`void HTTPExecCmd(BYTE** argv, BYTE argc)`
La fonction exécute la commande reçue par IP avec plusieurs arguments (argv)


```
Si la commande http est : thank.htm?name=Joe&age=25
argv[0] => thank.htm
argv[1] => name
argv[2] => Joe
argv[3] => age
argv[4] => 25
```

Utiliser argv[0] comme identificateur de commande
par exemple command = argv[0][0] - '0'; place le numéro de la commande dans « command »

```
var = argv[1][0] - '0'; retourne le numéro du premier argv.
```

Exemple de l'application :

```
    command = argv[0][0] - '0' ; // identifie le numéro de commande
    switch(command)
    {
    case CGI_CMD_DIGOUT:      // si le numero est 0 (ici commande un port E/S))
       var = argv[1][0] - '0'; // recupère le deuxième argument (ici numero de la LED)
        switch(var)
        {
        case CMD_LED1:// Si NAME=0
            LED1_IO ^= 1; // Bascule la LED.
            Break;
…
```

`WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)`
Cette fonction retourne la valeur du numéro de paramètre reçue par IP sous format ASCII

```
var         - identificateur de variable (%xx dans le fichier HTML)
ref         - référence indiquant si la variable a déjà été mise à jour.
val         - Buffer pour mémorisation.
output:      -   variable reference de l'application
```

Quand une variable est demandée sur un page WEB, HTTPServer appelle cette fonction
La fonction retourne la nouvelle valeur de la variable.

Cette fonction ne retourne qu'un caractère à chaque appel dans la variable locale « *val », elle devra être appelée plusieurs fois dans le cas de variables complexes.

Au départ `HTTPGetVar()` est appelé avec `ref = HTTP_START_OF_VAR` pour indiquer que c'est la premier appel. L'application doit utiliser cette référence pour commencer l'extraction de la valeur de la variable complexe et retourner la référence mise à jour de la variable suivante. S'il n'y a plus de valeur à retourner pour la variable complexe l'application retournera `HTTP_END_OF_VAR.`

**Exemple de l'application :**
Pour une variable simple (octet)

```
   switch(var)
    {
    case VAR_LED0:  *val = LED0_IO ? '1':'0'; // *val= ASCII(LED0_IO)
    break;
       …
```

Pour une variable complexe (chaine ASCII)

```
case VAR_ANAIN_AN0:          // chaine ASCII représentant la valeur de AN0
        *val = AN0String[(BYTE)ref];  //*val = l'octet "ref" de la chaine
                                      // *val va être envoyé par HTTPserver
        if(AN0String[(BYTE)ref] == '\0') return HTTP_END_OF_VAR;
       else if(AN0String[(BYTE)++ref] == '\0' ) return HTTP_END_OF_VAR;
```


```
// si c'est le dernier on retourne HTTP_END_OF_VAR sinon on incremente ref
        return ref;
```

