Premiers programmes en C# "étape par étape" avec une carte FEZ T18

[Mise à jour le : 13/2/2020]

Programmes réalisés dans le tuto

- 1. **Blink** - Sortie logique
- 2. **MesureAngle** - Entrée analogique, sortie série asynchrone
- 3. **Ventilation** - Entrée analogique, sorties PWM et série asynchrone

1. Préambule

Pour mener à bien ce tutoriel, vous devez disposer d'une carte Fez T18.

Microsoft **Visual Studio 2017 ou 2019** (Community) doit être installé sur le PC.

Le firmware de la carte doit être à jour. Si ce n'est pas le cas : suivez le "Guide d'installation" ou voir la vidéo "Updating BrainPad's Firmware - Tech Talk 041".

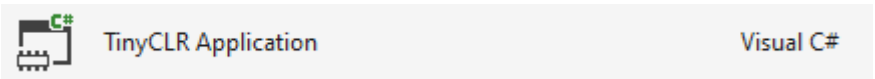
Les vidéos du cours sur les fondamentaux du langage C#, accessibles sur le site MVA sont un excellent préalable ou un complément à ce tutoriel.

2. Premier programme : Blink

Cahier des charges du programme 1 : faire clignoter la LED "**LED1**" de la carte FEZ T18 !

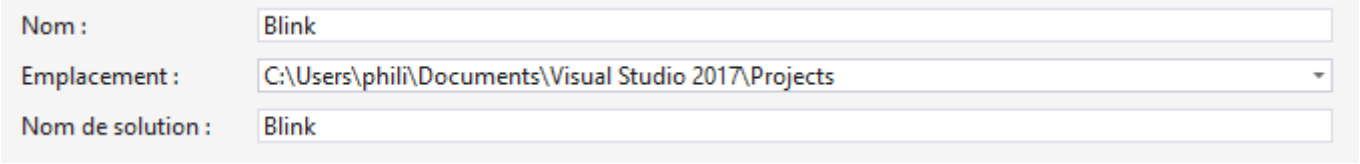
Etape 1 : Créer le projet Blink

- Ouvrir l'**IDE Visual Studio** en cliquant sur l'icône suivante : puis sélectionner : **Fichier → Nouveau projet** ou **[Ctrl+Maj+N]**,
- Dans la boîte de dialogue "Nouveau projet" sélectionner : **Installé → Visual C# → TinyCLR et TinyCLR Application**.



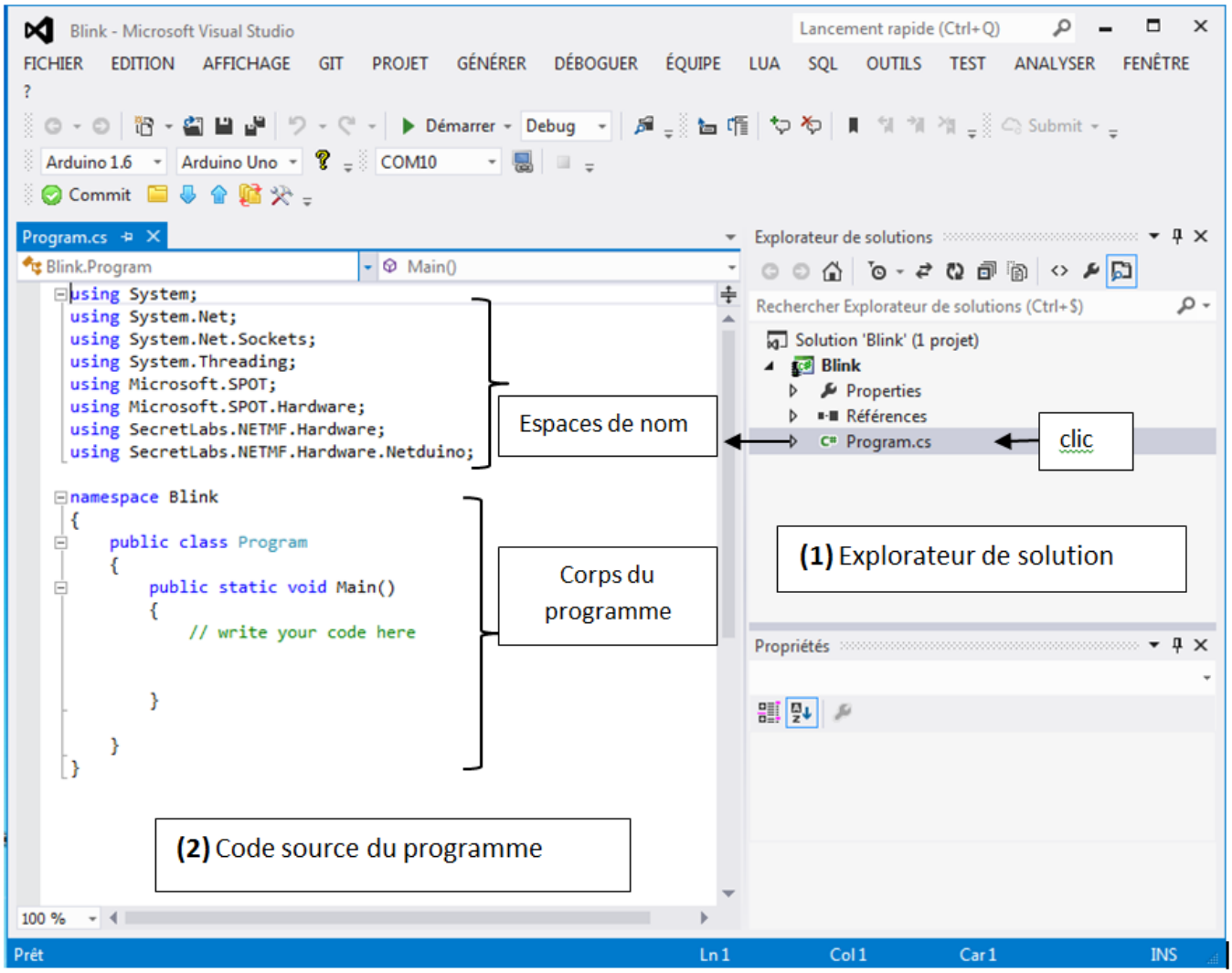
- Donner le nom "**Blink**" à l'application puis cliquer sur **Ok**.

