STS SE

FreeRTOS

Programmation réseau

WIFI

Programmation réseau

Socket Tcp

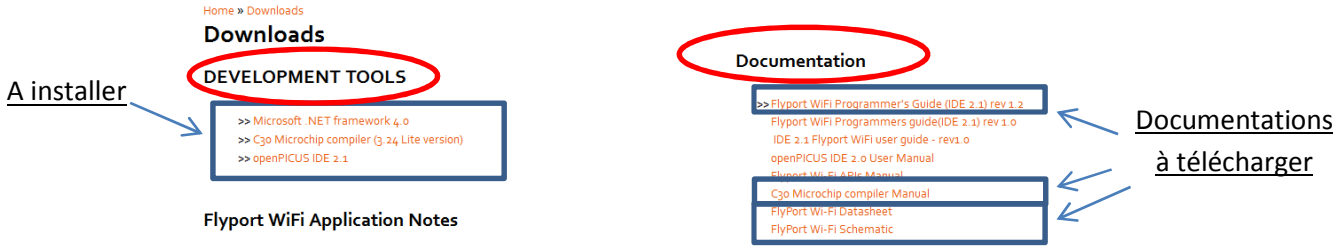
FlyPort smart Wi-Fi 802.11 module

Prérequis : langage C, connaissance réseau : Ip, service Tcp, Udp, client, serveur…

Mise en situation :

OpenPICUS est une plateforme open source, les ressources se trouvent :

- http://www.openpicus.com/site/downloads/downloads



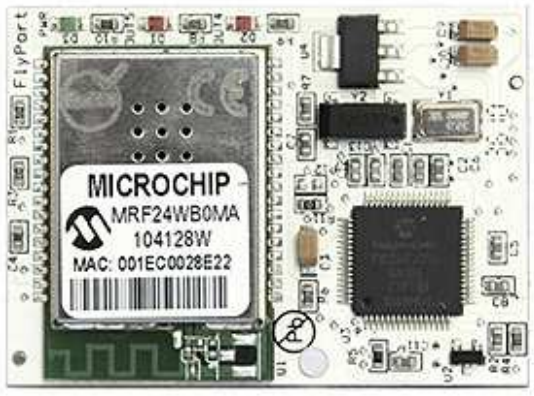
- Cette carte intègre un microcontrôleur **PIC24FJ256** et un module WIFI MRF24WB0MB.

Le développement est basé sur le système d'exploitation freeRTOS.

Il est possible d'acheter cette carte chez Lextronic :

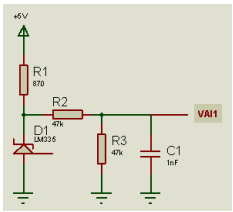
- http://www.lextronic.fr/R2943-openpicus.html

Afin de faciliter le développement, les éléments suivants sont nécessaires :

Le Module WLAN programmable "FlyPort" avec antenne intégrée.	La Platine d'évaluation "USB NEST".
	

Le micro-ordinateur est muni de Labview avec une interface WiFi et une interface USB en mode RS232.

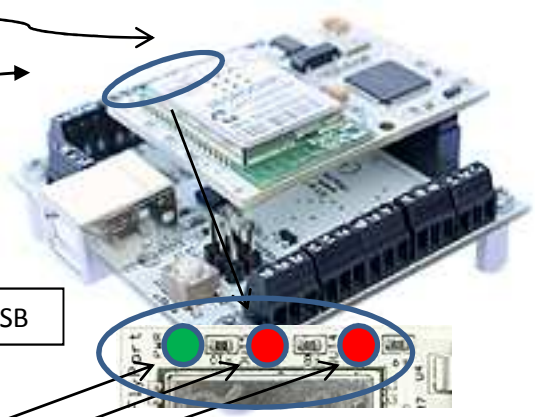
Pc avec logiciel FlyportClientTcpLm335 ou Labview



L'alimentation est prise sur le connecteur en avant.
La sortie température est câblée sur AI1.