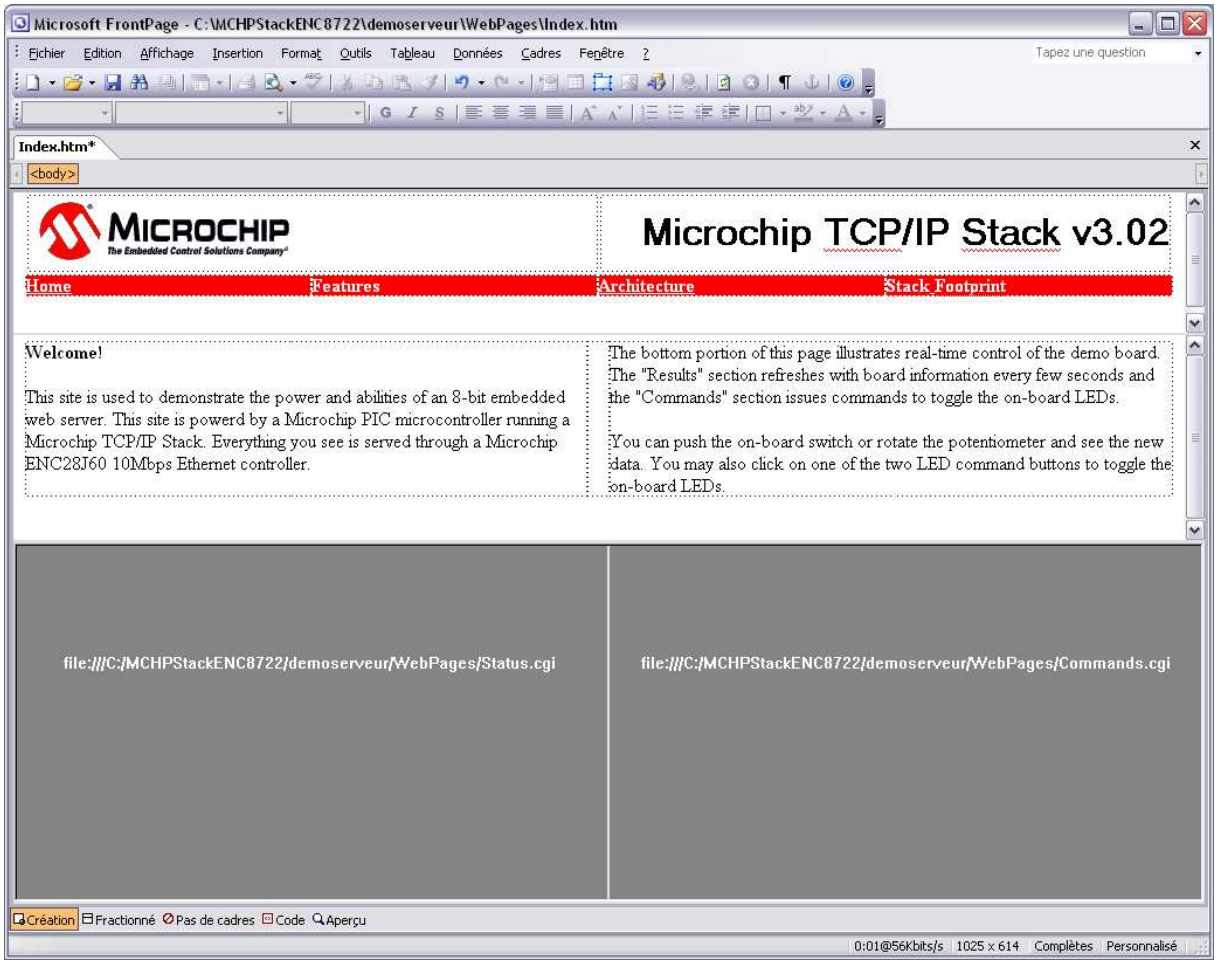
Pour une chaine de caractères (exemple STACK_VERSION)

```
#define VERSION   "v4.02"            // TCP/IP stack version
case VAR_STACK_VERSION:
 if(ref == HTTP_START_OF_VAR)
        {
                strncpypgm2ram((char*)VarString, VERSION, sizeof(VarString));
        }
 *val = VarString[(BYTE)ref];
 if(VarString[(BYTE)ref] == '\0') return HTTP_END_OF_VAR;
else if(VarString[(BYTE)++ref] == '\0' )return HTTP_END_OF_VAR;
 return ref;
```

