

**MINF**  
**Mise en œuvre**  
**des microcontrôleurs PIC32MX**

# **Chapitre 10**

## **Gestion de l'Ethernet et de TCP/IP**

### **✂ T.P. PIC32MX**

**Christian HUBER (CHR)**  
**Serge CASTOLDI (SCA)**  
**Version 1.81 juin 2022**



## CONTENU DU CHAPITRE 10

<b>10.</b>	<b><i>Gestion de l'Ethernet et de TCP/IP</i></b>	<b>10-1</b>
<b>10.1.</b>	<b>Hardware Ethernet sur le kit PIC32MX95F512L</b>	<b>10-1</b>
<b>10.2.</b>	<b>Réalisation d'un serveur TCP</b>	<b>10-2</b>
10.2.1.	Utilisation d'un exemple et portage sur le kit	10-2
10.2.2.	Ouverture du projet et simplification	10-2
10.2.3.	Modification de la configuration Harmony	10-4
10.2.3.1.	Test initial	10-6
10.2.3.2.	Conclusion au sujet du portage de l'exemple	10-8
10.2.4.	Intégration du générateur	10-8
10.2.4.1.	Mise en place des fichiers du générateur	10-8
10.2.4.2.	Ajout des timers	10-9
10.2.4.3.	Modification appgen	10-10
10.2.5.	Détails du fonctionnement de l'exemple tcpip_tcp_server	10-11
10.2.5.1.	Détection état du lien TCP	10-12
10.2.5.2.	Réception et envoi de données TCP	10-14
10.2.5.3.	Conclusion au sujet de la modification de l'exemple	10-14
<b>10.3.</b>	<b>Réalisation d'un webserver</b>	<b>10-15</b>
10.3.1.	Utilisation d'un exemple et portage sur le kit	10-15
10.3.1.1.	Ouverture du projet	10-15
10.3.1.2.	Réduction des configurations	10-16
10.3.1.3.	Reduction du projet	10-17
10.3.2.	Modification de la configuration Harmony	10-18
10.3.2.1.	Sélection BSP pic32mx_skes	10-18
10.3.2.2.	Ajout appgen	10-19
10.3.2.3.	Modification device configuration	10-19
10.3.2.4.	Sélection du chip Ethernet physique	10-20
10.3.2.5.	Timers	10-21
10.3.2.6.	Test compilation	10-21
10.3.2.7.	Etat après modifications	10-21
10.3.2.8.	Réduction de la flash nécessaire	10-22
10.3.2.9.	Test initial	10-22
10.3.2.10.	Conclusion au sujet du portage de l'exemple	10-24
10.3.3.	Intégration du générateur	10-25
10.3.3.1.	Mise en place des fichiers du générateur	10-25
10.3.3.2.	Adaptation de system_init.c	10-26
10.3.3.3.	Adaptation de system_interrupt.c	10-27
10.3.3.4.	Adaptation de appgen.c	10-28
10.3.3.5.	Test du générateur	10-29
10.3.4.	Page web personnalisée	10-30
10.3.4.1.	Copie du répertoire web_pages	10-30
10.3.4.2.	Génération des fichiers	10-30
10.3.4.3.	Fichiers modifiés	10-32
10.3.4.4.	Adaptations du fichier custom_http_app.c	10-33
10.3.4.5.	Observation de la page Web	10-39
10.3.4.6.	Traitement en mode Post	10-39
10.3.4.7.	Rafraichissement automatique de la page web	10-39

**10.4. Conclusion** \_\_\_\_\_ **10-40****10.5. Historique des versions** \_\_\_\_\_ **10-40**

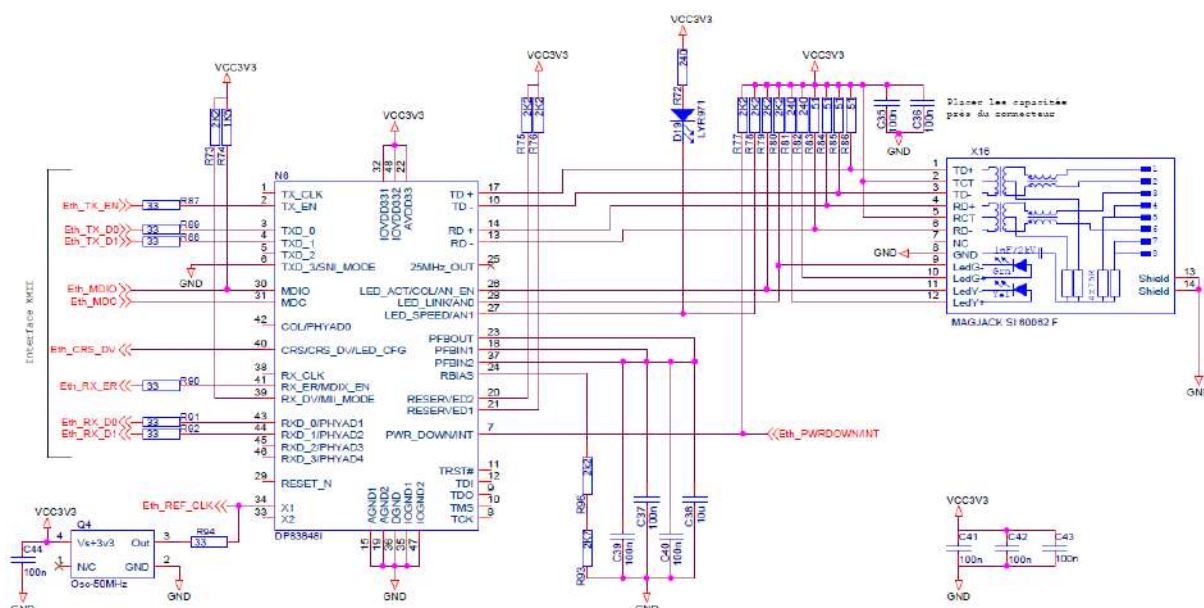
10.5.1.	Version 1.5 Mai 2015	10-40
10.5.2.	Version 1.51 Juin 2015	10-40
10.5.3.	Version 1.6 Juin 2016	10-40
10.5.4.	Version 1.6 b Juin 2016	10-40
10.5.5.	Version 1.7 mai 2017	10-40
10.5.6.	Version 1.71 mai 2019	10-40
10.5.7.	Version 1.8 avril 2022	10-40
10.5.1.	Version 1.81 juin 2022	10-40

## 10. GESTION DE L'ETHERNET ET DE TCP/IP

Ce chapitre a pour objectif de découvrir la gestion de l'Ethernet et de la pile TCP/IP fournies par Harmony.

## 10.1. HARDWARE ETHERNET SUR LE KIT PIC32MX95F512L

Voici la circuiterie Ethernet du kit PIC32MX795F512L qui utilise un circuit DP83848I (10/100 Mb/s Ethernet Physical Layer Transceiver) :



## 10.2. REALISATION D'UN SERVEUR TCP

Cette partie a pour objectif de découvrir comment modifier l'exemple Microchip `tcpip_tcp_server` pour aboutir au pilotage du générateur de signal à partir d'un client TCP distant.

### 10.2.1. UTILISATION D'UN EXEMPLE ET PORTAGE SUR LE KIT

Pour découvrir la réalisation d'un serveur TCP, nous allons utiliser l'exemple qui est sous :  
<Répertoire Harmony>\v<n>\apps\tcpip\tcpip\_tcp\_server

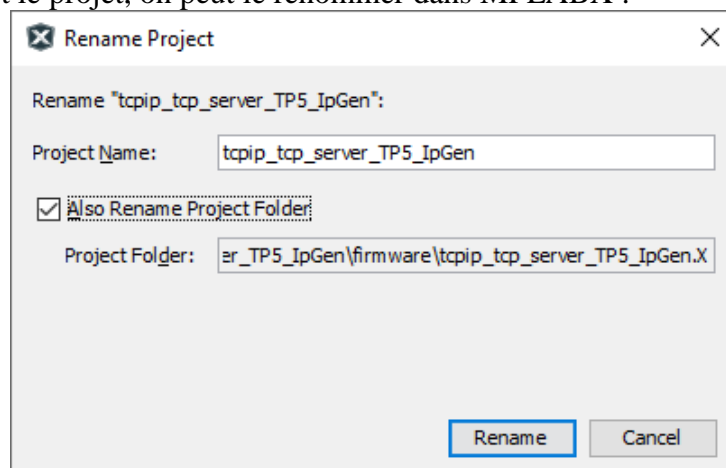
Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire `tcpip_tcp_server` au même niveau et nous le renommons `tcpip_tcp_server_TP5_IpGen`.

Le portage sur le kit SKES du projet suivant et son adaptation ont été réalisés avec les logiciels suivants :

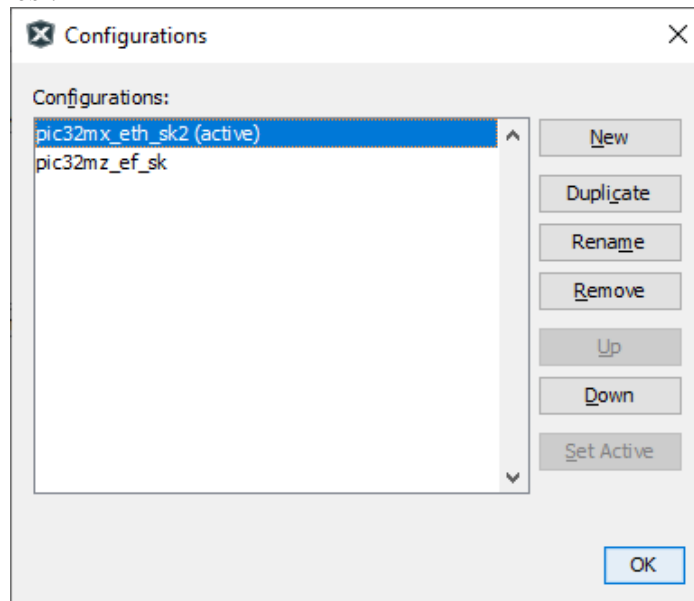
- Harmony v2.06
- MPLABX IDE v5.45
- XC32 v2.50

### 10.2.2. OUVERTURE DU PROJET ET SIMPLIFICATION

Après avoir ouvert le projet, on peut le renommer dans MPLABX :

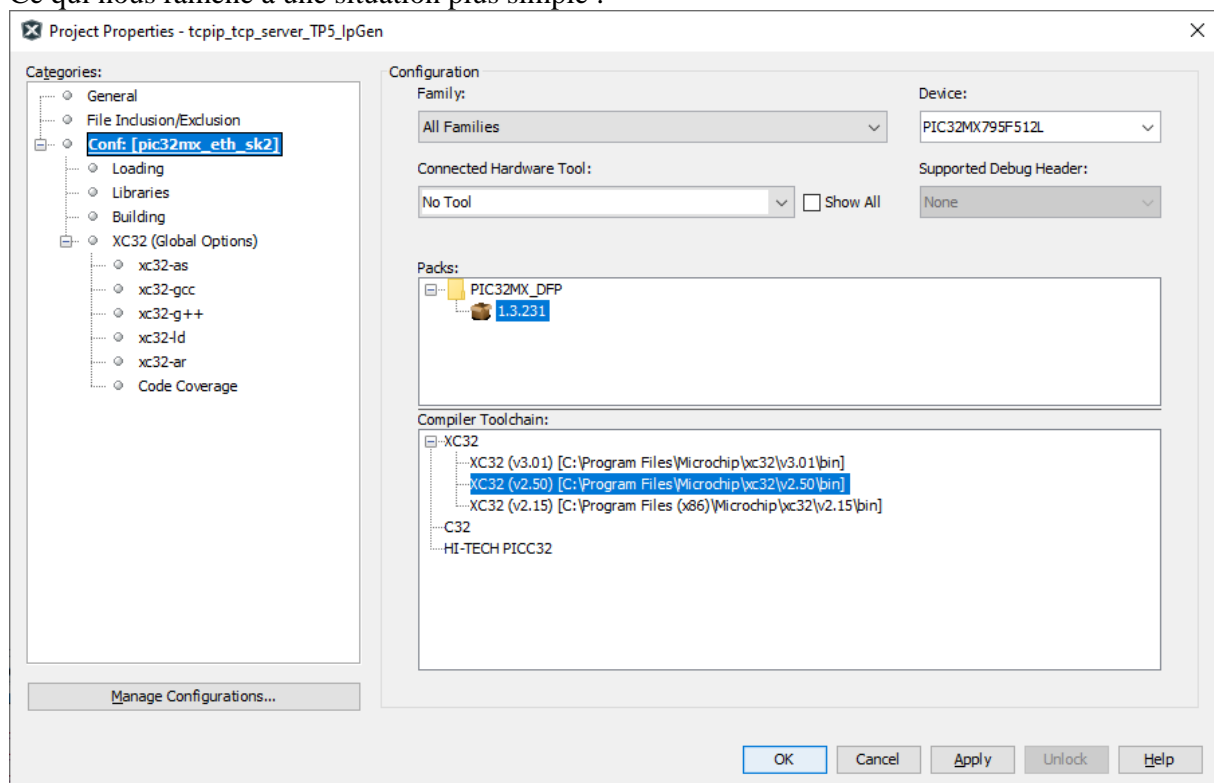


Puis, à partir des propriétés, dans « Manage Configurations », on va supprimer les configurations inutiles :