Liaison WiFi

Liaison USB



LEDS :

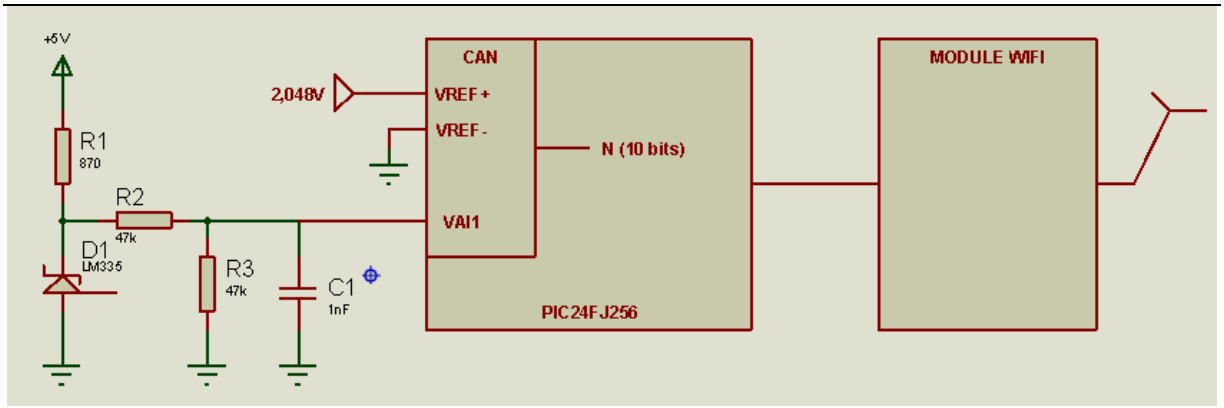
- Carte sous tension.
- OUT5 : WIFI activée.
- OUT4 : disponible.

Objectif : L'objectif est d'afficher sur un micro-ordinateur la température d'un site distant capté à l'aide d'un capteur lm335. La température mesurée pourra varier de -10 à 40°C. La liaison sera en wifi.

- Configuration module Flyport :
  - Le module Flyport est configuré avec les paramètres standards à l'aide du wizard.
- WIFI :
  - modes de déploiement : Ad-HOC (point à point).
  - Sécurité : pas de cryptage.
- IP :
  - Configuration par défaut.
  - Le module Flyport est configuré en serveur DHCP, il fournira ainsi l'adresse au client, qui se connectera.
- TCP :
  - La transmission est effectuée sur le port 2000.
  - Le module Flyport est le serveur, le PC est un client.

La liaison USB permet depuis le PC de programmer le module Flyport et d'envoyer des informations d'état.

Schéma de principe et calcul :



Le CAN est un convertisseur 10 bits

- Il a pour référence une tension de 2,048V. La résolution se calcule : $r = V_{REF+} / 2^{10} = 2$ mV.
- L'étendue de conversion du CAN varie de V_{ref-} à V_{ref+} donc de 0 à 2,048V.

Le capteur LM335 produit une tension de 10mV par Kelvin, soit $V_{LM335} = T_K \times 10$ mV/K.

- La température mesurée peut varier -10 à 40°C, soit de 263K à 313K.
- L'étendue de variation de V_{LM335} est donc de 2,63V à 3,13V. V_{LM335} est toujours supérieure à l'étendue de conversion du CAN, d'où la présence du diviseur par 2.

CAN et Capteur LM335

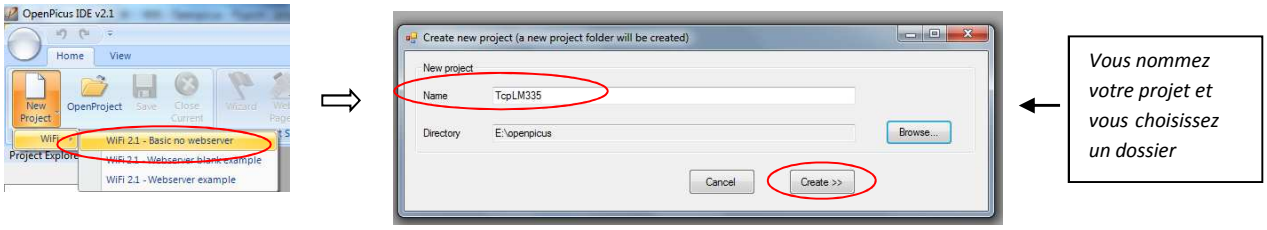
- L'étendue de variation de V_{AI1} est donc de 1,315V à 1,565V.
- Une variation d'un degré correspond à une variation de 5mV en V_{AI1}.
- La résolution du CAN est de 2mV, ce qui correspond à une variation en température de 2/5°C = 0,4°C.
- Le montage permet donc de mesurer la température avec une précision de 0,4°C.

