STS SE

FreeRTOS

Programmation réseau

WIFI

Programmation réseau

Socket Tcp

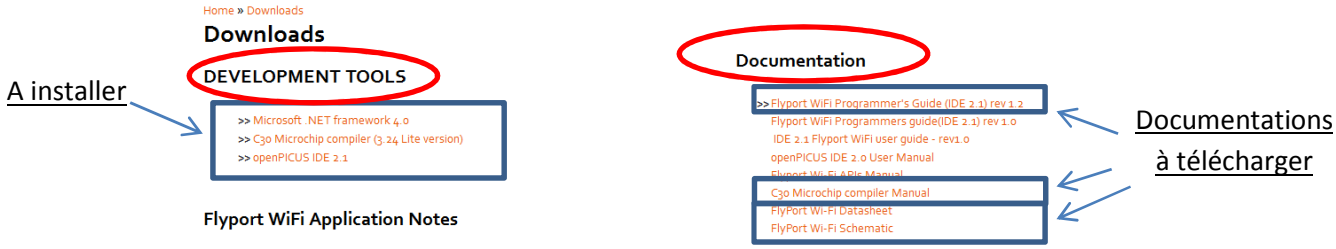
FlyPort smart Wi-Fi 802.11 module

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

Mise en situation :

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

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



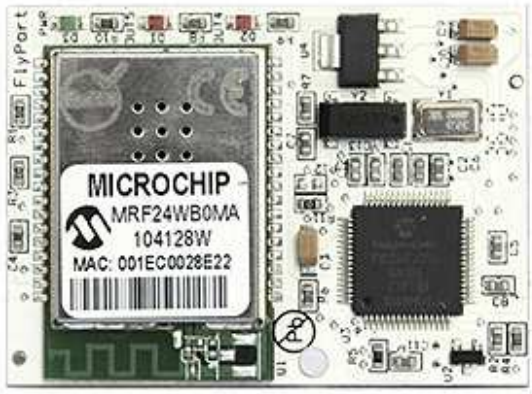
- Cette carte intègre un microcontrôleur **PIC24FJ256** et un module WIFI MRF24WB0MB.

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

Il est possible d'acheter cette carte chez Lextronic :

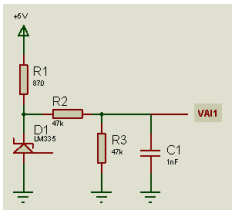
- http://www.lextronic.fr/R2943-openpicus.html

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

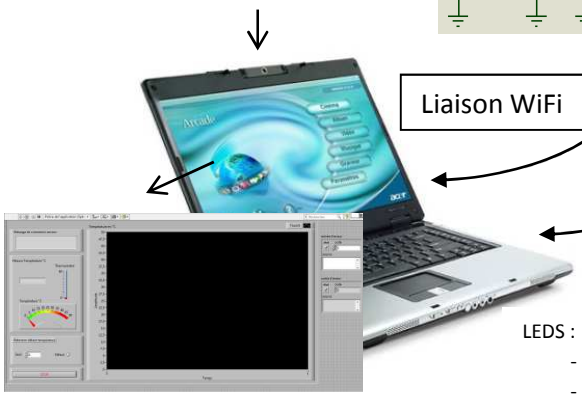
Le Module WLAN programmable "FlyPort" avec antenne intégrée.	La Platine d'évaluation "USB NEST".
	

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

PC avec logiciel FlyportClientTcpLm335 et Labview

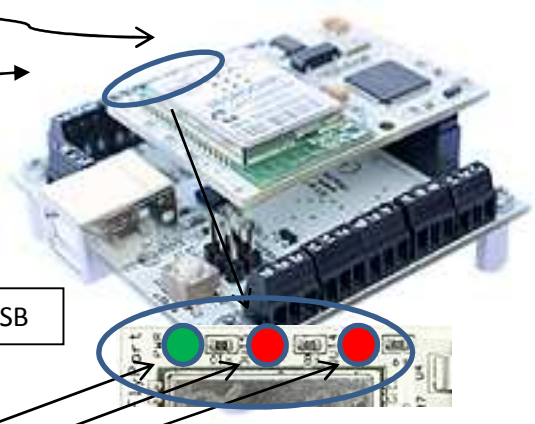


L'alimentation est prise sur le connecteur en avant.
La sortie température est câblée sur AI1.



