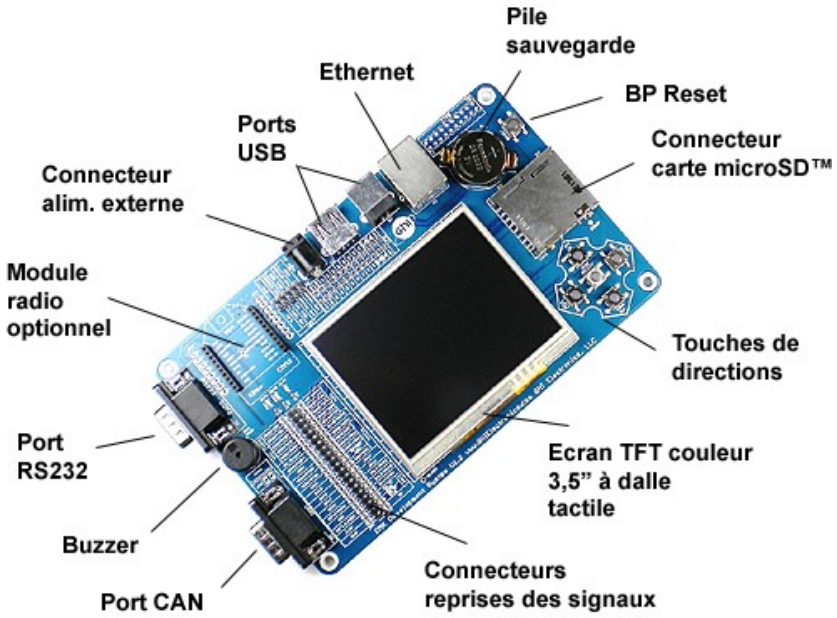
Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Carte de développement EMX



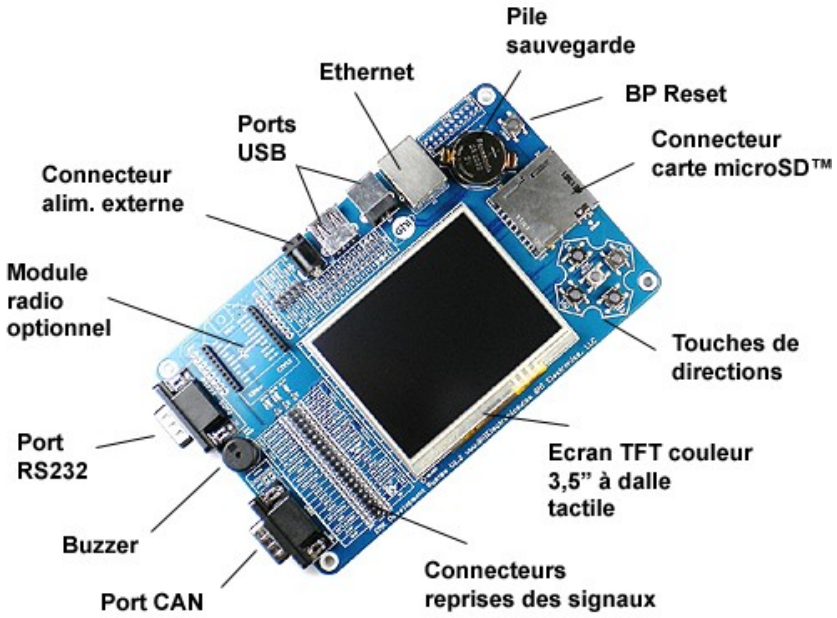
Lemoine Jean-Daniel
K4B 2010
2008-2011

Enseignants
Thierry LEQUEU
Philippe AUGER

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS
Institut Universitaire de Technologie
Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Carte de développement EMX



Lemoine Jean-Daniel
K4B 2010
2008-2011

Enseignants
Thierry LEQUEU
Philippe AUGER

Sommaire

Introduction

Si au cour de ces dernières années les processeurs ont subis une incroyable progression en terme de performance, il en fut de même pour les micro-processeurs devant traiter des tâches toujours plus complexes, un bon exemple est le téléphone mobile, du simple téléphone il y a quelques années au véritable ordi-phone. Leurs micro-processeurs ont du s'adapter.

Le but de ce projet est la découverte et la mise en œuvre de la programmation sur plateforme ARM7. Les micro-processeurs de la famille ARM utilisés dans de nombreuses applications embarquées tels que la téléphonie mobile et les ordinateurs de bord.