La relation suivante lie le nombre en sortie du CAN à la température mesurée :

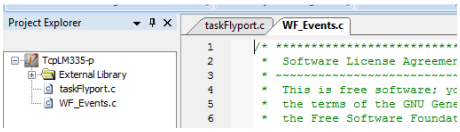
- $N = ((T_K \times 10mV/K) / 2) \times 1024 / 2048 = T_K \times 5 / 2$
- $N = (T_{°c} - 273) \times 5 / 2$

Production du programme :

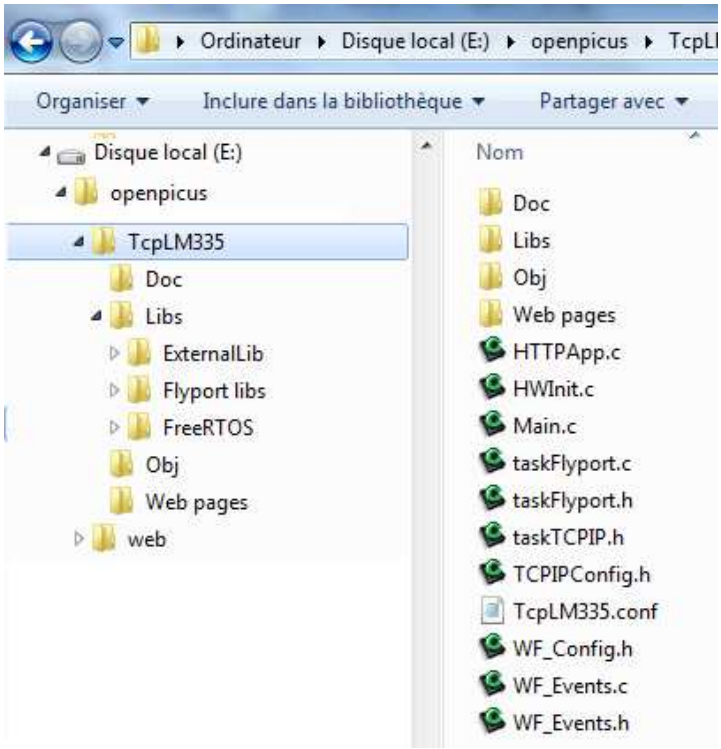
Le projet créé sera le plus simple possible. Vous lancez Openpicus et créez un nouveau projet :



La configuration du projet est la suivante :



Le projet a été créé dans le dossier défini\TcpLM335. Nous trouvons dans ce dossier l'ensemble du projet en langage C avec différentes librairies, dont « FreeRTOS ».