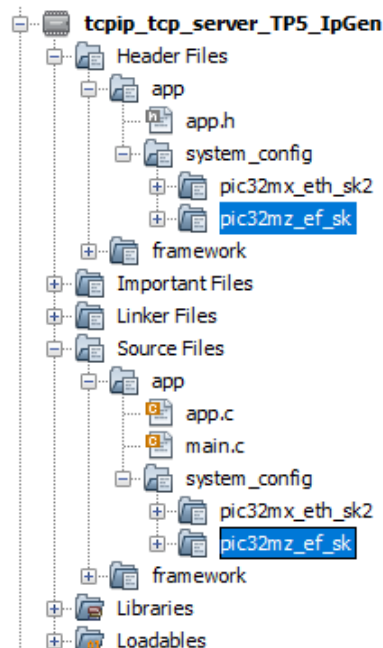
La configuration « pic32mx\_eth\_sk2 » est la plus proche de celle du starter-kit ES. On ne garde que celle-ci.

Ce qui nous ramène à une situation plus simple :



- Au besoin, régler la bonne version du compilateur.
- Ne pas oublier la coche xc32-gcc > preprocessing and messages > additional warnings.
- Afin de faciliter le debug, régler le niveau d'optimisation du compilateur à 0.

Dans « Header Files » et « Source Files », on peut également supprimer les fichiers devenus inutiles du fait de la suppression de la configuration :



Suite à cela, un « Clean and build » doit se passer sans erreur.

### 10.2.3. MODIFICATION DE LA CONFIGURATION HARMONY

On ouvre le MHC pour adapter la configuration à notre kit (principalement : choix du BSP, sélection et configuration du chip Ethernet physique et ajout d'une application supplémentaire).

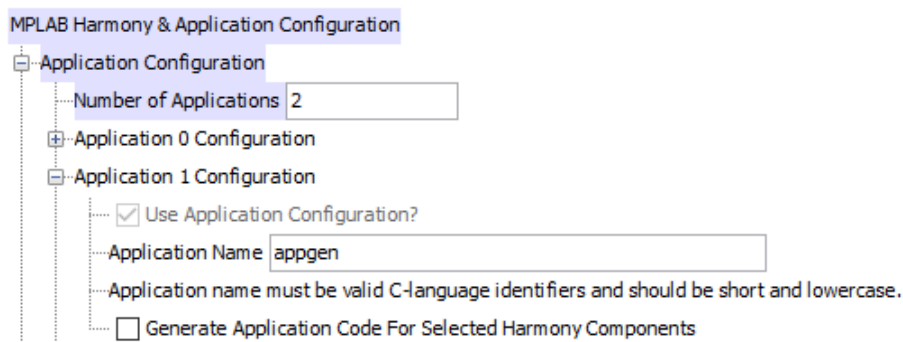
Commencer par sélectionner le BSP du starter-kit ES :



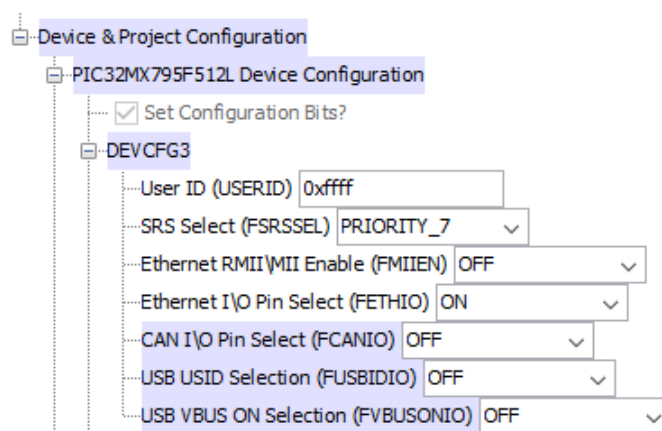
**Ne pas importer la configuration par défaut, sans quoi toute la configuration de l'exemple serait à refaire !**



Ajouter une 2<sup>ème</sup> application que nous appelons « appgen » :

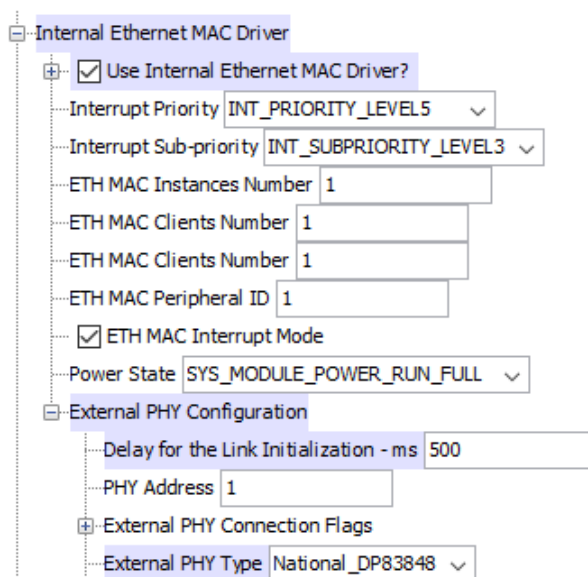


Modifier la configuration dans « Device configuration ». Dans DEVCFG3, régler FETHIO à ON :



On peut également éventuellement désactiver l'oscillateur secondaire dans DEVCFG1 (FOSCEN à OFF).

Comme la configuration utilisée est basée sur un kit utilisant un autre chip PHY, il faut sélectionner celui du starter-kit ES. Sous Drivers > Internal Ethernet MAC Driver > External PHY Configuration > External PHY Type, régler à « National\_DP83848 » :



Générer le code, accepter les modifications.

Suite à cela, un « Clean and build » doit se passer sans erreur.

### Etat après modifications

A ce stade, le projet a été porté; il doit être compilable et fonctionnel sur le kit SKES. Moyennant que l'on connaisse l'adresse IP attribuée au kit, on devrait pouvoir tester l'accès au serveur TCP.

#### 10.2.3.1. TEST INITIAL

##### 10.2.3.1.1. Configuration du BSP

Ce projet exemple étant destiné à un autre kit, il est possible, sous certaines conditions, qu'il faille reconfigurer les GPIO du PIC32 manuellement.

Cela peut se faire dans le MHC. Dans les fonctions de base nécessaires, on citera :

- Les 4 GPIO nécessaires aux signaux de contrôle du LCD à mettre en sortie,
- Les 8 GPIO nécessaires au LED à mettre en sortie, à l'état d'initialisation désiré.

##### 10.2.3.1.2. Ajout dans appgen.c

Dans la 2<sup>ème</sup> application, fichier appgen.c, on ajoute le code suivant à l'initialisation :

```
lcd_init();
lcd_bl_on();
printf_lcd("TP5 IpGen 2021");
lcd_gotoxy(1,2);
printf_lcd("SCA"); // nom(s) a adapter
```

Il est nécessaire d'ajouter au préalable : `#include "Mc32DriverLcd.h"`

##### 10.2.3.1.3. Affichage erreur init stack, modif app.c

Au niveau de app.c, dans le case APP\_TCPIP\_WAIT\_INIT, on transforme de la manière suivante :

```
case APP_TCPIP_WAIT_INIT:
    tcpipStat = TCPIP_STACK_Status(sysObj.tcpip);
    if(tcpipStat < 0)
    { // some error occurred
        SYS_CONSOLE_MESSAGE("APP: TCP/IP stack
                             initialization failed!\r\n");

        //ajout SCA
        lcd_gotoxy(1,4);
        printf_lcd("TCP/IP error !");
        appData.state = APP_TCPIP_ERROR;
```

Il est nécessaire d'ajouter au préalable : `#include "Mc32DriverLcd.h"`

#### 10.2.3.1.4. Affichage adresse IP

Dans le case APP\_TCPIP\_WAIT\_FOR\_IP, on peut afficher l'adresse IP à chaque changement au niveau de la boucle for :

```
// if the IP address of an interface has changed
// display the new value on the system console
nNets = TCPIP_STACK_NumberOfNetworksGet();

for (i = 0; i < nNets; i++)
{
    netH = TCPIP_STACK_IndexToNet(i);
    if(!TCPIP_STACK_NetIsReady(netH))
    {
        return;    // interface not ready yet!
    }
    ipAddr.Val = TCPIP_STACK_NetAddress(netH);
    if(dwLastIP[i].Val != ipAddr.Val)
    {
        dwLastIP[i].Val = ipAddr.Val;

        SYS_CONSOLE_MESSAGE(TCPIP_STACK_NetNameGet(netH));
        SYS_CONSOLE_MESSAGE(" IP Address: ");
        SYS_CONSOLE_PRINT("%d.%d.%d.%d \r\n", ipAddr.v[0],
            ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
        //ajout SCA : affichage adr. IP
        lcd_gotoxy(1,4);
        printf_lcd("IP:%03d.%03d.%03d.%03d  ", ipAddr.v[0],
            ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
    }
    appData.state = APP_TCPIP_OPENING_SERVER;
}
break;
```

#### 10.2.3.1.5. Résultat

Ce premier test permet de vérifier si la base fonctionne et de s'assurer que l'introduction du BSP ne perturbe pas les actions :

- Connecter le kit au réseau d'expérimentation.
- Le message d'annonce, puis l'adresse IP doivent s'afficher.  
Remarque : Les adresses IP 0.0.0.0, ou encore celle statique par défaut 192.168.100.115 peuvent apparaître avant la valeur effective.
- Si l'adresse IP est affichée et semble plausible (voir remarque ci-dessous en cas de problème), ouvrir un terminal TCP en tant que client (par exemple PuTTY en mode « raw ») et se connecter à l'adresse IP du kit sur le port TCP n°9760.

Le programme du PIC32 ouvre un serveur TCP qui écoute sur le port 9760 et reboucle les caractères reçus en majuscules. De plus, le serveur fermera la connexion TCP s'il reçoit le caractère correspondant à l'appui sur 'ESC'.

**Remarque**

Si le kit ne semble pas (toujours) obtenir d'adresse IP via DHCP, il est possible que cela provienne d'un timeout dû à un serveur DHCP trop lent. Ce cas de figure a pu être observé sur le réseau d'expérimentation de l'ES.

- Un symptôme peut être que l'adresse IP statique par défaut reste affichée (192.168.100.115), comme si on ne recevait pas de réponse DHCP.
- La solution est d'augmenter le timeout dans le MHC :  
Harmony Framework Configuration → TCP/IP Stack → DHCP Client → DHCP Request Time-out (seconds) → passer de 2s à 5s.  
Avec 5s, une configuration IP devrait être correctement reçue via DHCP et affichée.

**10.2.3.2. CONCLUSION AU SUJET DU PORTAGE DE L'EXEMPLE**

Cette partie a montré comment porter l'exemple Microchip sur le kit SKES. La partie suivante concerne l'intégration d'une application à l'exemple et les interactions avec le serveur TCP.

**10.2.4. INTEGRATION DU GENERATEUR**

Il s'agit maintenant d'intégrer les fichiers du générateur et de faire évoluer l'application. Il est encore nécessaire d'ajouter manuellement les timers 1 et 3.

Dans cet exemple, nous reprenons les fichiers issus de l'adaptation à l'USB.