Il existe différentes façons de programmer ces micro-processeurs, pour ce projet nous utiliserons l'environnement de développement intergré de Microsoft, Visual Studio 2010. La programmation s'effectuera en C sharp, un langage orienté objet comme le C++. Nous verrons au travers de ce projet comment utiliser Visual Studio et le configurer pour utiliser pleinement la carte EMX

Dans une première partie je détaillerais le cahier des charges de ce projet, puis je traiterais des recherches bibliographiques qui m'ont amené vers la carte EMX. Ensuite je présenterais la Carte EMX et le langage de programmation Csharp. La suite du rapport s'articulera autour des différents tests et parties de programme que je réaliserais.

1. Cahier des charges

1.1. Descriptif du projet

Le projet mettra en œuvre une carte de développement basée sur micro-processeur de la famille ARM. La carte devra pouvoir communiquer via un BUS CAN et utiliser ses différentes entrées/sorties. Une mise en réseau via un module Wifi additionnel devra être envisagée.

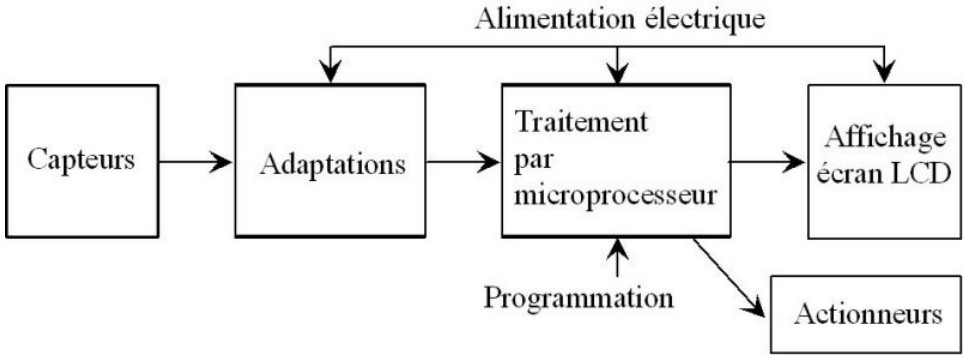


Illustration 1: Analyse fonctionnelle [0]

1.2. Contraintes

Le projet ayant un but didactique, les principales contraires sont celles liées à l'utilisation de la carte de développement.

La carte devra comporter un BUS CAN, ainsi que différentes entrées/sorties analogiques afin de relier d'éventuels capteurs. Aucunes contraintes ne sont appliquées à ce projet, toutefois afin d'envisager une possible embarcation du dispositif dans un véhicule, la taille devra être restreinte.

2. Recherches bibliographiques

La conception d'une carte basée sur un micro-processeur ARM n'étant pas réalisable en 7 semaines, il a fallu trouver un dispositif répondant au cahier des charges. Plusieurs choix étaient envisageables toutefois afin de faciliter le projet et d'éviter d'éventuels problèmes de compatibilité, notre choix s'est donc porté sur une carte de développement intégrant micro-processeur et BUS CAN.

La carte EMX de *GHI Electronics* répondant parfaitement à ces critères, Il a fallu s'attarder sur la façon de la programmer. Une grosse partie du projet s'est donc porté sur l'apprentissage du langage orienté objet Csharp et sur l'utilisation de l'EDI[1] de Microsoft VIISUAL STUDIO 2010.

Afin de répondre au cahier des charges une partie des recherches portera également sur le BUS CAN ainsi que le BUS I²C.

1 *EDI: Environnement de Développement intégrer.*

3. Carte de développement EMX

3.1. Module EMX

La carte choisie s'articule autour d'un module EMX, ce module est le cœur de la carte puisque c'est lui qui comporte le micro-processeur, la mémoire flash et la RAM. Mais aussi tous les circuits électroniques annexes gérant les autres fonctions de la carte (contrôleur USB, Ethernet, BUS CAN, etc).

Illustration 2: Vue du dessus du module EMX[1]

Caractéristique du module EMX

- Processeur ARM7 72Mhz 32 bit
- 16MB de mémoire RAM
- 4,5MB de mémoire FLASH
- Un contrôleur LCD
- Un contrôleur Ethernet
- Implémentation de la couche TCP/IP
- Protocole SSL
- Un driver WiFi ZG2100
- Un contrôleur USB
- 76 boutons possibles
- 39 entrées d'interruption
- 2 bus spi
- un bus i2c
- 4 bus UART
- 2 channels CAN
- 7 entrées analogiques 10 bits

- une sortie analogique 10 bits
- 6 sorties PWM

Le module comporte également un autre avantage, sa faible consommation puisque en activant toutes ses fonctionnalités elle est seulement de 160mA.