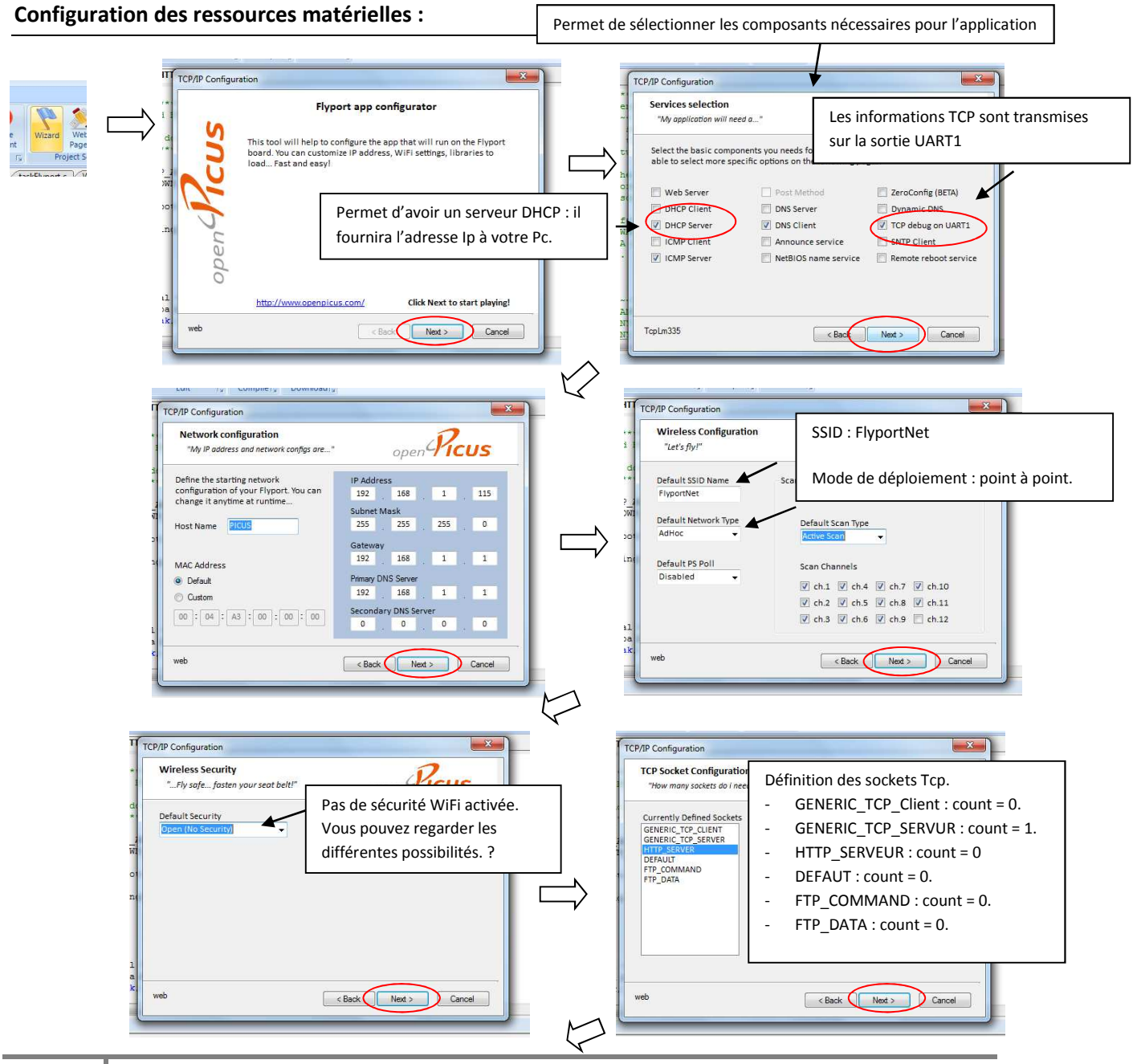
Deux fichiers en langage C sont accessibles et modifiables avec le logiciel Openpicus.
Remarque : dans ce TP nous modifierons uniquement le fichier TaskFlyport.c

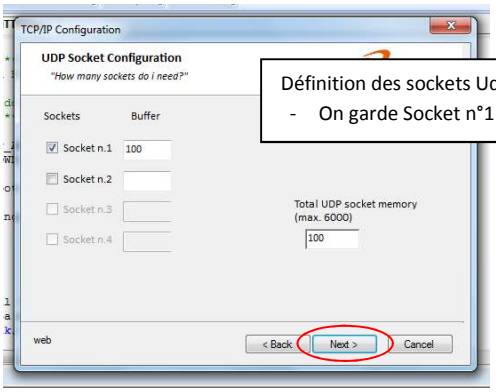
- TaskFlyport.c : contient la fonction FlyportTask().
- WF_events.c : contient les fonctions de gestion événementielle du WiFi.

Le wizard nous permet de produire la configuration réseau de l'OS freeRTOS. Nous allons configurer les services afin de ne valider que les services, qui seront utiles :

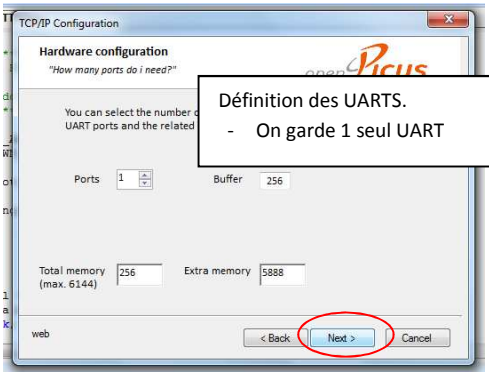
- Sélection des services,
- Configuration Ip,
- Configuration wifi,
- Configuration des sockets,