**10.2.4.1. MISE EN PLACE DES FICHIERS DU GENERATEUR**

Il faut copier presque tous les fichiers source du TP générateur USB dans le répertoire du projet actuel:

```
<Répertoire Harmony>\v<n>\apps\tcpip\tcpip_tcp_server_TP5_IpGen
\firmware\src
```

**10.2.4.1.1. Résultat de l'ajout dans les header et sources**

#### 10.2.4.2. AJOUT DES TIMERS

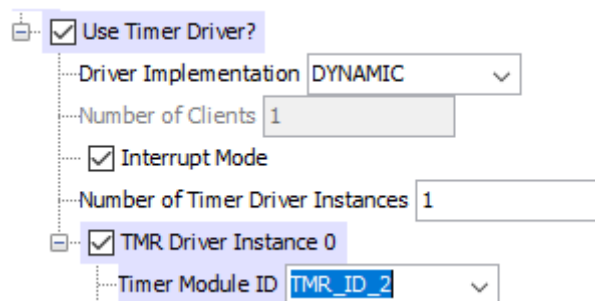
L'exemple de Microchip utilisé tcpip\_tcp\_server inclut un driver de timer dynamique. Ce driver utilise le timer 1.

Pour notre générateur de fonctions, nous utilisons les timers 1 et 3 en mode statique. Il nous faut des timings inférieurs à la milliseconde et aussi précis que possible. Un driver dynamique ne convient pas, et le MHC ne permet pas de mixer les types statique et dynamique.

La configuration de l'exemple ne convient donc pas au générateur de fonctions ; nous allons la modifier.

##### 10.2.4.2.1. Modification du timer utilisé par l'exemple

Dans le MHC, changer le timer dynamique utilisé de TMR\_ID\_1 en TMR\_ID\_2 afin de libérer le timer 1 (utilisé par notre générateur), puis régénérer le code.



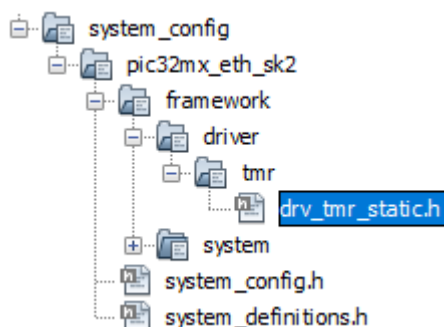
##### 10.2.4.2.2. Ajout des timers 1 et 3

Nous allons ici reprendre les fichiers des drivers statiques des timers 1 et 3 créés dans le projet du générateur USB. Il suffit de copier le répertoire **tmr** de l'ancien projet et de le copier sous :

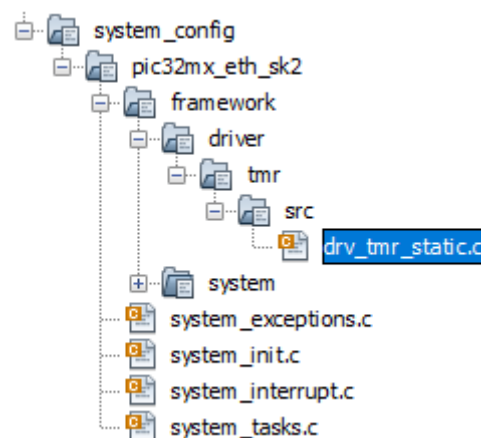
<Répertoire Harmony>\v<n>\apps\tcpip\tcpip\_tcp\_server\_TP5\_IpGen\firmware\src\system\_config\pic32mx\_eth\_sk2\framework\driver.

Ajouter les fichiers .h et .c dans le projet en recréant les logical folder :

Dans les header :



Dans les sources :



Il faut encore ajouter manuellement les appels aux fonctions d'initialisation des timers, par exemple dans la fonction `SYS_Initialize` ou encore dans `APPGEN_Initialize`. La deuxième alternative présente l'avantage de ne pas être écrasée par une régénération du code.

```
// ajout init drivers timers statiques
DRV_TMR0_Initialize();
DRV_TMR1_Initialize();
```

Un `#include` est nécessaire pour les appels des fonctions du driver de timer :

```
#include "driver/tmr/drv_tmr_static.h" //driver timer static
```

Finalement, copier le code des ISR des timers dans `system_interrupt.c` :

```
// timer 1 configure pour interrupt toutes les 1 ms
// anti-rebond et appgen
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0 (void)

// timer 3 pour la generation de signal
Void __ISR(_TIMER_3_VECTOR, ipl7AUTO)
    _IntHandlerDrvTmrInstance1 (void)
```

💡 Il est possible qu'il faille renommer une ISR pour éviter un doublon ! Le nouveau nom choisi est sans influence sur le reste du programme.

#### 10.2.4.3. MODIFICATION APPGEN

A ce stade, le code de `app_gen` peut être en grande partie repris du générateur USB, en adaptant les parties interagissant avec l'USB afin de communiquer en TCP.

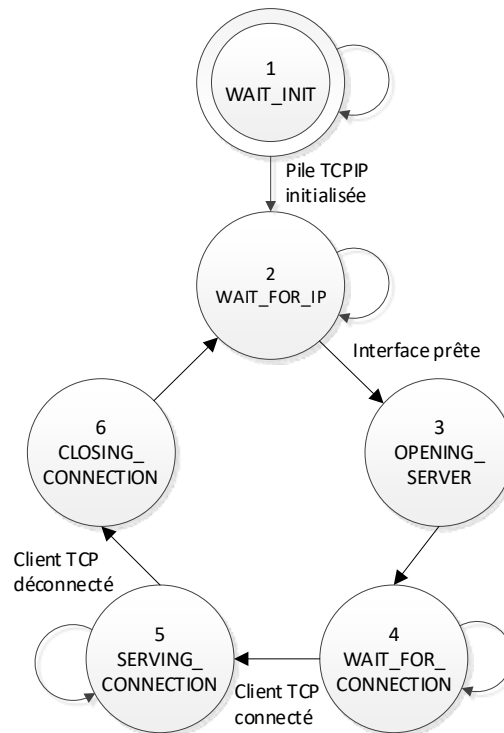
Les principales parties à adapter sont :

- Affichage de la nouvelle adresse IP à chaque changement.  
Le §10.2.3.1.4 donne les principes de détection du changement et d'obtention de la nouvelle adresse IP. Dans le cas d'une application à plusieurs machines d'état, il est toutefois plus propre de centraliser l'affichage (idéalement dans `menugen.c`).
- Détection de l'état du lien TCP au lieu de la connexion USB.
- Réception et envoi de données via TCP au lieu d'USB.

Ces 2 derniers points sont détaillés ci-dessous.

### 10.2.5. DÉTAILS DU FONCTIONNEMENT DE L'EXEMPLE TCPIP\_TCP\_SERVER

Le projet exemple tcpip\_tcp\_server peut être décrit selon la machine d'état suivante :



Les états ont des noms explicites. Voici une description des états permettant les interactions ou la communication :

2. Dans cet état, une nouvelle adresse IP obtenue pourra être détectée (voir §10.2.3.1.4).
3. Un serveur TCP (socket) est ouvert. Le programme est ensuite disponible pour qu'un client se connecte.
4. Attente d'une connexion TCP de la part d'un client.
5. Un client est connecté. Les transferts de données peuvent être réalisés dans cet état.
6. Fermeture du serveur TCP.

### 10.2.5.1. DÉTECTION ÉTAT DU LIEN TCP

Les fonctions suivantes peuvent être utiles pour connaître l'état d'une liaison réseau :

- **TCPIP\_STACK\_NetIsLinked()**  
Permet de connaître l'état physique (connecté/déconnecté) de la liaison Ethernet. Ne donne que l'état de la couche TCPIP d'accès au réseau et ne renseigne pas sur la connexion d'un client TCP au serveur.
- **TCPIP\_TCP\_WasReset()**  
Indique si le lien TCP (socket) a été déconnecté depuis le dernier appel.
- **TCPIP\_TCP\_IsConnected()**  
Renvoie l'état du lien TCP.

Les 2 dernières fonctions sont d'une même utilité. La dernière (TCPIP\_TCP\_IsConnected) sera préférée car elle est déjà utilisée dans l'exemple en question.

Toutefois des modifications supplémentaires sont nécessaires afin de pouvoir détecter une fermeture non propre de la connexion (de la part du client) ou encore une interruption du réseau. Il faut forcer le socket TCP serveur à créer un échange de données régulier avec le client, même en cas d'absence de données à transférer (« keepAlive »). Ce mécanisme permet de vérifier que le client est toujours joignable, et donc que le lien TCP est actif.

Cela peut être fait simplement en réglant les propriétés du socket TCP à la suite de son ouverture dans l'état « OPENING\_SERVER » (ajouts en gras) :

```
case APP_TCPIP_OPENING_SERVER:
{
    SYS_CONSOLE_PRINT("Waiting for Client Connection on port:
        %d\r\n", SERVER_PORT);
    appData.socket = TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE_IPV4,
        SERVER_PORT, 0);
    if (appData.socket == INVALID_SOCKET)
    {
        SYS_CONSOLE_MESSAGE("Couldn't open server socket\r\n");
        break;
    }

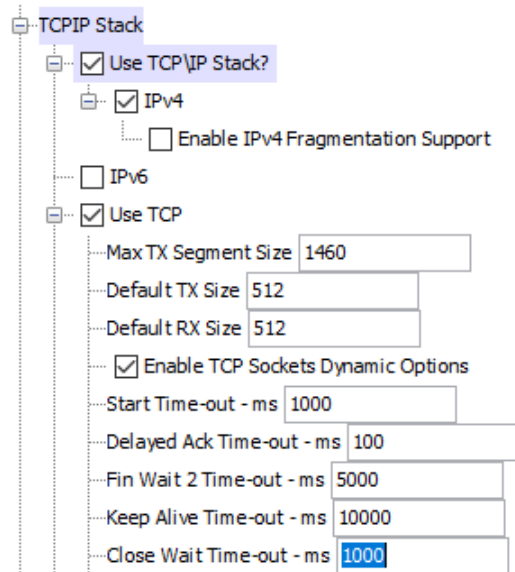
    //SCA set keepalive
    // Nécessaire si on veut que TCPIP_TCP_IsConnected() détecte
    // déconnexion du câble
    appData.keepAlive.keepAliveEnable = true;
    appData.keepAlive.keepAliveTmo = 1000;
    // [ms] / 0 => valeur par défaut
    appData.keepAlive.keepAliveUnackLim = 2;
    // [nb de tentatives] / 0 => valeur par défaut
    TCPIP_TCP_OptionsSet(appData.socket, TCP_OPTION_KEEP_ALIVE,
        &(appData.keepAlive));

    appData.state = APP_TCPIP_WAIT_FOR_CONNECTION;
}
break;
```

La déclaration du champ de structure appData.keepAlive est nécessaire au préalable.



Finalement, il faut régler la propriété TCP « Close Wait Time-Out » dans le MHC à une valeur différente de zéro. Par exemple 1000 ms :



Avec le mécanisme et les réglages décrits ci-dessus, la machine d'état de l'exemple sortira automatiquement de l'état « SERVING\_CONNECTION » en cas de déconnexion TCP, quelle qu'en soit la cause (déconnexion physique ou logicielle, propre ou non).

Le lecteur est invité à se reporter à l'aide de Harmony pour davantage d'informations.

### 10.2.5.2. RECEPTION ET ENVOI DE DONNEES TCP

Dans l'état « SERVING\_CONNECTION » les fonctions suivantes peuvent être utilisées pour communiquer via TCP :

#### **TCPIP\_TCP\_GetIsReady (appData.socket)**

- Renvoie le nombre d'octets qui peuvent être lus (qui ont été reçus).
- Paramètre : socket. A laisser tel quel.

#### **TCPIP\_TCP\_ArrayGet (appData.socket, buffer, len)**

- Lit des octets reçus via TCP (les enlève du buffer de réception TCP et les copie dans buffer).
- Paramètre 1 : socket. A laisser tel quel.
- Paramètre 2 : buffer où vont être copiées les données.
- Paramètre 3 : nombre d'octets à lire.
- Retourne le nombre d'octets lus.

#### **TCPIP\_TCP\_PutIsReady (appData.socket)**

- Renvoie le nombre d'octets qui peuvent être envoyés (la place libre dans le buffer).
- Paramètre : socket. A laisser tel quel.

#### **TCPIP\_TCP\_ArrayPut (appData.socket, buffer, len)**

- Envoie des données via un socket TCP. Les données sont copiées dans le buffer d'envoi TCP ; l'envoi à proprement parler sera géré par la pile TCPIP.
- Paramètre 1 : socket. A laisser tel quel.
- Paramètre 2 : buffer où sont stockées les données à envoyer.
- Paramètre 3 : nombre d'octets à envoyer.
- Retourne le nombre d'octets effectivement copiés dans le buffer d'envoi.