Remarque: L'emplacement identifié ci-dessous peut changer en fonction de la version du logiciel et de l'arborescence des répertoires du PC.



- **Cliquer** sur *Program.cs* dans la fenêtre **Explorateur de solution**

L'IDE est alors configuré comme sur la copie d'écran ci-dessous (ou un équivalent selon sa version) :

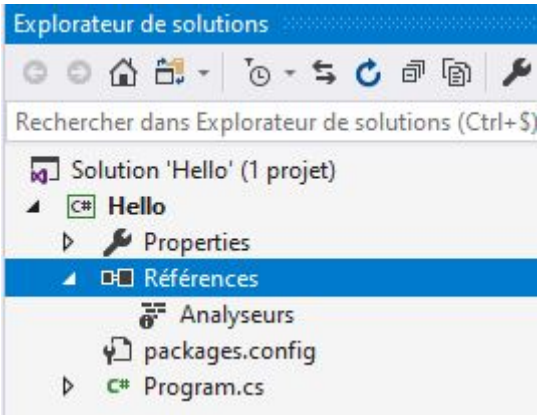


Remarque: Les espaces de nom sont des racourcis. Le projet en contient plus ou moins selon son type.

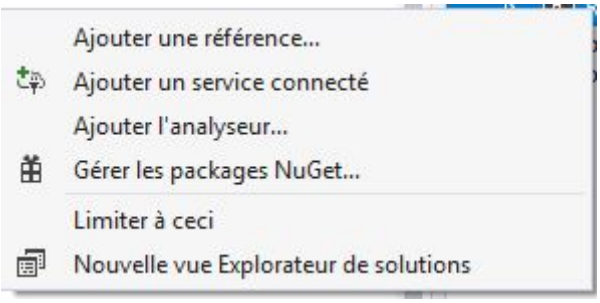
Le projet ***Blink*** est contenu dans la solution ***Blink***. Vous allez écrire le code source du programme dans le fichier **Program.cs**.

Etape 2 : Installer des bibliothèques dans le projet

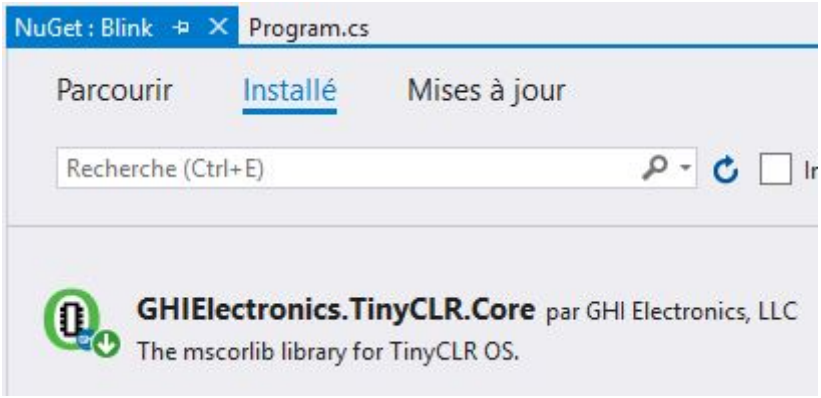
- Effectuer un **clic droit** sur **Références** dans l'explorateur de solution (s'il n'est pas visible : **Affichage → Explorateur de solution**)