Configuration des ressources matérielles :

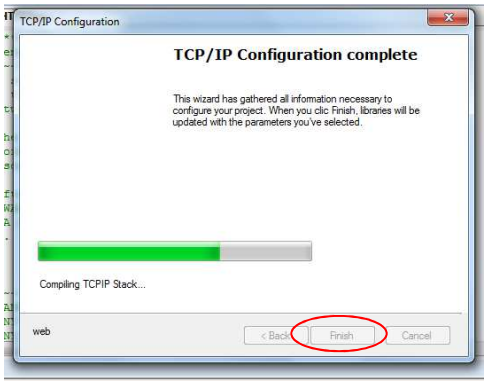




Définition des sockets Udp.
- On garde Socket n°1



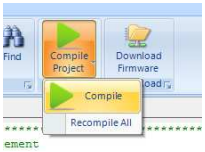
Définition des UARTS.
- On garde 1 seul UART

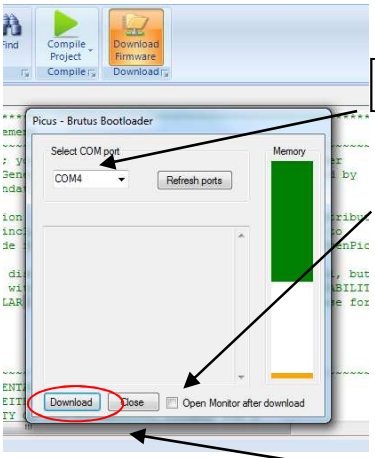


La configuration réseau est terminée. La mémoire est réservée pour les différents sockets programmés.

Vous compilez le programme avant de passer à la suite.

Programmation de la carte « Flyport » :



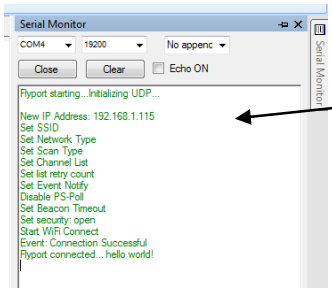


Vous sélectionnez le port com correspondant au « Flyport »

Vous cochez cette case pour voir les données présentes sur la liaison série

Vous lancer la programmation.

A la fin le programme se lance.

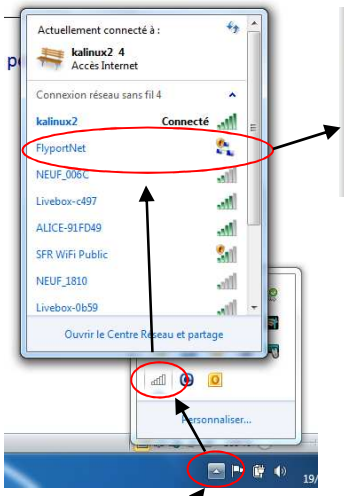


Flyport envoie sur le COM (USB) le résultat de son démarrage.

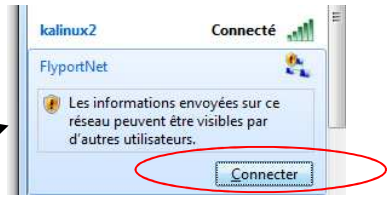
La LED rouge allumée sur la carte Flyport indique que le WiFi est actif.

Test de la configuration wifi, Ip et Tcp.

Connecter en WiFi le PC à la carte Flyport en mode point à point. Sous Windows 7 :



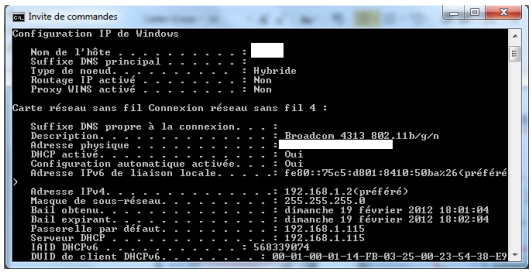
Barre de tache à droite



C'est fait. Le wifi fonctionne.