### 10.2.5.3. CONCLUSION AU SUJET DE LA MODIFICATION DE L'EXEMPLE

Cette partie a montré comment modifier l'exemple Microchip en lui l'intégrant une autre application sous la forme d'une deuxième machine d'état. La manière de communiquer via TCP a ensuite été abordée.

## 10.3. REALISATION D'UN WEBSERVER

Cette partie a pour objectif de découvrir comment modifier l'exemple Microchip `web_server_nvm_mpfs` pour aboutir au pilotage du générateur de signal à partir de sa page web. Cet exemple de base présente un serveur web dont les pages sont stockées dans la flash interne du PIC32.

### 10.3.1. UTILISATION D'UN EXEMPLE ET PORTAGE SUR LE KIT

Pour découvrir la réalisation d'un web server, nous allons utiliser l'exemple qui est sous :  
<Répertoire Harmony>\v<n>\apps\tcpip\**web\_server\_nvm\_mpfs**

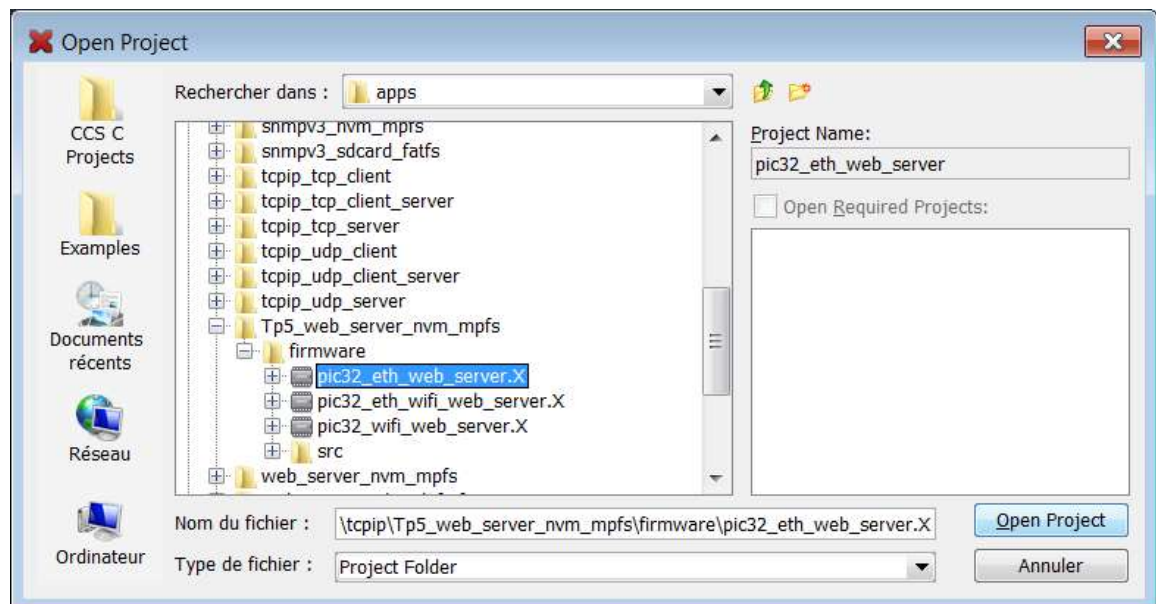
Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire **web\_server\_nvm\_mpfs** au même niveau et nous le renommons **Tp5\_web\_server**.

Le portage sur le kit SKES du projet suivant et son adaptation ont été réalisés avec les logiciels suivants :

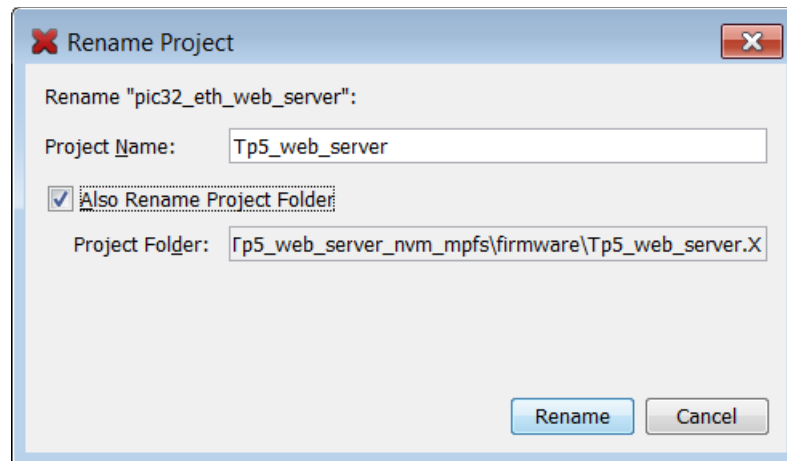
- Harmony v2.05.01
- MPLABX IDE v4.15
- XC32 v2.05

#### 10.3.1.1. OUVERTURE DU PROJET


Le projet contient une triple configuration. Nous choisissons `pic32_eth_web_server.X` comme ci-dessous :

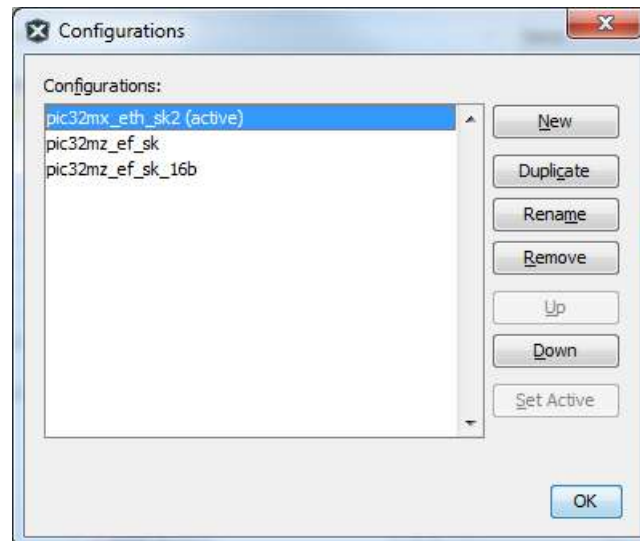


On renomme le projet en Tp5\_web\_server :



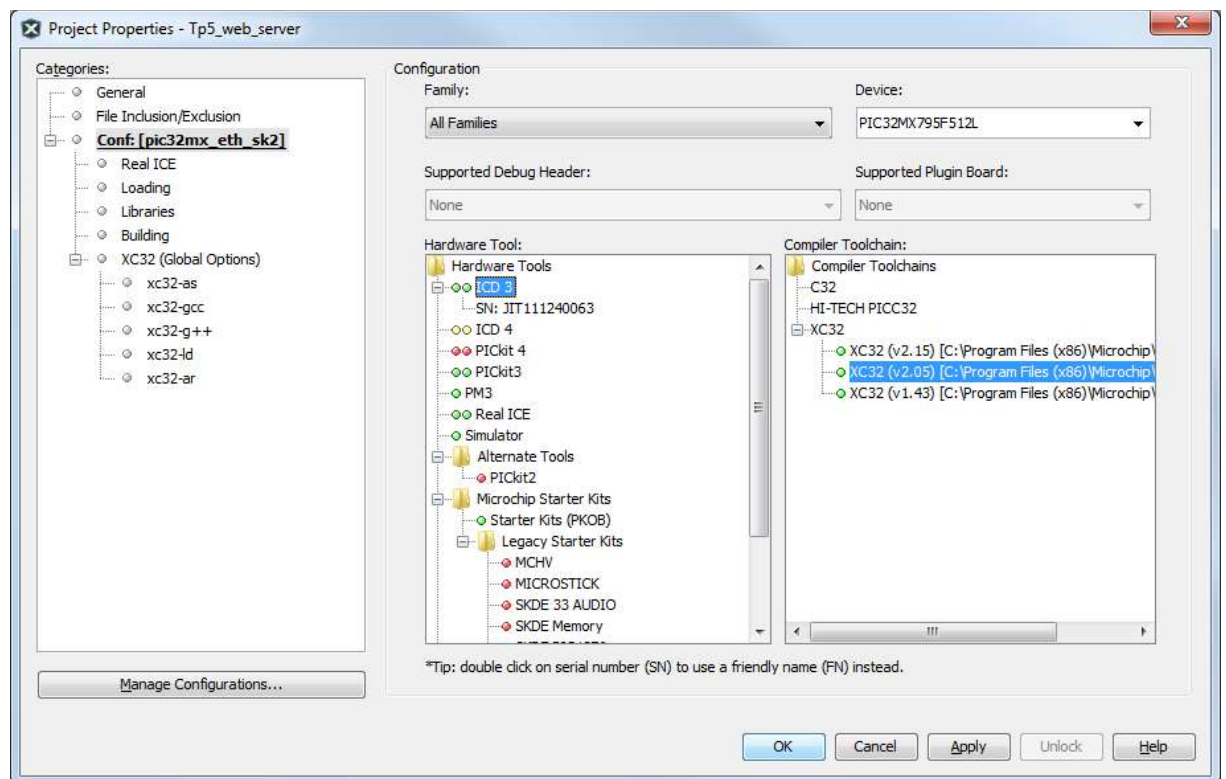
#### 10.3.1.2. REDUCTION DES CONFIGURATIONS

A partir des propriétés du projet, puis  , on obtient :



Nous ne conservons que la première (pic32mx\_eth\_sk2), c'est la plus proche de la configuration de notre kit.

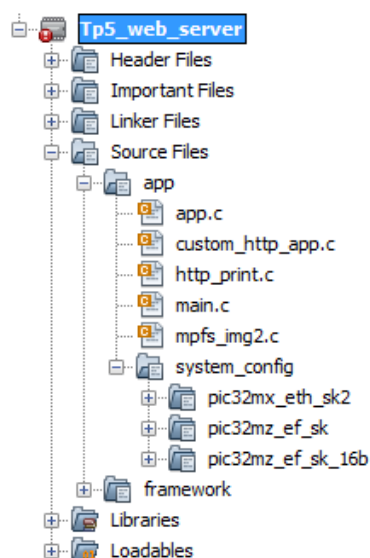
Ce qui nous ramène à une situation plus simple :



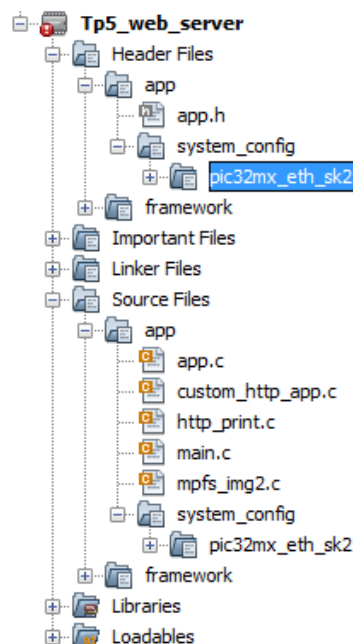
- Au besoin, régler la bonne version du compilateur.
- Ne pas oublier la coche xc32-gcc > preprocessing and messages > additionnal warnings.
- Afin de faciliter le debug, régler le niveau d'optimisation du compilateur à 0.

### 10.3.1.3. REDUCTION DU PROJET

Dans l'arborescence du projet, sous "header files" et "source files", on retrouve les différentes configurations et les bsp.

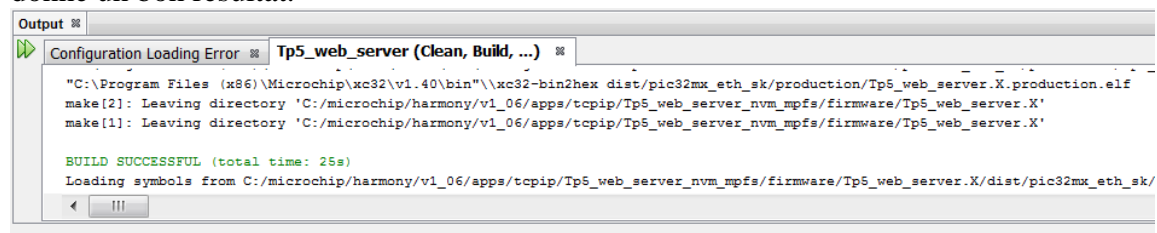


En utilisant **Remove From Project**, on supprime les différents répertoires pour ne garder que "pic32mx\_eth\_sk2" à chaque fois. On aboutit à la situation suivante :



#### 10.3.1.3.1. Vérification si compilation OK

Avant d'effectuer d'autres modifications, il faut vérifier si le résultat du Clean and Build donne un bon résultat.