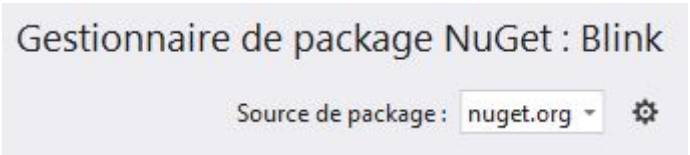
- Cliquer sur **“Gérer les packages NuGet…“**



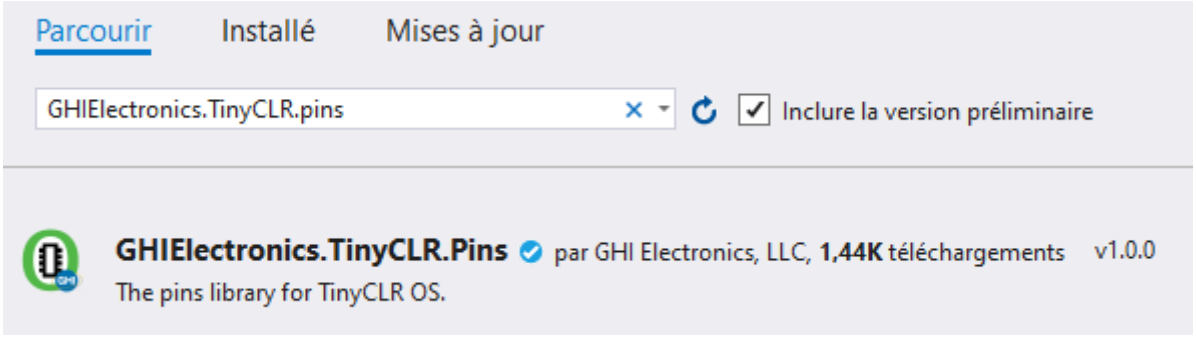
- Afficher la bibliothèque présente en vous plaçant sur l'onglet **“Installé”**.



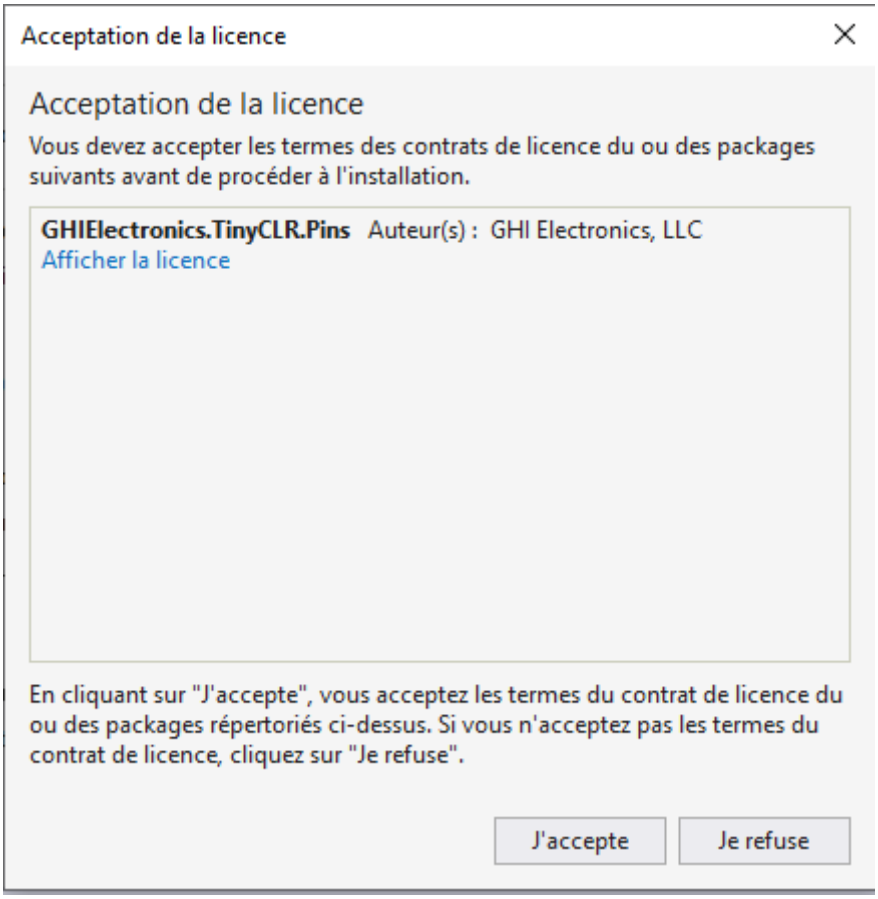
- Sélectionner **nuget.org** dans **“Source de package**”.