On teste le protocole Ip.

Vous pouvez vérifier les paramètres Ip de votre poste avec la commande Ipconfig depuis la ligne de commande. Vous pouvez pinguer la carte d'adresse 192.167.1.115.



Les services installés sont actifs et fonctionnels, il faut maintenant écrire le programme, qui va permettre de renvoyer la valeur de la température mesurée sur le port 2000.

Fichier taskFlyport.c

La fonction main() n'est pas directement accessible et est réservée au système d'exploitation RTEOS. Il n'est pas conseillé de la modifier.

La fonction « void FlyportTask() » du fichier « taskFlyport.c » est une des taches du système d'exploitation. Elle est réservée pour écrire notre application.
A droite nous trouvons la version de base produite par l'application Openpicus avec quelques commentaires ajoutés en rouge.

```
#include "taskFlyport.h"

void FlyportTask()
{
        WFConnect(WF_DEFAULT);  // Connexion au profil wifi.
        while (WFStatus != CONNECTED); // Attente connexion effective.
        UARTWrite(1,"Flyport connected... hello world!\r\n"); // envoie message sur UART.
        while(1)
        {
                // Boucle sans fin, le programme de traitement sera écrit ici.
        }
```

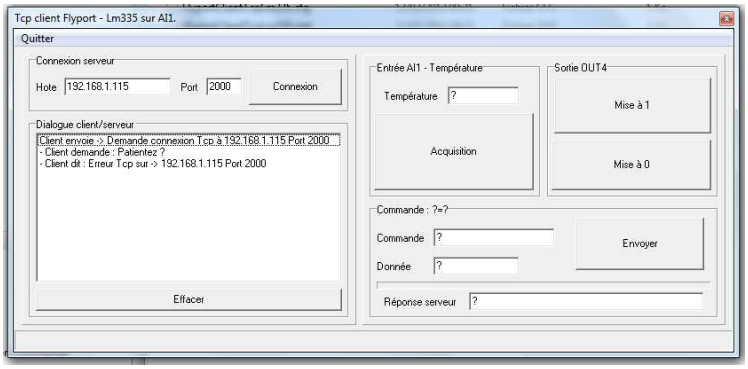
Vous trouvez l'explication des différentes fonctions utilisées au lien :
http://code.google.com/p/openpicus/downloads/detail?name=DOC%20Flyport%20Programmer%27s%20Guide%20Rel%201.2.pdf&can=2&q=

Fichier taskFlyport.c – création d'un serveur, réponse à un client.

Le client « FlyportClientTcpLm335.exe » est disponible en téléchargement. Il est assez simple mais peut présenter quelques erreurs de fonctionnement.

La démarche est la suivante :

- Connecter le Wifi avec windows.
- Définir les paramètres de l'hôte.
- Etablir « Connexion ».
- Lancer commandes avec partie droite.
- Fermer la connexion.



Une commande aura toujours l'écriture suivante : « commande=valeur », par exemple pour :

- Lire la valeur température sur AI4 : Temp= ?
- Mettre la sortie OUT4 à 1 : OUT4=1

La partie en bas à droite permet d'écrire des commandes de manière plus souple.

Le programme du serveur Tcp est donné page suivante.

Vous copiez ce programme dans un projet Openpicus et vérifiez que le dialogue est correct.

En analysant le programme et le résultat obtenu, vous justifiez le rôle de chacune des lignes du programme.

Le programme du serveur Tcp avec lecture de la température.

La fonction « lire_temperature » est fournie :

```
void Tcp_lire_temperature(BYTE TCPSocket)
{
    char AN0String[8];
    unsigned int ADval;
    ADval = ADCVal(1);
    ADval = ADval * 2 / 5 - 273;
    uitoa(ADval, (BYTE *)AN0String);
    myTCPWrite(TCPSocket, AN0String);
    TCPWrite(TCPSocket, "\r\n",2);
}
```