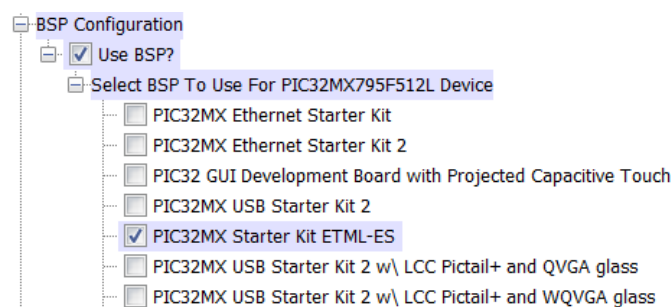


### 10.3.2. MODIFICATION DE LA CONFIGURATION HARMONY

On ouvre le MHC pour adapter la configuration à notre kit (principalement : choix du BSP, sélection du chip Ethernet physique et ajout d'une application supplémentaire).

#### 10.3.2.1. SELECTION BSP PIC32MX\_SKES

Sélection de ☒ PIC32MX Starter Kit ETML-ES



### 10.3.2.2. AJOUT APPGEN

Ajout d'une 2<sup>ème</sup> application que nous appelons appgen.

Application Configuration

Number of Applications 2

Application 0 Configuration

Application 1 Configuration

☒ Use Application Configuration?

Application Name appgen

Application name must be valid C-language identifiers and should be short and lowercase.

☐ Generate Application Code For Selected Harmony Components

### 10.3.2.3. MODIFICATION DEVICE CONFIGURATION

La configuration ne demande que quelques modifications :

#### 10.3.2.3.1. Modification DEVCFG3

Il faut modifier FETHIO de OFF à ON pour utiliser le port Ethernet par défaut (cette configuration sélectionne le jeu de pins RMII connectées au chip physique externe).

DEVCFG3

User ID (USERID) 0xffff

SRS Select (FSRSEL) PRIORITY\_7

Ethernet RMII/MII Enable (FMIIEN) OFF

Ethernet I/O Pin Select (FETHIO) ON

CAN I/O Pin Select (FCANIO) OFF

USB USID Selection (FUSBIDIO) OFF

USB VBUS ON Selection (FVBUSONIO) OFF

#### 10.3.2.3.2. Modification DEVCFG1

Désactiver l'oscillateur secondaire (FSOSCEN à OFF au lieu de ON).

DEVCFG1

Oscillator Selection Bits (FNOOSC) PRIPLL

Secondary Oscillator Enable (FSOSCEN) OFF

Internal/External Switch Over (IESO) ON

Primary Oscillator Configuration (POSCMOD) XT

CLKO Output Signal Active on the OSCO Pin (OSCIOFNC) OFF

Peripheral Clock Divisor (FPBDIV) DIV\_1

Clock Switching and Monitor Selection (FCKSM) CSECME

Watchdog Timer Postscaler (WDTPS) PS1048576

Watchdog Timer Enable (FWDTEN) OFF

#### 10.3.2.3.3. Modification DEVCFG0

Pour DEBUG ON au lieu de OFF.

DEVCFG0

Background Debugger Enable (DEBUG) ON

ICE/ICD Comm Channel Select (ICESEL) ICS\_PGx2

Program Flash Write Protect (PWP) OFF

Boot Flash Write Protect bit (BWP) OFF

Code Protect (CP) OFF

#### 10.3.2.4. SELECTION DU CHIP ETHERNET PHYSIQUE

Avec la configuration choisie (la plus proche de notre kit), le projet exemple se base sur l'utilisation d'un chip physique Ethernet qui est différent du nôtre.

Dans Harmony Framework Configuration > Drivers > Internal Ethernet MAC Driver > External PHY Configuration, sélectionner le bon chip (National\_ DP83848) :

External PHY Configuration

Delay for the Link Initialization - ms 500

PHY Address 1

External PHY Connection Flags

External PHY Type National\_DP83848

PHY Instances Number 1

PHY Clients Number 1

PHY Peripheral Index Number 1

PHY Peripheral ID 1

PHY Negotiation Time-out - ms 1

PHY Negotiation Done Time-out - ms 2000

PHY Reset Clear Time-out - ms 500

☐ Use a Function to be called at PHY Reset



### 10.3.2.5. TIMERS

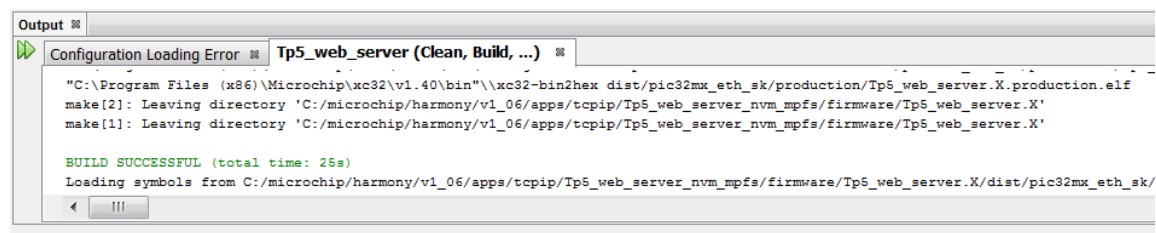
L'exemple de Microchip inclut un driver de timer dynamique. Ce driver utilise déjà le timer 2.

Pour notre générateur de fonctions, nous utilisons les timers 1 et 3 en mode statique. Il nous faut des timings inférieurs à la milliseconde et aussi précis que possible. Un driver dynamique ne convient pas, et il n'est pas possible de mixer les types statique et dynamique.

La configuration de l'exemple ne convient donc pas au générateur de fonctions. Nous conservons la configuration du timer de l'exemple. Les 2 drivers de timers seront ajoutés manuellement au projet par la suite.

### 10.3.2.6. TEST COMPILATION

Génération du code, puis test de compilation.



```
Output
Configuration Loading Error  Tp5_web_server (Clean, Build, ...)
"C:\Program Files (x86)\Microchip\xc32\v1.40\bin"\xc32-bin2hex dist/pic32mx_eth_sk/production/Tp5_web_server.X.production.elf
make[2]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'
make[1]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'

BUILD SUCCESSFUL (total time: 25s)
Loading symbols from C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X/dist/pic32mx_eth_sk/
```

### 10.3.2.7. ETAT APRES MODIFICATIONS

A ce stade, le projet a été porté; il doit être compilable et fonctionnel sur le kit SKES. Moyennant que l'on connaisse l'adresse IP attribuée au kit, on devrait pouvoir tester l'accès à sa page web. L'obtention de l'adresse IP est détaillée ci-après.

Remarquons qu'une grande partie de la flash interne est utilisée par les fichiers du serveur web et d'autres fonctionnalités qui ne sont pas forcément utiles.

### 10.3.2.8. REDUCTION DE LA FLASH NECESSAIRE

Cette étape n'est pas obligatoire. Elle permet de réduire la flash nécessaire au projet, par exemple pour utilisation sur un autre modèle de PIC32. On peut enlever différentes fonctionnalités que l'on n'utilise pas. Dans MHC, enlever les coches :

- Sous Harmony Framework Configuration → System Services :
  - Command
  - console (l'exemple inclut une ligne de commande accessible via USB)
  - debug
- Sous Harmony Framework Configuration → TCPIP Stack :
  - FTP (les fichiers du serveur web sont également accessibles via FTP, mais de manière limitée dans la flash interne.
  - Telnet Server
  - TCPIP commands (le fait d'enlever cette coche enlève également la coche USB Stack)
- Sous Harmony Framework Configuration:
  - USB Stack

Ces modifications demandent d'enlever du code les parties qui utilisent les fonctionnalités enlevées. Dans dans app.c, enlever :

- Les appels à SYS\_CONSOLE\_PRINT() et SYS\_CONSOLE\_MESSAGE()
- Les appels à SYS\_CMD\_READY\_TO\_READ()
- Les variables netName et netBiosName (déclaration et affectations)

### 10.3.2.9. TEST INITIAL

Plusieurs modifications peuvent être utiles afin d'utiliser par exemple les LED ou encore l'affichage LCD afin d'y afficher l'adresse IP.

- Configuration du BSP
  - Se reporter au §10.2.3.1.1
- Ajout dans appgen.c
  - Se reporter au §10.2.3.1.210.2.3.1.1
- Affichage erreur init stack, modif app.c
  - Se reporter au §10.2.3.1.3

### 10.3.2.9.1. Affichage adresse IP

On peut également afficher l'adresse IP. Il est possible de se reporter au §10.2.3.1.4 qui détaille un fonctionnement basique.

Une version plus élaborée, qui tient compte de la déconnexion de l'interface réseau (câble Ethernet), est détaillée ci-dessous.

Dans le case APP\_TCPIP\_TRANSACT, on peut afficher l'adresse IP à chaque changement au niveau de la boucle for :

```
// if the IP address of an interface has changed
// display the new value on the system console
nNets = TCPIP_STACK_NumberOfNetworksGet();

for(i = 0; i < nNets; i++)
{
    netH = TCPIP_STACK_IndexToNet(i);
    ipAddr.Val = TCPIP_STACK_NetAddress(netH);

    //ajout SCA pour obtenir nouvel affichage à chaque
    changement
    netIsLinked[1] = netIsLinked[0];    //décalage états
    //lecture nouvel état
    netIsLinked[0] = TCPIP_STACK_NetIsLinked(netH);

    if(dwLastIP[i].Val != ipAddr.Val || (netIsLinked[1] !=
        netIsLinked[0])) //IP ou état du lien changé ?
    {
        dwLastIP[i].Val = ipAddr.Val;
        if (!netIsLinked[0]) //interface déconnectée ?
        {
            ipAddr.Val = 0;
        }
        //SYS_CONSOLE_PRINT("%s   IP   Address:   %d.%d.%d.%d
                               \r\n",
        //          TCPIP_STACK_NetNameGet(netH),
        //          ipAddr.v[0], ipAddr.v[1], ipAddr.v[2],
        //          ipAddr.v[3]);
        lcd_gotoxy(1,4);
        printf_lcd("IP:%03d.%03d.%03d.%03d  ", ipAddr.v[0],
            ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
    }
}
```

Cela nécessite la déclaration suivante préalable :