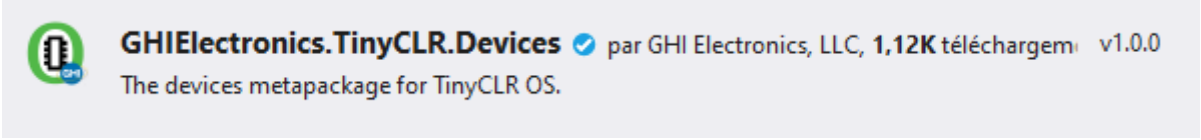
- Se placer dans l'onglet **“Parcourir”** et entrer **GHIElectronics.TinyCLR.pins** .



- Cliquer sur ***GHIElectronics.TinyCLR.Pins*** puis "**Installer**" pour placer les bibliothèques dans le projet.



- Cliquer sur **"J'accepte"**. La liste des bibliothèques installées est accessible dans **Références**.
- Refaire les mêmes opérations pour installer ***GHIElectronics.TinyCLR.Devices***



- Fermer le gestionnaire de paquets

Etape 3 : Editer le code source du programme

- Cliquer sur ***Program.cs*** dans l'explorateur de solution pour faire apparaître le squelette du

programme.

a) Déclarer les espaces de nom

Une partie du code a été renseigné par Visual Studio. C'est le cas de la liste des espaces de noms installés par défaut et de la structure minimum du corps de programme (**repère 2 de la copie d'écran ci-dessus**).

Pour le moment seuls les espaces de nom suivants sont présents :

*.cs

```
using System;
using System.Collections;
using System.Text;
using System.Threading;

namespace Blink
{
    class Program
    {
        static void Main()
        {
        }
    }
}
```

Modifier cette liste pour qu'elle corresponde au code ci-dessous.

*.cs

```
using System;
using System.Collections;
using System.Text;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

namespace Blink
{
    class Program
    {
        static void Main()
        {
        }
    }
}
```

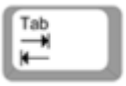
b) Construction d'un objet virtuel “led” pour contrôler la LED “LED1” de la carte

Entrer le code ci-dessous.

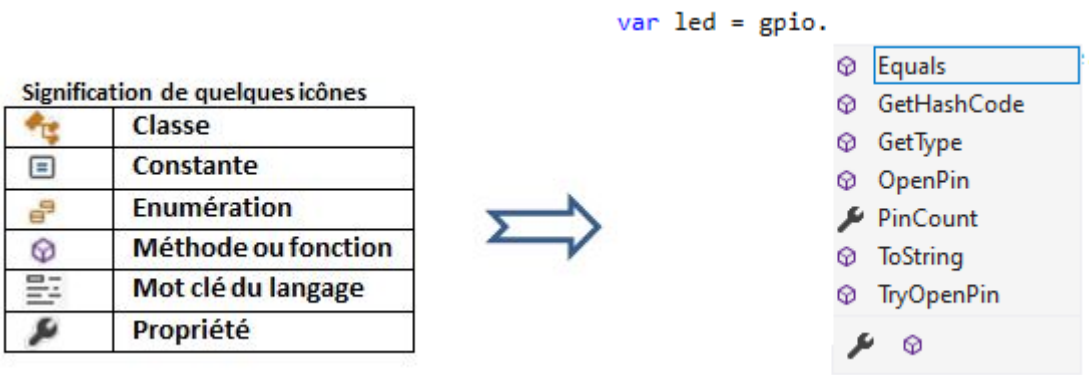
led.cs

```
        static void Main()
         {
             var gpio = GpioController.GetDefault(); // Ces deux lignes
ouvrent la broche
             var led = gpio.OpenPin(FEZ.GpioPin.Led1); // sur laquelle
est connectée LED1
             led.SetDriveMode(GpioPinDriveMode.Output); // Cette ligne
la configure en sortie
         }
```

Remarque : Vous pouvez constater l'effet de l'**autocomplétion (intellisense)** au fur et à mesure de l'écriture du code. Les propositions faites par le logiciel sont une aide précieuse lors de l'écriture du code.

Il n'est pas nécessaire d'écrire complètement un mot : un **appui sur la touche tabulation** 

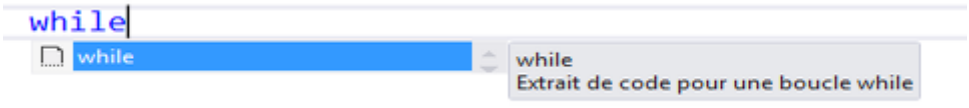
permet de l'insérer lorsqu'il est sélectionné.



---

c) Partie exécutive du programme (le corps du programme) : à placer dans une boucle infinie

Ecrire le mot ***while*** . Le logiciel vous propose un gabarit (template) de code.



Appuyer **deux fois** sur la touche de tabulation 

. Le logiciel construit la structure ***while*** pour vous.