Vous devez modifier le programme précédent pour que le serveur renvoie à chaque requête du serveur la valeur de la température mesurée.

```
#include "taskFlyport.h"
#define myTCPWrite(TCPSocket, cmde) TCPWrite(TCPSocket, cmde,strlen(cmde))
void FlyportTask()
{
	BOOL clconn = FALSE;
	char TCPPort[5]="2000";
	BYTE TCPSocket = INVALID_SOCKET;
	char TCPBuff[31];
	IOInit (o4,out);
	IOPut(o4,off);
	WFConnect(WF_DEFAULT);
	while (WFStatus != CONNECTED);
	UARTWrite(1,"WIfi Flyport connected... hello world!\r\n");
	UARTWrite(1,"Creation socket Tcp : ");
	UARTWrite(1,TCPPort);
	UARTWrite(1,".\r\n");
	TCPSocket = TCPServerOpen(TCPPort);
	UARTWrite(1,"Serveur Socket Tcp ouvert sur port ");
	UARTWrite(1,TCPPort);
	UARTWrite(1,".\r\n");
	while(1)
	{
	if (TCPisConn(TCPSocket)) // Un client est-il connecte ?
	  {
		if (clconn == FALSE) // On teste si le client vient de se connecter.
			{
			clconn = TRUE;  // on mémorise que la connexion est prise en compte
			IOPut(o4,on);  // affiche led IO4 client connecte.
			myTCPWrite(TCPSocket, "Bonjour, bienvenue\r\n"); // renvoie hello au client
			UARTWrite(1,"Client connecte\r\n"); // pour test indique sur rs232.
			}
		else
			{
			 if (TCPRxLen(TCPSocket) > 0) // y a t'il un message du client ?
			    {
			     TCPRead(TCPSocket, TCPBuff,TCPRxLen(TCPSocket));
			     UARTWrite(1, TCPBuff);  // copie message reçu vers RS232 pour info.
			     UARTWrite(1,"\r\n");
			    myTCPWrite(TCPSocket, TCPBuff); // repond au client
			    myTCPWrite(TCPSocket, " \r\n");
			     }
			}
	  }
	  else
	  {
		if (clconn == TRUE) // test fin de connexion client
		  {
			IOPut(o4,off);
			UARTWrite(1,"CLIENT deconnecte\r\n");
			clconn = FALSE;
		  }
	  }
	}
}
```

Voici les fonctions correspondant à l'interpréteur de commande :

```
void Tcp_lire_temperature(BYTE TCPSocket)
{
	char AN0String[8];
	unsigned int ADval;
	ADval = ADCVal(1);
	ADval = ADval * 2 / 5 - 273;
	uitoa(ADval, (BYTE *)AN0String);
	myTCPWrite(TCPSocket, AN0String);
	TCPWrite(TCPSocket, "\r\n",2);
}
void Tcp_ecrire_out4(BYTE TCPSocket,char etat)
{
	 if (etat ==  '1')	IOPut(o4,on);
	  else	IOPut(o4,off);
	 myTCPWrite(TCPSocket,"Ecrire OUT4");
	 TCPWrite(TCPSocket, "\r\n",2);
}
void commande(char *cmde,BYTE TCPSocket)
{
  char *ptr_egal;
  ptr_egal = memchr(cmde, '=', strlen(cmde)); // cherche si caractère egal existe ?
	if (ptr_egal != NULL)  // si non nul on execute la commande
	{
		*ptr_egal = 0;  //TCPBuff contient la commande, ptr_egal+1 contient la commande.
		if (memcmp(cmde,"Temp=?",strlen(cmde))==0) Tcp_lire_temperature(TCPSocket);
		 else if(memcmp(cmde,"OUT4",strlen(cmde))==0) Tcp_ecrire_out4(TCPSocket,*(ptr_egal+1));
		  else
			{
			myTCPWrite(TCPSocket, "Commande '");
			myTCPWrite(TCPSocket, cmde);
			myTCPWrite(TCPSocket," --> ");
			myTCPWrite(TCPSocket,(ptr_egal+1));
			myTCPWrite(TCPSocket, "' inconnue");
			TCPWrite(TCPSocket, "\r\n",2);
			}
	} else myTCPWrite(TCPSocket,"Commande inconnue\r\n");
}
```

Après avoir analysé ces fonctions vous devez les intégrer au programme et valider le fonctionnement.

Ensuite vous devez modifier le programme tel que :

- Supprimer l'écho TCP sur la liaison RS232.
- Les commandes non reconnues sont transmises vers la liaison RS232.