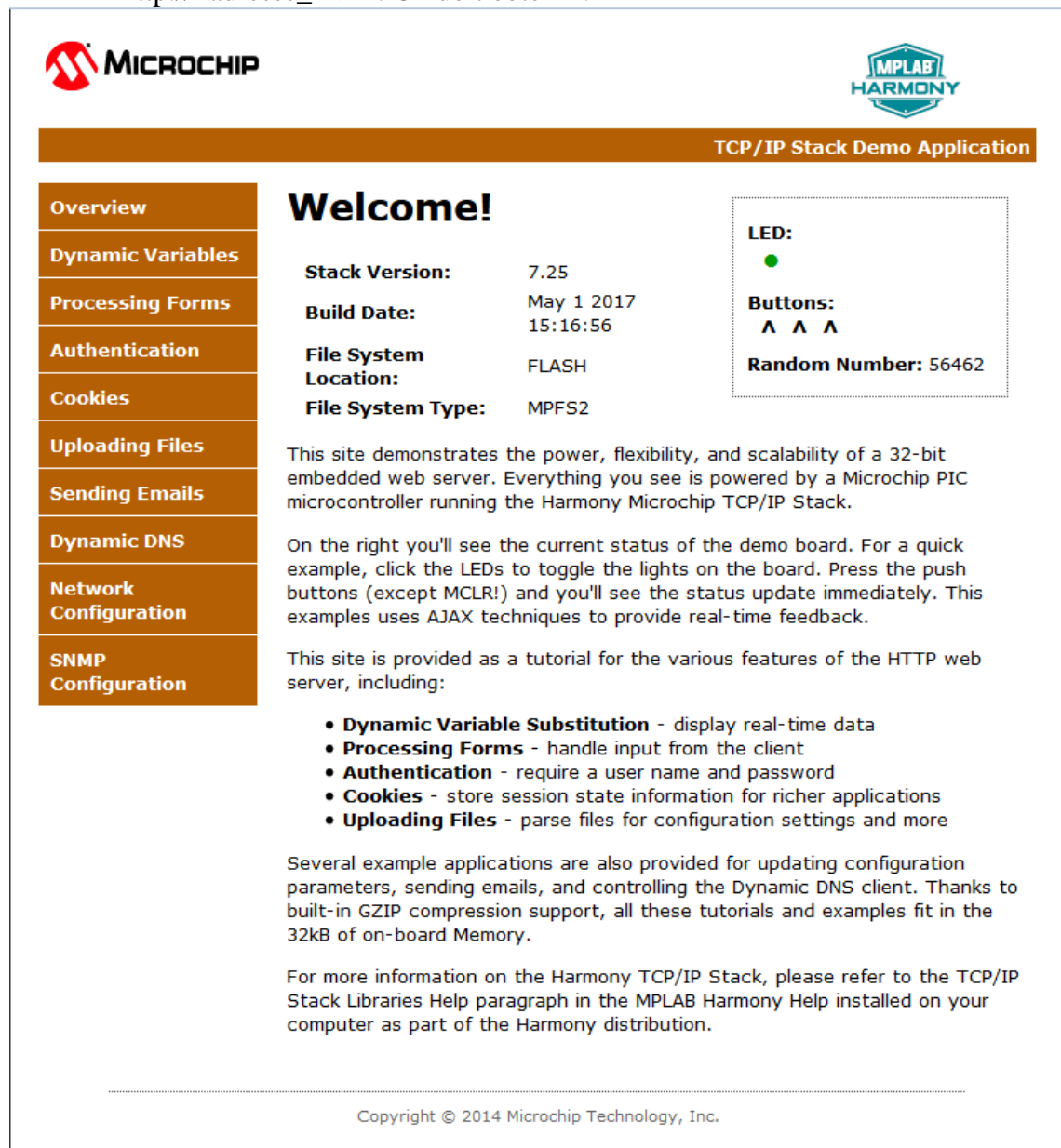
```
//SCA pr détection changement adr. IP
static bool          netIsLinked[2] = {false, false};
```

La variable netIsLinked[0] peut être utilisée pour connaître l'état du réseau.

### 10.3.2.9.2. Résultat

Ce premier test permet de vérifier si la base fonctionne et de s'assurer que l'introduction du BSP ne perturbe pas les actions :

- Connecter le kit au réseau d'expérimentation.
- Le message d'annonce, puis l'adresse IP doivent s'afficher.  
Remarque : Les adresses IP 0.0.0.0, ou encore celle statique par défaut 192.168.100.115 peuvent apparaître avant la valeur effective.
- Si l'adresse IP est affichée et semble plausible, ouvrir un navigateur à l'adresse « http://<adresse\_IP> ». On doit obtenir :



Les touches du kit doivent modifier l'affichage sous "Buttons".

### 10.3.2.10. CONCLUSION AU SUJET DU PORTAGE DE L'EXEMPLE

Cette partie a montré comment porter l'exemple Microchip sur le kit SKES. La partie suivante concerne l'intégration d'une application à l'exemple et les interactions avec le webserver.

### 10.3.3. INTEGRATION DU GENERATEUR

Il s'agit maintenant d'intégrer les fichiers du générateur et de faire évoluer l'application. Il est encore nécessaire d'ajouter manuellement les timers 1 et 3.

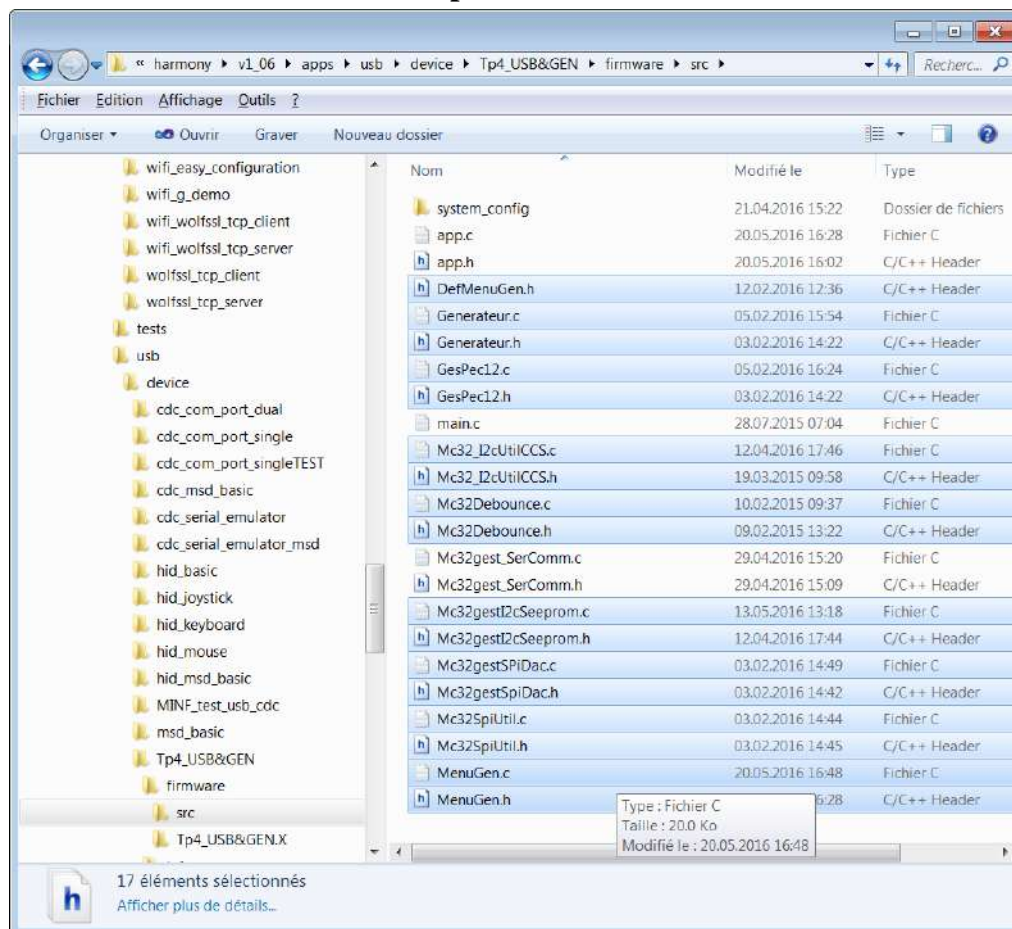
Dans cet exemple, nous reprenons les fichiers issus de l'adaptation à l'USB.

#### 10.3.3.1. MISE EN PLACE DES FICHIERS DU GENERATEUR

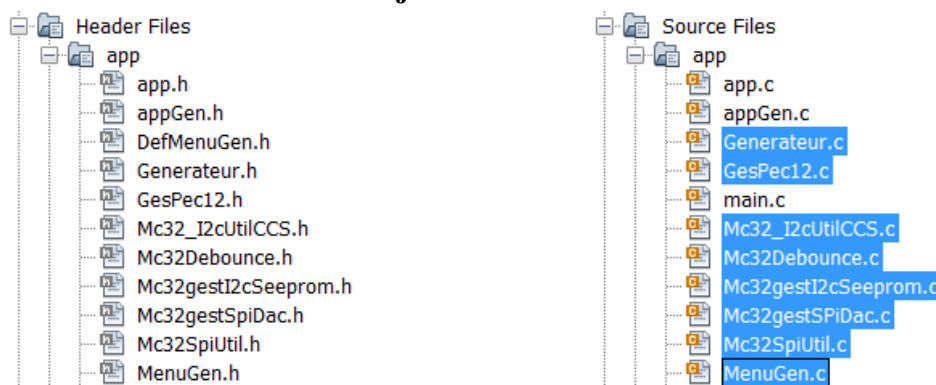
Il faut copier presque tous les fichiers source du TP générateur USB dans le répertoire du projet actuel :

<Répertoire Harmony>\v<n>\apps\tcpip\Tp5\_web\_server\_nvm\_mpf\firmware\src

##### 10.3.3.1.1. Liste des fichiers à copier



##### 10.3.3.1.2. Résultat de l'ajout dans les header et sources



### 10.3.3.2. ADAPTATION DE SYSTEM\_INIT.C

L'exemple fourni n'a pas de driver statique pour les timers.

#### 10.3.3.2.1. Récupération des fonctions driver timer static

Nous allons ici copier les fichiers des drivers statiques des timers créés dans un autre projet.

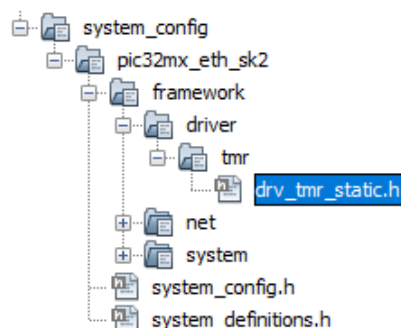
Il suffit de copier le répertoire **tmr** depuis :

<Répertoire Harmony>\v<n>\apps\usb\device\Tp4\_UsbGen\firmware\src\  
system\_config\pic32mx\_usb\_sk2\_int\_sta\framework\driver  
sous :

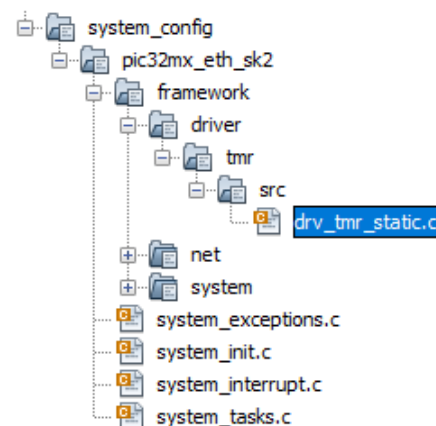
<Répertoire Harmony>\v<n>\apps\tcpip\Tp5\_web\_server\_nvm\_mpf\firmware\src\  
system\_config\pic32mx\_eth\_sk2\framework\driver qui est vide.

Ajouter les fichiers .h et .c .dans le projet en recréant les logical folder :

Dans les header :



Dans les sources :



#### 10.3.3.2.2. Ajout des appels init des drivers statiques

Il faut ajouter les 2 appels des init. des driver timer static dans la fonction SYS\_Initialize.

```
/* Board Support Package Initialization */
BSP_Initialize();

/* Initialize Drivers */
// CHR ajout init drivers timers statiques
DRV_TMR0_Initialize();
DRV_TMR1_Initialize();

sysObj.drvTmr0 = DRV_TMR_Initialize(DRV_TMR_INDEX_0,
                                     (SYS_MODULE_INIT *) &drvTmr0InitData);
```

#### 10.3.3.2.3. Ajout include driver static dans system\_definitions.h

Il est nécessaire d'ajouter le #include suivant dans le fichier **system\_definitions.h** pour pouvoir compiler system\_init.c.

```
#include "app.h"
#include "appgen.h"
#include "driver/tmr/drv_tmr_static.h" //driver timer static
```

#### 10.3.3.3. ADAPTATION DE SYSTEM\_INTERRUPT.C

Il s'agit d'ajouter dans le fichier system\_interrupt.c les réponses aux interruptions des timers 1 & 3 que l'on récupère du projet du Tp4\_UsbGen.

Il faut ouvrir les 2 fichiers dans MPLAB X et effectuer la copie.