Liaison WiFi

Liaison USB



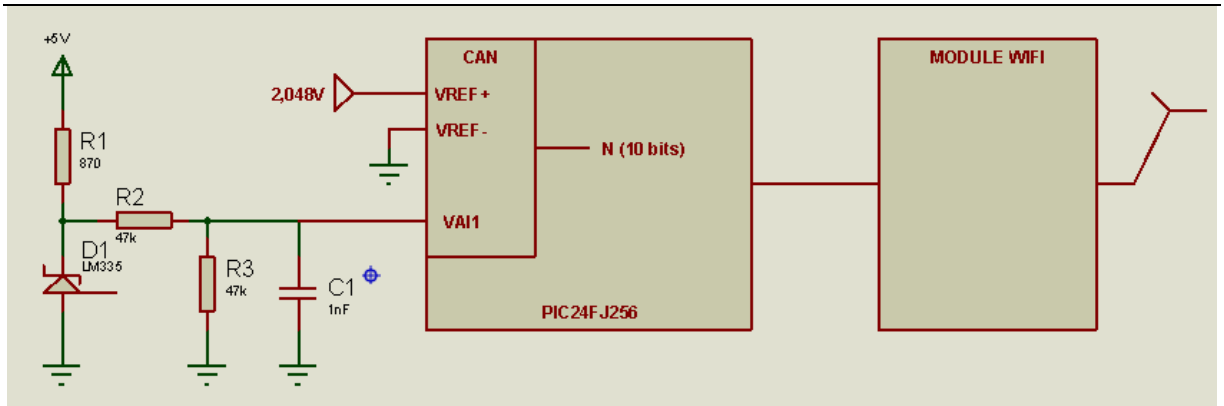
LEDS :

- Carte sous tension.
- OUT5 : WIFI activé.
- 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.
- VI Labview :
  - La face avant est donnée page 5.
  - Il affiche la mesure de température suivant plusieurs outils, ainsi qu'une représentation temporelle.
  - Il indique sur une LED un défaut lorsque la température mesurée est supérieure à la température de seuil.

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 :

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

Programmation du module Flyport :

Vous configurez le logiciel suivant le cahier des charges.
Vous implantez le programme (donné en fin de document : faire un copié/collé dans TaskFlyport.c) et vous validez le fonctionnement à l'aide du logiciel « FlyportClientTcpLm335.exe ».

Remarques :

Les commandes interprétées par le module Flyport sont du type :

- **Commande=donnee** et se terminent par un caractère de valeur 0.

Deux commandes sont implémentées :

- **Temp=?** : renvoie la valeur de la température en °C mesurée sur l'entrée AI1 sous forme de chaine de caractère terminée par CR puis LF.
- **OUT4=X** (X=0 ou 1) : éteint ou allume la LED rouge câblée sur le port OUT4. Il renvoie la chaine de caractère « **Ecrire OUT4** » terminée par CR puis LF.

Labview

Vous utilisez le vi Labview à télécharger sur le site du BTS SE du lycée Victor Hugo. Ce VI est incomplet, il faut le compléter. Il a été écrit en modifiant un exemple de client fourni avec le logiciel Labview. La face avant est donnée page 5.

Remarque : ce VI a été écrit en modifiant un exemple de client fourni avec le logiciel Labview.

Indiquez le port Tcp et l'adresse Ip utilisés pour le dialogue :