Compléter la structure ***while*** avec le code ci-dessous :

*.cs

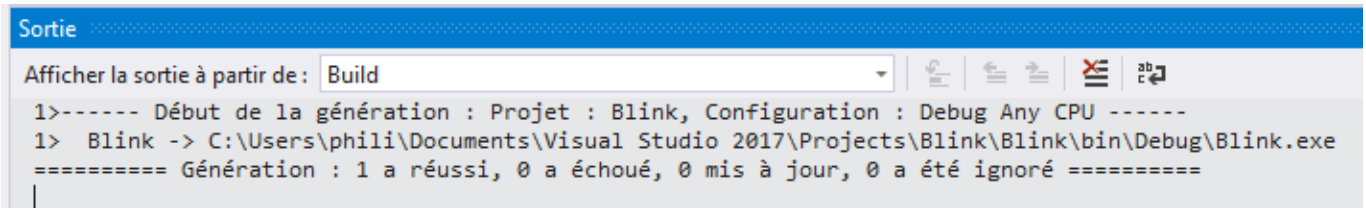
```
        static void Main()
        {
            var gpio = GpioController.GetDefault(); // Ces deux lignes
ouvrent la broche
            var led = gpio.OpenPin(FEZ.GpioPin.Led1); // sur laquelle
est connectée LED1
            led.SetDriveMode(GpioPinDriveMode.Output); // Cette ligne
la configure en sortie
        }
        while {
                led.Write(GpioPinValue.High);
                Thread.Sleep(100);
                led.Write(GpioPinValue.Low);
                Thread.Sleep(100);
              }
```

Etape 4 : Générer la solution

La fenêtre de sortie donne des informations pendant l'exécution d'un programme. Elle est utile lors de sa mise au point ou pour remplacer un périphérique d'affichage.

Si la fenêtre de sortie n'est pas présente, vous pouvez là faire apparaître en sélectionnant : **Affichage → Sortie ou [Ctrl+Alt+O]**.

Dans l'éditeur, sélectionner : **Générer → Générer la solution ou [Ctrl+Maj+B]**. S'il n'y a pas d'erreurs dans le code le logiciel indique :



Etape 5 : Transférer le programme et l'exécuter

Vérifier que l'éditeur est en mode Debug.

En cas de problème de transfert :

Appuyer sur le bouton **Reset**
-> lancer le transfert
-> relâcher le bouton Reset

Pour transférer et exécuter le programme dans la carte, cliquer sur Démarrer ou appuyer sur la

touche **F5**.

Après une série d'actions, l'IDE doit afficher **Prêt sur un fond rouge** en bas et à gauche de la fenêtre.

La LED de la carte doit clignoter !

Etape 6 : Tester le programme en mode pas à pas

Pour mettre un programme "**au point**" (**déboguer**), il est possible de l'arrêter, de le redémarrer ou de le mettre en pause.

Pour bien comprendre les possibilités du débogueur, vous allez faire fonctionner le programme en **mode pas-à-pas.**

Cliquer sur l'icône "Arrêter", ajouter l'espace de nom :

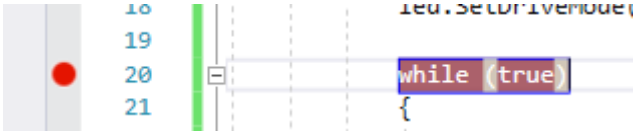
```
using System.Diagnostics;
```

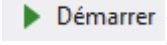
et modifier la boucle while comme ci-dessous :

*.cs

```
            while (true)
            {
                led.Write(GpioPinValue.High);
                Debug.WriteLine("Led1 éclairée");
                Thread.Sleep(100);
                led.Write(GpioPinValue.Low);
                Debug.WriteLine("Led1 éteinte");
                Thread.Sleep(100);
            }
```

Placer un **point d'arrêt** (rond rouge) en cliquant à gauche du mot ***while*** comme ci-dessous.

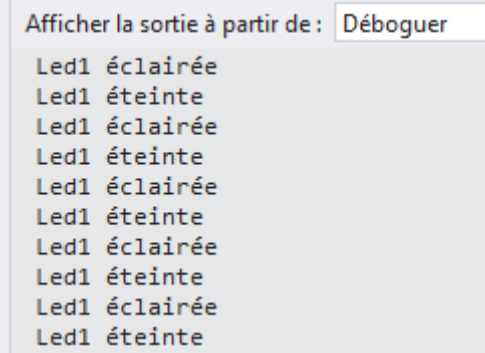


Relancer le programme (touche F5 ou 

). Celui-ci s'arrête sur ***while***. Vous pouvez

l'exécuter ligne par ligne (**mode pas-à-pas**) en appuyant sur la **touche F10**. (Un appui exécute une ligne)