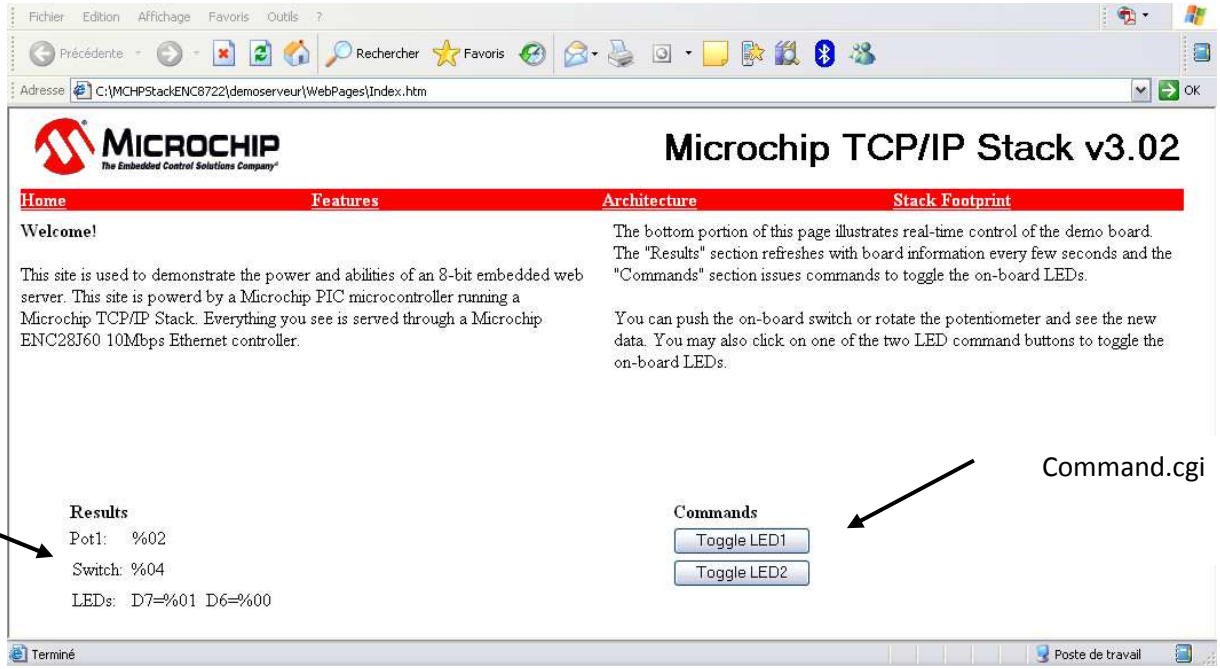
## 8) Simplification, l'utilisation de frames (V3.02 de la PILE)

Le découpage d'un page HTML en "FRAME" évite l'utilisation de fonctions spécifiques pour appeler les CGI. Il suffit de placer un lien dans le cadre souhaité vers un fichier CGI. (voir PILE MICROCHIP TCPIP V3.02)


Index.htm durant l'édition de la page (ici avec FRONTPAGE)




index.htm durant l'exécution (avec IE)




Header.htm
home.htm

Command.cgi

Status.cgi


index.htm.

```
<HTML>
<HEAD>
<TITLE>Microchip TCP/IP Stack Demo</TITLE></HEAD>
<FRAMESET rows="15%,40%,20%" border=0>
<FRAME name="top" src="Header.htm" marginheight=0 marginwidth=10>
<FRAME name="middle" src="Home.htm" marginheight=5 marginwidth=10>
<FRAMESET cols="50%,50%" border=0>
<FRAME name="left" src="Status.cgi" marginheight=2 marginwidth=50%>
<FRAME name="right" src="Commands.cgi" marginheight=2 marginwidth=50%>
</FRAMESET>
</FRAMESET>
</HTML>
```

Les fichiers Header.htm et Home.htm contiennent l'entête et le corps de la page (ici les logos MICROCHIP et le texte en anglais.

Status.cgi

%02 et %03 font références aux valeurs des tensions issues des potentiomètres
%00 et %01 font références à l'état des LED

```
<html>
<meta http-equiv="refresh" content="3">
<body>
<table>
<tr><td><b>Results</b></td></tr>
<tr><td>Pot1:</td><td>%02</td></tr>
<tr><td>Pot2:</td><td>%03</td></tr>
</table>
<table cellpadding="3">
<tr><td>Switch:</td><td>%04</td></tr>
<tr><td>LEDs:</td><td>D6=%01</td><td>D5=%00</td></tr>
</table>
</body>
</html>
```

Mise à jour toutes les 3S

Command.cgi

```
<html>
<body bgcolor="#FFFFFF">
<FORM METHOD=GET action=0>
<table>
<tr><td><b>Commands</b></td></tr>
<tr><td><input type=submit name=0 value="BASCULE LED1"></td></tr>
<tr><td><input type=submit name=1 value="BASCULE LED2"></td></tr>
</table>
</body>
</html>
```