- Port = Adresse Ip =

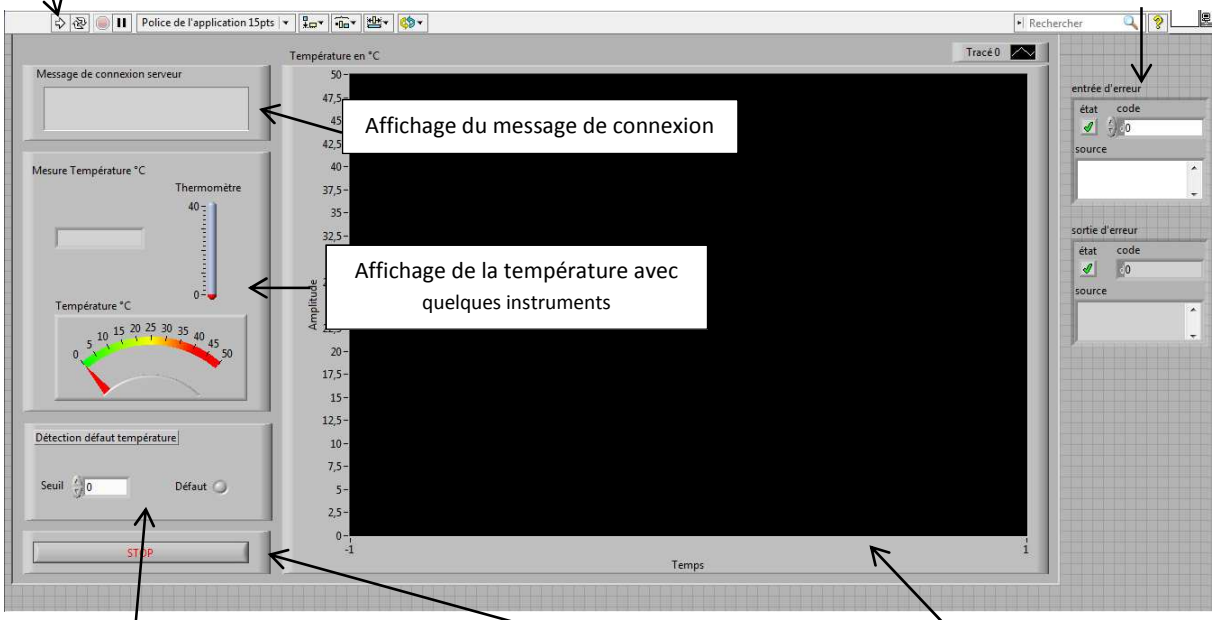
Repérez et complétez la fonction, qui permet d'ouvrir la connexion TCP avec ses paramètres.

Repérez et complétez (entrée mode) les fonctions « Tcp Lire » pour qu'elles soient compatibles avec les chaines de caractères transmises par le module Flyport.

Testez le fonctionnement du modèle obtenu.

On lance l'application Labview

Affichage des erreurs



Affichage du message de connexion

Affichage de la température avec quelques instruments

Programmation d'un seuil de défaut (température mesurée supérieure à température de seuil) et affichage sur la LED du défaut

Pour arrêter l'application

Affichage de la température en fonction du temps

Modifier le VI afin que la LED branchée sur OUT4 s'allume lorsque la température mesurée est supérieure à la température de seuil.

Programme pour Flyport

```
#include "taskFlyport.h"
void commande(char *cmde, TCP_SOCKET TCPSocket);
void Tcp_lire_temperature(TCP_SOCKET TCPSocket);
void Tcp_ecrire_out4(TCP_SOCKET TCPSocket,char etat);
#define myTCPWrite(TCPSocket, cmde) TCPWrite(TCPSocket, cmde,strlen(cmde))
void FlyportTask()
{
	BOOL clconn = FALSE;
	char TCPPort[5]="2000";
	TCP_SOCKET TCPSocket = INVALID_SOCKET;
	char TCPBuff[31];
	ADCInit ( );
	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)) // On fait si un client est connecté.
			{
			 if (clconn == FALSE) // On teste si le client vient de se connecter.
				{		// si le client vient de se connecter, on lui envoie un message.
				clconn = TRUE;  // on mémorise que la connexion est prise en compte
		IOPut(o4,on);			// affiche led IO4 connexion Tcp.
				myTCPWrite(TCPSocket, "Bonjour, bienvenue\r\n"); // renvoie hello au client
				UARTWrite(1,"Client connecte\r\n");	// pour test indique sur rs232.
				}
				else // sinon on vérifie si le client envoie un ordre
				 {
				  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");
					commande(TCPBuff,TCPSocket);
				    }
				 }
			}
			else
			{
			  if (clconn == TRUE) // test fin de connexion client
				{
					IOPut(o4,off);
					UARTWrite(1,"CLIENT deconnecte\r\n");

					clconn = FALSE;
				}
			}
	}
}
```

```
void commande(char *cmde, TCP_SOCKET 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");
}
void Tcp_lire_temperature(TCP_SOCKET TCPSocket)
{
        char AN0String[8];
        int ADval;
        ADval =  ADCVal(1);
        ADval = ADval * 2 / 5 - 273;
        sprintf(AN0String,"%d\r\n",ADval);
        myTCPWrite(TCPSocket, AN0String);
}
void Tcp_ecrire_out4(TCP_SOCKET TCPSocket,char etat)
{
         if (etat ==  '1')          IOPut(o4,on);
          else        IOPut(o4,off);
         myTCPWrite(TCPSocket,"Ecrire OUT4");
         TCPWrite(TCPSocket, "\r\n",2);
}
```