Résultat attendu dans la fenêtre de sortie et comportement de la LED



Exercices

Exercice 1
Modifier le programme pour que la LED clignote à une fréquence de 4Hz avec un rapport cyclique de 1/10

---

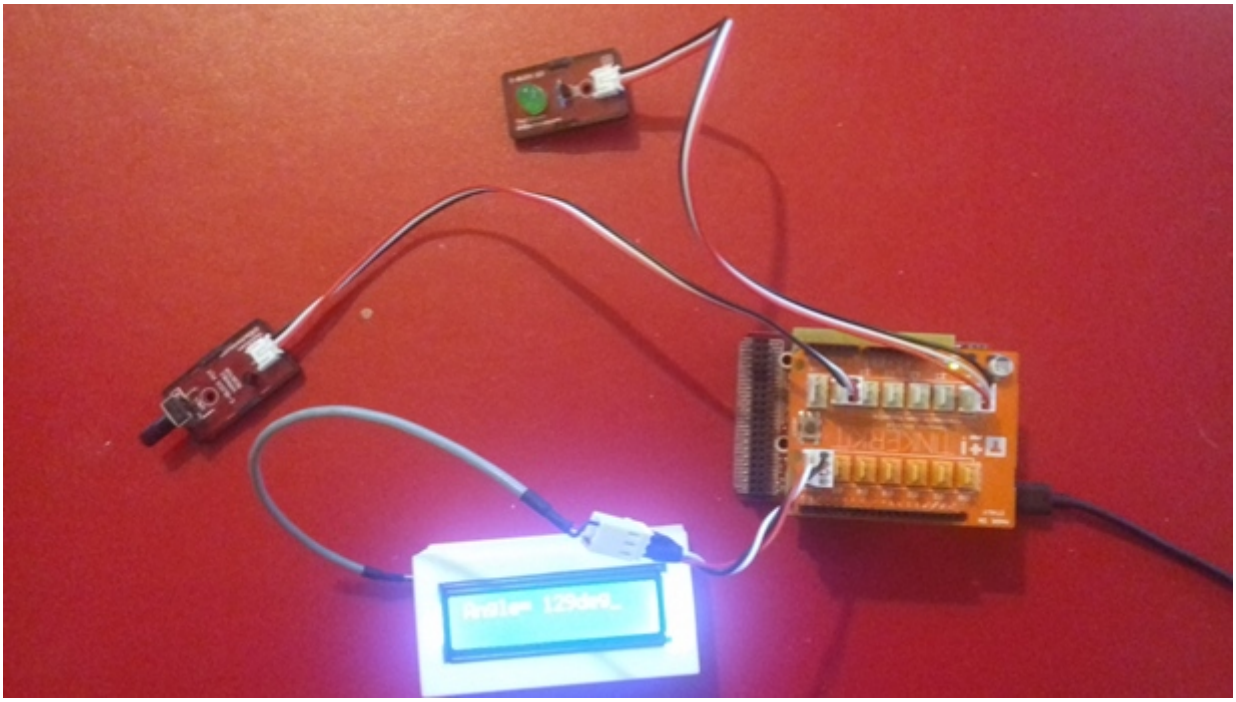
3. Deuxième programme : "MesureAngle" (entrée analogique et sortie numérique)

Cahier des charges du programme 2 : afficher la position angulaire d'un axe sur un LCD.

Matériels

- Carte FEZ T18 et shield Tinkerkit
- Potentiomètre
- Afficheur LCD 2 lignes de 16 caractères et module ELCD162

Montage à réaliser



Etape 1 : Créer le projet

Reprendre la démarche vue dans le programme 1. Nommer le projet ***MesureAngle***.

Etape 2 : Installer des bibliothèques dans le projet

Installer les bibliothèques du projet précédent et ajouter **Webge.TCLR.Displays.ELCD162**.

Etape 3 : Editer le code source du programme

1. Modifier la liste des espaces de noms comme ci-dessous (on conserve seulement ceux qui sont utiles au projet !)

using.cs

```
using System;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.Adc;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;
using Webge.TCLR.Displays;
using System.Threading;
```

2. Partie déclarative du programme

Compléter le fichier "*Program.cs*" avec les extraits de code ci-dessous:

a. Déclaration des constantes

const.cs

```
namespace MesureAngle
{
    class Program
    {
        static void Main()
        {
          // Constantes
          const UInt16 delay = 500;   // En ms
        }
    }
}
```

b. Déclaration des variables

var.cs

```
 // A la suite du code précédent, entrer :
 // Variables
 var Npot = 0;
 var Angle = 0;
```

c. Configuration des entrées, sorties

es.cs

```
// A la suite du code précédent, entrer :

// Création des objets pour une carte FEZ T18
//-----------------------------------------------------------------
// CAN
var adc = AdcController.GetDefault();
var Potentiometre = adc.OpenChannel(FEZ.AdcChannel.A5); //
Potentiomètre connecté sur I5 Tinkerkit

// Afficheur
var lcd = new ELCD162();  // Connecté sur SERIAL Tinkerkit
```

d. Initialisation de l'afficheur

init.cs

```
 // A la suite du code précédent, entrer :
// Initialisation
//-----------------------------------------------------------------
lcd.Init(); lcd.CursorOff();
```