☛ Il faut modifier le nom d'une ISR pour éviter un doublon !

```
// Int Timer1 cycle 1ms pour antirebond
// Activation application tous les 10 cycles
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0(void)
```

A renommer en :

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerTimer1(void)
```

#### 10.3.3.3.1. Ajout dans appgen.h

Pour arriver à compiler le fichier system\_interrupt.c, il est nécessaire d'ajouter des #include dans appgen.h, ainsi que des définitions.

##### 10.3.3.3.1.1. Ajouts #include

```
// Specifique menu et generateur
#include "DefMenuGen.h"
#include "Mc32Debounce.h"
#include "GesPec12.h"
#include "Generateur.h"
```

**10.3.3.4. ADAPTATION DE APPGEN.C****10.3.3.4.1. Partie APPGEN\_STATE\_INIT**

Cette partie contient l'initialisation des différents éléments du générateur. Elle est reprise telle quelle du générateur projet du générateur USB :

```
case APPGEN_STATE_INIT:
{
    //init. lcd
    lcd_init();
    lcd_bl_on();

    SPI_InitLTC2604(); // Init SPI DAC

    I2C_InitMCP79411(); //init. comme I2C avec EEPROM
                        externe I2C

    Pec12Init(); // Initialisation du PEC12

    // Initialisation du menu et rapatrie valeurs signal de
    l'EEPROM externe
    MENU_Initialize(&LocalParamGen);

    //par défaut (tant que rien reçu), signal USB = signal
    local
    RemoteParamGen = LocalParamGen;

    // Initialisation du generateur
    GENSIG_Initialize(&LocalParamGen);

    // Active les timers
    DRV_TMR0_Start();
    DRV_TMR1_Start();

    appgenData.state = APPGEN_STATE_WAIT;

    break;
}
```



#### 10.3.3.4.2. Partie APPGEN\_STATE\_SERVICE\_TASKS

Dans cette partie, on enlève tout ce qui touche à la gestion de l'USB. On se retrouve alors avec une gestion simple du menu :

```
case APPGEN_STATE_SERVICE_TASKS:
    //affichage menu
    if (MENU_Execute(&LocalParamGen, !APP_NetIsLinked()))
    {
        //mise à jour signal si le menu le demande
        //met à jour fréquence
        GENSIG_UpdatePeriode(&LocalParamGen);
        //met à jour échantillons
        GENSIG_UpdateSignal(&LocalParamGen);
    }
    appgenData.state = APPGEN_STATE_WAIT;
    break;
```

Par rapport au fonctionnement en USB, il y a eu une simplification et on n'utilise plus qu'un seul jeu de paramètres.

La fonction APP\_NetIsLinked() a dû être créée en remplacement de APP\_USBIConnected() qui était valable pour le générateur de fonctions USB. Cette nouvelle fonction indique l'état de la connexion ethernet pour affichage du menu local ou remote. Voir § 10.2.3.1.4 "Affichage adresse IP" pour lecture de l'état du réseau.

#### 10.3.3.5. TEST DU GENERATEUR

A ce stade, la compilation doit être OK.

Sans le câble réseau, on obtient le réglage du générateur et son action.

Lorsque le câble réseau est branché, on obtient l'affichage avec le # et les actions sur le Pec12 sont sans effet.

La page web n'a pas changé et n'a pas d'interaction avec le générateur de fonctions.

### 10.3.4. PAGE WEB PERSONNALISEE

Ce qui suit reprend les explications issues du webinaire de Microchip en 3 parties :

1. "TCP/IP Networking: Web-Based Status Monitoring"  
document Microchip en543032
2. "TCP/IP Networking, Part 2: Web-Based Control"  
document Microchip en543035
3. "TCP/IP Networking, Part 3: Advanced Web-Based Control"  
document Microchip en543038

Cette documentation est disponible sous :

...\Maitres-Eleves\SLO\Modules\SL229\_MINF\TP\CoursLabo

#### 10.3.4.1. COPIE DU REPERTOIRE WEB\_PAGES

Il faut tout d'abord supprimer le répertoire web\_pages sous

<Répertoire Harmony>\v<n>\apps\tcpip\Tp5\_web\_server\_nvm\_mpfs\firmware\src

Et le remplacer par celui qui est fourni sous :

...\Maitres-Eleves\SLO\Modules\SL229\_MINF\TP\TP5\_WebGen.

#### 10.3.4.2. GENERATION DES FICHIERS

Il faut exécuter l'utilitaire "Microchip MPFS Generator" dont l'exécutable est le fichier **mpfs2.jar** qui se trouve sous :

<Répertoire Harmony>\v<n>\utilities\mpfs\_generator

Cet utilitaire sert à analyser le code HTML de notre page web et à y repérer les variables à afficher. Ces variables doivent être présentées entre tildes ('~').

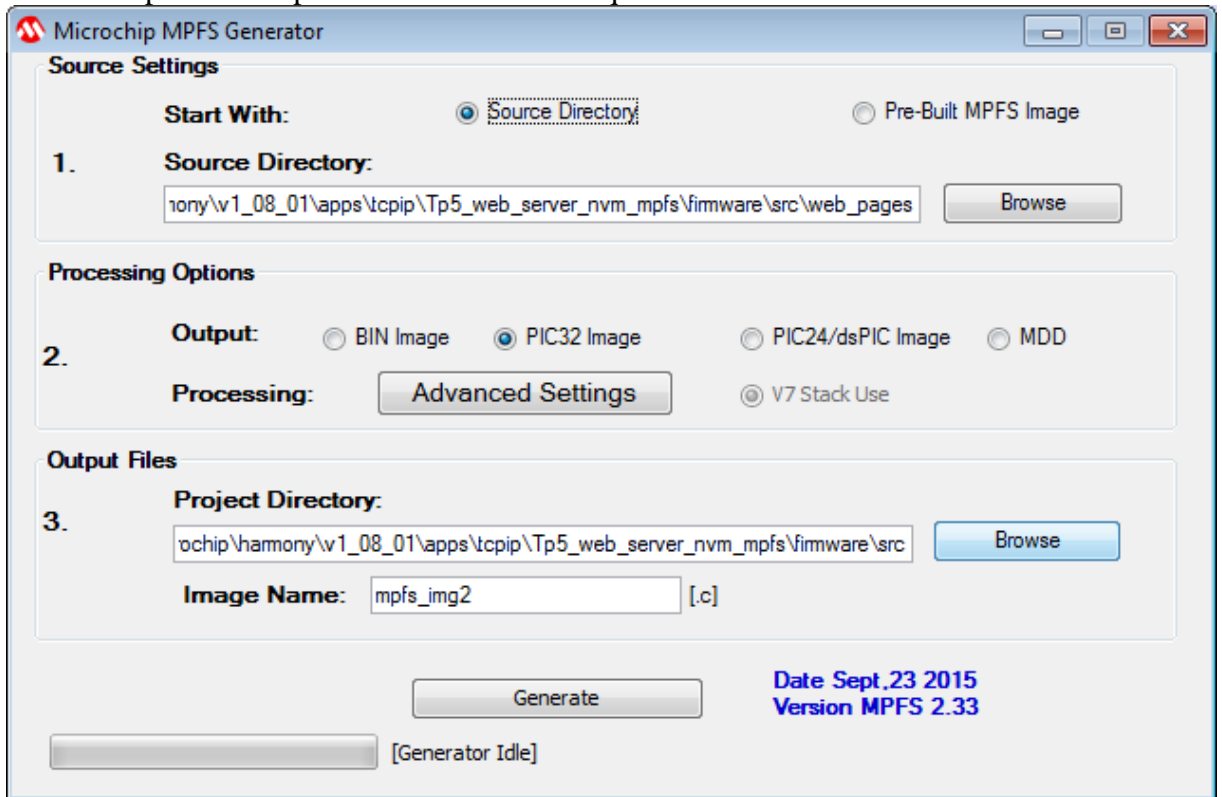
Exemple d'extrait de code HTML :

Valeur de la variable : ~variable~

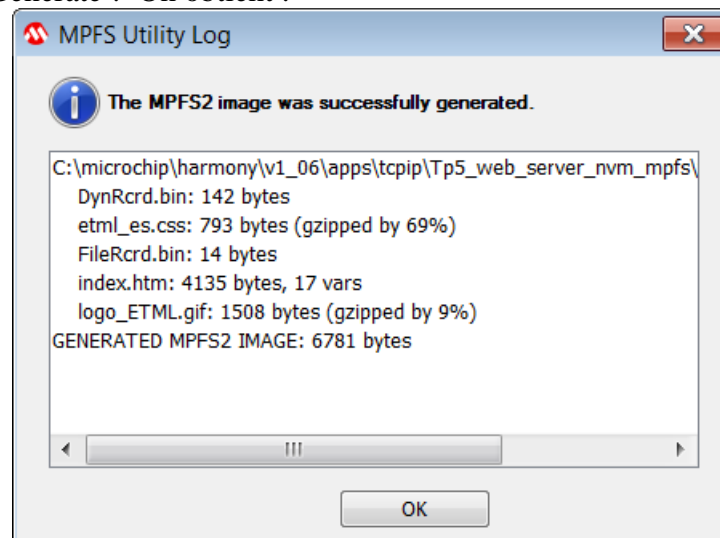
Régler les éléments suivants :

1. Chemin d'accès vers les fichiers web du projet
2. Sélectionner PIC32 image

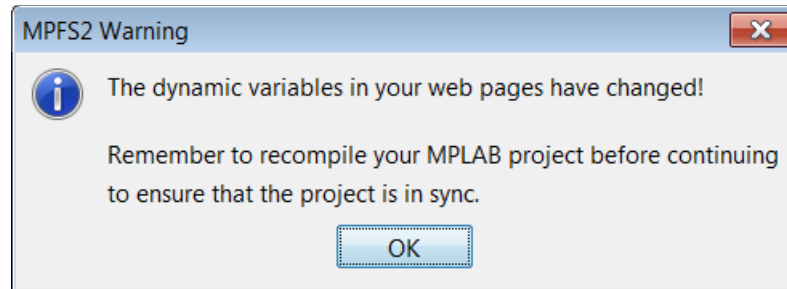
Les autres paramètres peuvent être laissés tels quels :



Et finalement Generate ! On obtient :



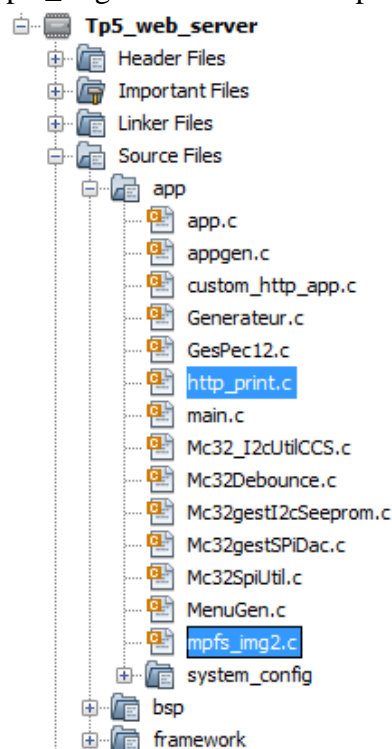
Puis :



Le message signale que des variables dynamiques ont changé. Il y a donc des adaptations de code à effectuer (ci-dessous).

#### 10.3.4.3. FICHIERS MODIFIES

Les fichiers `http_print.c` et `mpfs_img.c` ont été modifiés par l'utilitaire :



👉 Ils ne doivent pas être modifiés manuellement !

##### 10.3.4.3.1. Contenu `mpfs_img.c`

Ce fichier contient, sous la forme d'un tableau constant, le système de fichiers pour la flash interne du microcontrôleur incluant les fichiers web du serveur.

##### 10.3.4.3.2. Contenu `http_print.c`

Pour chaque variable à afficher présente dans le code HTML de la page web, un appel à une fonction d'affichage correspondante a été généré :

```
#include "tcpip/tcpip.h"

void TCPIP_HTTP_Print(HTTP_CONN_HANDLE connHandle, uint32_t callbackID);
void TCPIP_HTTP_Print_shapeSelected(HTTP_CONN_HANDLE connHandle, uint16_t);
void TCPIP_HTTP_Print_freq(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_ampl(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_offset(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_dutyCycle(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_version(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_builddate(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_drive(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTP_Print_fstype(HTTP_CONN_HANDLE connHandle);
```

Remarque : cela correspond aux prototypes des fonctions qu'il faut implémenter dans le fichier custom\_http\_app.c.

#### 10.3.4.4. ADAPTATIONS DU FICHIER CUSTOM\_HTTP\_APP.C

Après un clean and build, on obtient les erreurs suivantes :

```
"C:\Program Files (x86)\Microchip\xc32\v1.40\bin\xc32-gcc.exe" -mprocessor=32MX795F512L -o dist/pic32mx_eth_sk/production/Tp5_web_server.X.production
build/pic32mx_eth_sk/production/_ext/1360937237/http_print.o: In function 'TCPIP_HTTP_Print':
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:60: undefined reference to 'TCPIP_HTTP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:63: undefined reference to 'TCPIP_HTTP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:66: undefined reference to 'TCPIP_HTTP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:69: undefined reference to 'TCPIP_HTTP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:72: undefined reference to 'TCPIP_HTTP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:75: undefined reference to 'TCPIP_HTTP_Print_freq'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:78: undefined reference to 'TCPIP_HTTP_Print_ampl'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:81: undefined reference to 'TCPIP_HTTP_Print_offset'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server_nvm_mpfs/firmware/src/http_print.c:84: undefined reference to 'TCPIP_HTTP_Print_dutyCycle'
collect2.exe: error: ld returned 255 exit status
make[2]: *** [dist/pic32mx_eth_sk/production/Tp5_web_server.X.production.hex] Error 255
```

Il est nécessaire d'implémenter les fonctions dans le fichier custom\_http\_app.c

##### 10.3.4.4.1. Ajout des fonctions dans custom\_http\_app.c

Il y a 4 fonctions à ajouter :

- TCPIP\_HTTP\_Print\_shapeSelected()
- TCPIP\_HTTP\_Print\_freq()
- TCPIP\_HTTP\_Print\_ampl()
- TCPIP\_HTTP\_Print\_offset()

Cela correspond aux nouvelles 4 variables de la page web pour lesquelles aucune fonction d'affichage n'existe encore. Il suffit de copier les prototypes et d'ajouter { }.



Il faut encore corriger la fonction TCPIP\_HTTP\_Print\_shapeSelected.