3. Partie exécutive du programme

while.cs

```
 // A la suite du code précédent, entrer :
while (true)
 {
   // Lire la position angulaire
   Npot = Potentiometre.ReadValue();

   // Calcul de la position angulaire
   Angle = 270*Npot/4095;

   // Afficher la position angulaire
   lcd.ClearScreen();
   lcd.PutString("Pos = " + Angle.ToString("F1") + "deg");
   Thread.Sleep(delay);
 }
```

Etape 4 : Générer la solution

Reprendre la démarche vue dans le programme 1.

Etape 5 : Transférer le programme et l'exécuter

Reprendre la démarche vue dans le programme 1.

Exercices

Exercice 2
Modifier le programme pour qu'une LED simulant une alarme (à rajouter sur la carte) s'éclaire si l'angle n'est pas compris dans l'intervalle [**120°, 150°C**]. En C#, le **OU** logique se code ||, le **ET** logique se code **&&**. Pour laConstruction de l'objet virtuel “led”, entrer le code ci-dessous.

led.cs

```
        static void Main()
         {
             var gpio = GpioController.GetDefault(); // Ces deux lignes
ouvrent la broche
             var led = gpio.OpenPin(FEZ.GpioPin.A0); // sur laquelle est
connectée LED1
             led.SetDriveMode(GpioPinDriveMode.Output); // Cette ligne
la configure en sortie
         }
```

Exercice 3
Remplacer le potentiomètre par un module thermomètre GHI pour afficher la température ambiante sur le LCD.
Traitement : T ← 0,0185N - 20,5

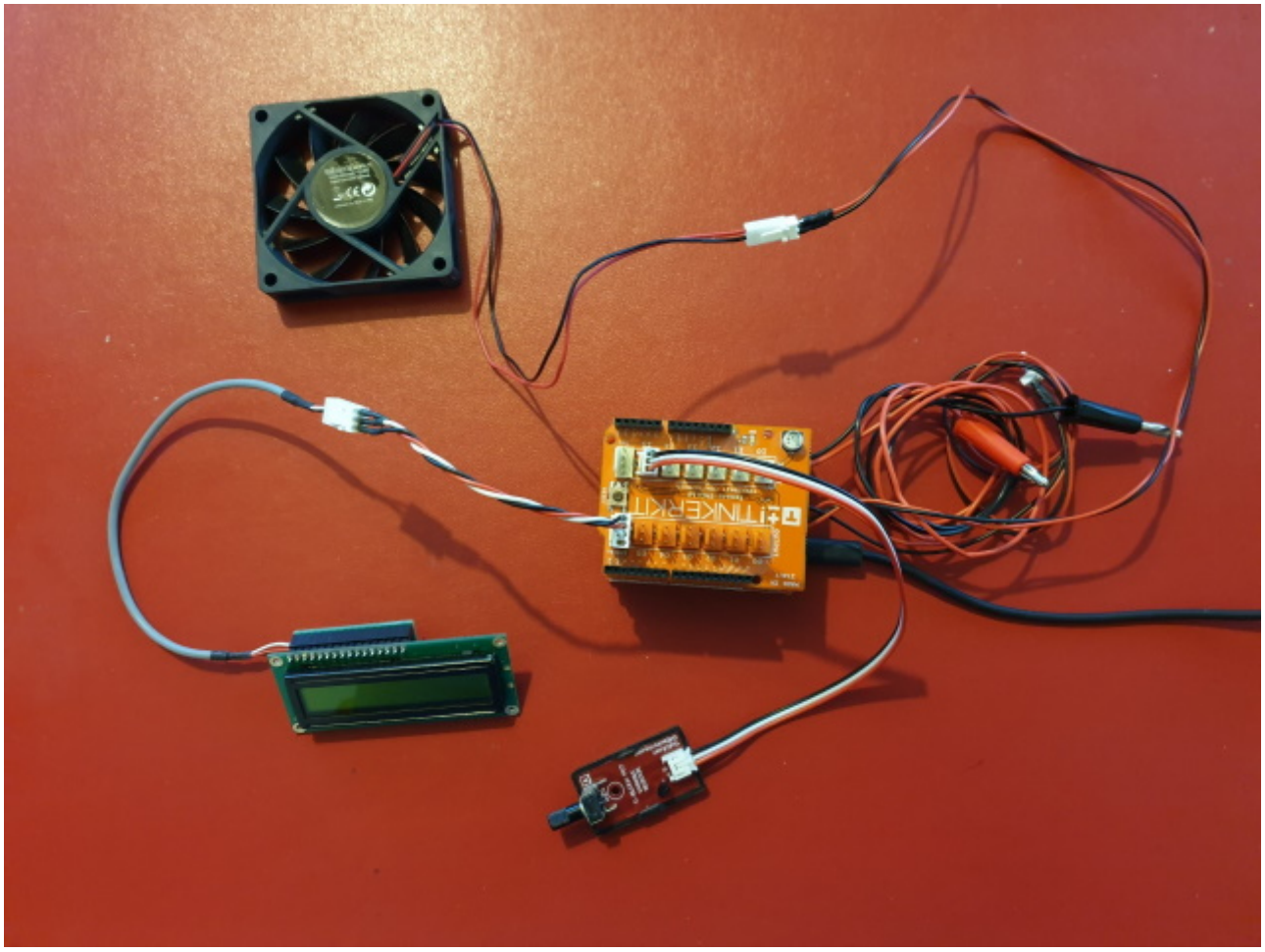
4. Troisième programme : Commande d'un ventilateur (entrée analogique, sortie PWM)

Cahier des charges du programme 3 : régler la vitesse d'un ventilateur entre 0 et 100% de sa valeur maximum à partir de la mesure de la position de l'axe d'un potentiomètre. Afficher le réglage sur un LCD

Matériels

- Carte FEZ T18 et shield Tinkerkit
- Shield Ardumoto
- Potentiomètre
- Ventilateur 12V - 200mA
- Afficheur LCD 2 lignes de 16 caractères et module ELCD162

Montage à réaliser



Etape 1 : Créer le projet

Reprendre la démarche vue dans le programme 1. Nommer le projet ***Ventilation***.

Etape 2 : Installer des bibliothèques dans le projet

- GHIElectronics.TinyCLR.Pins
- GHIElectronics.TinyCLR.Devices
- Webge.TCLR.Displays.ELCD162

Etape 3 : Editer le code source du programme

1. Modifier la liste des espaces de noms comme ci-dessous (on conserve seulement ceux qui sont utiles au projet !)

using.cs

```
using System;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.Adc;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.Pwm;
```

```
using GHIElectronics.TinyCLR.Pins;
using System.Threading;
using Webge.TCLR.Displays;
```

2. Partie déclarative du programme Compléter le fichier "*Program.cs*" avec les extraits de code ci-dessous:

a. Déclaration des constantes

const.cs

```
namespace MesureAngle
{
    class Program
    {
        static void Main()
        {
          // Constantes
          // aucune
        }
    }
}
```

b. Déclaration des variables

var.cs

```
 // A la suite du code précédent, entrer :
 // Variables
 var Npot = 0.0;
 var ConsigneVentilation = 0.0;
```

c. Configuration des entrées, sorties

es.cs

```
// A la suite du code précédent, entrer :

// Création des objets pour une carte FEZ T18
//-----------------------------------------------------------------
// CAN 12bits
// Voir le programme 2

// Signal PWM
var pwm = PwmController.FromName(FEZ.PwmChannel.Controller3.Id);
var CdeVentilation = pwm.OpenChannel(FEZ.PwmChannel.Controller3.D11);
pwm.SetDesiredFrequency(5000);
```

```
// GPIO
var gpio = GpioController.GetDefault();
var SENS = gpio.OpenPin(FEZ.GpioPin.D13);
SENS.SetDriveMode(GpioPinDriveMode.Output);
SENS.Write(GpioPinValue.Low);

// Afficheur
// Voir le programme 2
```

d. Initialisation de l'afficheur

init.cs

```
// A la suite du code précédent, entrer :
// Initialisation
//-----------------------------------------------------------------
CdeVentilation.SetActiveDutyCyclePercentage(0); // Commande Ventilation
= 0%
CdeVentilation.Start();
lcd.Init(); lcd.CursorOff();
```

3. Partie exécutive du programme

while.cs

```
 // A la suite du code précédent, entrer :
while (true)
 {
   // Lire la position angulaire
   // Voir le programme 2

   // Calculer la consigne de ventilation en %
   // Voir le programme 2

   // Commander le ventilateur
   // Voir la méthode dans Initialisation ci-dessus

   // Afficher la consigne de ventilation sur le lcd
   // Voir le programme 2
 }
```

Etape 4 : Générer la solution

Reprendre la démarche vue dans le programme 1.

Etape 5 : Transférer le programme et l'exécuter

Reprendre la démarche vue dans le programme 1.

Exercice

Ecrire l'algorithme du programme “Ventilation”

Sources des exemples Les sources des exemples (compilées avec Visual Studio Community 2017) sont téléchargeables ici.