```
//-----  
// ajout SCA : fonctions pour affichage variables dynamiques TP5 WebGen  
  
// nécessaire de nommer le 2ème paramètre  
void TCPIP_HTTP_Print_shapeSelected(HTTP_CONN_HANDLE connHandle, uint16_t shapeNb)  
{  
}  
  
void TCPIP_HTTP_Print_freq(HTTP_CONN_HANDLE connHandle)  
{  
}  
  
void TCPIP_HTTP_Print_ampl(HTTP_CONN_HANDLE connHandle)  
{  
}  
  
void TCPIP_HTTP_Print_offset(HTTP_CONN_HANDLE connHandle)  
{  
}  
  
#endif // #if defined(TCPIP_STACK_USE_HTTP_SERVER)
```

☞ Ce sont les fonctions pour afficher les variables du PIC sur la page Web. Nous les implémenterons par la suite.

#### 10.3.4.4.2. Observation de la page Web

A ce stade, une compilation ne doit pas générer d'erreur. On peut tester le programme et l'accès à la page web.

Malgré les fonctions vides, on obtient la page spécifique du générateur. On peut modifier les paramètres, mais cela n'a pas d'effet ; il manque les mécanismes de transmission de valeurs.

#### 10.3.4.4.3. Affichage des variables du PIC sur la page web

Selon le § 10.3.4.2 "Génération des fichiers" :

- Les variables que l'on désire afficher sur la page web ont été placées entre tildes dans le code html.
- Sur la base des variables trouvées, l'utilitaire Microchip MPFS Generator a généré :
  - Un fichier mpfs\_img2.c contenant les fichiers web sous la forme d'une grande constante C.
  - Un prototype de fonction pour l'affichage de chaque variable dans http\_print.c.

Il est finalement nécessaire d'implémenter le retour des valeurs des variables. Les fonctions sont à implémenter dans custom\_http\_app.c.

##### 10.3.4.4.3.1. Fonction d'affichage d'une variable

En s'inspirant des fonctions TCPIP\_HTTP\_Print\_...() déjà présentes, on peut facilement créer sur le même modèles celles qui permettront d'afficher nos variables. Par exemple, pour l'affichage de l'offset, cela donne :

```
void TCPIP_HTTP_Print_offset(HTTP_CONN_HANDLE connHandle)
{
    char str[6];
    TCP_SOCKET sktHTTP =
        TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

    //conversion entier->ascii
    itoa(str, LocalParamGen.Offset, 10);

    //envoi chaîne
    TCPIP_TCP_StringPut(sktHTTP, (uint8_t*)str);
}
```

Remarque :

L'utilitaire Microchip MPFS Generator génère les prototypes de fonctions nécessaires dans http\_pprint.c. Il ne modifie pas custom\_http\_app.c. Ce dernier fichier peut donc contenir des fonctionnalités qui étaient présentes dans l'exemple de page web de Microchip, mais qui ne sont plus nécessaires dans notre cas (par exemple : fonctions TCPIP\_HTTP\_Print\_...() d'affichages de variables qui n'existent pas sur notre page).

#### 10.3.4.4.4. Réception des variables de la page web par le PIC

##### 10.3.4.4.4.1. Fonction *GetExecute* dans *custom\_http\_app.c*

La page web fournie est conçue pour transmettre ses valeurs au PIC à travers un mécanisme http de type "GET". Le principe est d'accéder à une page et de transmettre des paramètres dans l'adresse de la page accédée (URL). C'est une méthode simple. Son désavantage est la limitation à environ 100 octets transmis, mais cela est largement suffisant pour nous.

La fonction **GetExecute** contient le traitement correspondant à la page demo de Microchip :

```
HTTP_IO_RESULT TCPIP_HTTP_GetExecute(
                                HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    uint8_t filename[20];
    uint8_t* httpDataBuff;

    // Load the file name
    // Make sure uint8_t filename[] above is large enough
    // for your longest name
    SYS_FS_FileNameGet(TCPIP_HTTP_CurrentConnectionFileGet
                      (connHandle), filename, 20);

    httpDataBuff
        = TCPIP_HTTP_CurrentConnectionDataBufferGet
          (connHandle);

    // If its the forms.htm page
    if(!memcmp(filename, "forms.htm", 9))
    {
        // Seek out each of the four LED strings, and if it
        // exists set the LED states
        ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                                (const uint8_t *)"led2");
        if(ptr)
            BSP_LEDStateSet(APP_TCPIP_LED_3, (*ptr == '1'));
            //LED2_IO = (*ptr == '1');

        ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                                (const uint8_t *)"led1");
        if(ptr)
            BSP_LEDStateSet(APP_TCPIP_LED_2, (*ptr == '1'));
            //LED1_IO = (*ptr == '1');
    }

    else if(!memcmp(filename, "cookies.htm", 11))
    {
        // This is very simple. The names and values we
        // want are already in
        // the data array. We just set the hasArgs value
        // to indicate how many
    }
}
```



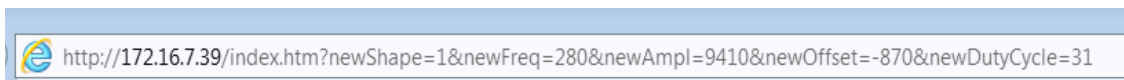
```

        // name/value pairs we want stored as cookies.
        // To add the second cookie, just increment this
        value.
        // remember to also add a dynamic variable callback
        to control the printout.
        TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle,
                                                0x01);
    }
    // If it's the LED updater file
    else if(!memcmp(filename, "leds.cgi", 8))
    {
        // Determine which LED to toggle
        ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                                (const uint8_t *) "led");
        // Toggle the specified LED
        switch(*ptr) {
            case '0':
                BSP_LEDToggle(APP_TCPIP_LED_1);
                //LED0_IO ^= 1;
                break;
            case '1':
                BSP_LEDToggle(APP_TCPIP_LED_2);
                //LED1_IO ^= 1;
                break;
            case '2':
                BSP_LEDToggle(APP_TCPIP_LED_3);
                //LED2_IO ^= 1;
                break;
        }
    }
    return HTTP_IO_DONE;
}

```

#### 10.3.4.4.2. Adaptation de *GetExecute* dans *custom\_http\_app.c*

Nous devons adapter le code à la chaîne reçue (l'URL) :



Pour cela, nous reprenons le principe utilisé dans le code de la demo. Il s'agit de :

- Vérifier que l'accès est fait sur la bonne page web (index.htm).
- Parser l'URL à l'aide de la fonction `TCPIP_HTTP_ArgGet()` pour trouver les valeurs des paramètres.

Le code de la fonction nous donne :

```

HTTP_IO_RESULT TCPIP_HTTP_GetExecute(
                                HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    uint8_t filename[20];

```

```

uint8_t *httpDataBuff;
int32_t tmp;

// Load the file name.
// Make sure uint8_t filename[] above is large enough
// for your longest name.

SYS_FS_FileNameGet(TCPIP_HTTP_CurrentConnectionFileGet(
    connHandle), filename, 20);

httpDataBuff
    = TCPIP_HTTP_CurrentConnectionDataBufferGet
        (connHandle);

//accès à la page index.htm ?
if(!memcmp(filename, "index.htm", 9))
{
    ptr = TCPIP_HTTP_ArgGet
        (httpDataBuff, (uint8_t*)"newShape");

    //récupère forme
    tmp = atoi((char*)ptr);
    LocalParamGen.Forme = tmp -1;

    //récupère fréq.
    ptr = TCPIP_HTTP_ArgGet
        (httpDataBuff, (uint8_t*)"newFreq");
    tmp = atoi((char*)ptr);
    LocalParamGen.Frequence = tmp;

    //récupère ampl.
    ptr = TCPIP_HTTP_ArgGet
        (httpDataBuff, (uint8_t*)"newAmpl");
    tmp = atoi((char*)ptr);
    LocalParamGen.Amplitude = tmp;

    //récupère offset
    ptr = TCPIP_HTTP_ArgGet
        (httpDataBuff, (uint8_t*)"newOffset");
    tmp = atoi((char*)ptr);
    LocalParamGen.Offset = tmp;

    //signale nouvelles valeurs reçues
    APPGEN_NewValuesReceived();
}
return HTTP_IO_DONE;
}

```

Suivant le cas de figure de page web, il pourrait être nécessaire de tester la validité des valeurs reçues ou de les traiter (par exemple : arrondi, limitation, etc).

Ne pas oublier gérer la mise à jour de l'affichage et du signal en cas de réception de nouvelles valeurs.

#### 10.3.4.5. OBSERVATION DE LA PAGE WEB

On obtient l'affichage des valeurs avec quittance et action sur le générateur.



#### 10.3.4.6. TRAITEMENT EN MODE POST

La méthode http GET fait passer directement les données à transmettre dans l'URL, et elle est limitée à environ 100 octets.

Il existe une méthode POST qui palie à ces désavantages. Les données sont alors transmises dans le corps de la requête HTTP et la taille à transmettre est illimitée. Elle peut s'avérer nécessaire, par exemple pour transmettre un fichier d'un navigateur à un serveur, mais est plus compliquée à traiter que le GET..

Cette méthode ne sera pas décrite ici, le lecteur pouvant se référer à la partie 3 des webinaires de Microchip (voir sources au § 10.3.4), ainsi qu'au programme exemple web\_server\_nvm\_mpfs, utilisé dans ce document, qui inclut un exemple de POST.

#### 10.3.4.7. RAFRAICHISSEMENT AUTOMATIQUE DE LA PAGE WEB

Il peut également être nécessaire de rafraîchir la page web automatiquement, sans que l'utilisateur ne doive la recharger. L'exemple de Microchip inclut à cet effet un mécanisme AJAX, dont le principe est le suivant :

1. Le navigateur charge la page web à afficher
2. Cette page inclut du code javascript qui va périodiquement refaire une requête d'un fichier xml au serveur. Ce fichier contient uniquement les valeurs des variables qui doivent être remises à jour sur la page.
3. Le code javascript modifie l'affichage des variables (sans avoir dû retransmettre la page en entier).

Le lecteur est invité à se reporter à l'exemple web\_server\_nvm\_mpfs de Microchip pour en savoir plus.

## 10.4. CONCLUSION

Ce document devrait permettre, en s'inspirant des exemples effectués, de créer d'autres applications supportant TCP/IP.

## 10.5. HISTORIQUE DES VERSIONS

### 10.5.1. VERSION 1.5 MAI 2015

Création du document étape par étape. Version 1.5 pour indiquer l'utilisation de Harmony.

### 10.5.2. VERSION 1.51 JUIN 2015

Ajout de l'intégration du générateur.

### 10.5.3. VERSION 1.6 JUIN 2016

Adaptation au projet sous Harmony 1.06 et Mplabx 3.10.

### 10.5.4. VERSION 1.6 B JUIN 2016

Adaptation de l'intégration du générateur.

### 10.5.5. VERSION 1.7 MAI 2017

Reprise du document par SCA. Relecture. Compléments page web et mise au propre.

### 10.5.6. VERSION 1.71 MAI 2019

Mise à jour pour Harmony 2 (config. du projet exemple pic32mx\_eth\_sk2 remplace pic32mx\_eth\_sk qui n'existe plus, chip physique change).

### 10.5.7. VERSION 1.8 AVRIL 2022

Ajout portage exemple serveur TCP.

### 10.5.1. VERSION 1.81 JUIN 2022

Ajout réglage « close wait timeout » dans le paramétrage du serveur TCP afin de détecter correctement une déconnexion.