Le module respecte la norme Rhos[2], c'est-à-dire que la carte ne comporte pas l'une des 6 substances jugées dangereuses couramment utilisées en électronique, à savoir : le plomb, le mercure, le cadmium, le chrome, les polybromobiphényles et les polybromodiphényléthers.

Le module implémente le .NET Micro Framework de *Microsoft* qui vise à simplifier le développement d'application pour le monde de l'embarqué. Mais aussi à faire profiter ces systèmes des innovations des outils informatiques modernes.

3.2. Carte de développement EMX.

La carte de développement proposée par *GHI Electronics* permet de mettre en œuvre rapidement le module EMX et de tester ses possibilités. Mais elle permet également d'avoir une véritable base de travail sans avoir à créer des prototypes, cela réduit donc grandement les coûts de fabrication.

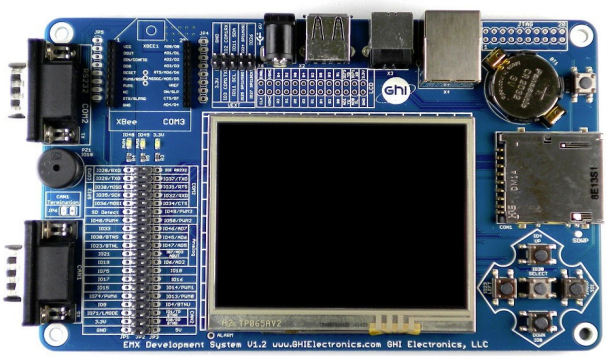


Illustration 3: Carte de développement[1]

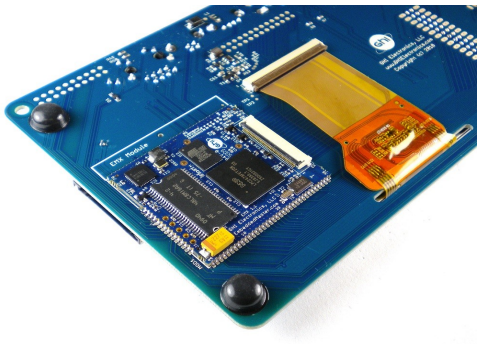


Illustration 4: Implantation du module EMX au dos de la carte[1]

2 *RoHS: Restriction of use of certain Hazardous Substances*

Caractéristique la carte de développement

- Écran TFT 320 x 240mm avec écran tactile
- Connecteur RJ45 pour l'Ethernet
- 5 Boutons
- Accès à tous les bus (I²C, SPI)
- Connecteur USB
- Empreinte module Xbee
- Bus CAN sur connecteur DSUB 9 broches
- Un buzzer
- Une horloge temps réel
- Un port RS232
- Un connecteur d'alimentation

Cette carte reprend donc les principales fonctions du module EMX et met à disposition de façon simplifiée toutes les entrées/sorties via des connecteurs. Elle comporte également un port USB dédié à la programmation et au débogage.

Cette carte respecte également la norme RoHS

4. Le langage de programmation

4.1. Le C#

Le C# est un langage de programmation crée par *Microsoft*. Il reprend les concepts de différents langages comme le C, C++ et le Java, c'est donc un langage orienté objet. C'est un langage récent puisque la première version est apparue en 2001.

Sa principale fonction est de pouvoir exploiter pleinement le framework .NET également crée par *Microsoft.* La dernière version de C# est la C# 4.0 sortie le 12 avril 2010, c'est cette version qui sera utilisée dans ce projet.

4.2. Le .NET Micro Framework

Le .NET Micro Framework utilisé dans ce projet, n'est autre que l'application du .Net au monde de l'embarqué. Il est conçu pour utiliser moins de ressources et moins d'énergie.

Son but est de rendre la programmation de système embarqué plus simple et surtout similaire à celle sur .NET pour que le plus grand nombre de développeur puisse l'utiliser.

Le .NET Micro Framework ne prend pas en charge d'autres langages que le C# ce qui le rend fortement dépendant à Windows. La version actuelle est la 4,1 sortie le 24 août 2010.

5. Microsoft Visual Studio 2010

5.1. Présentation

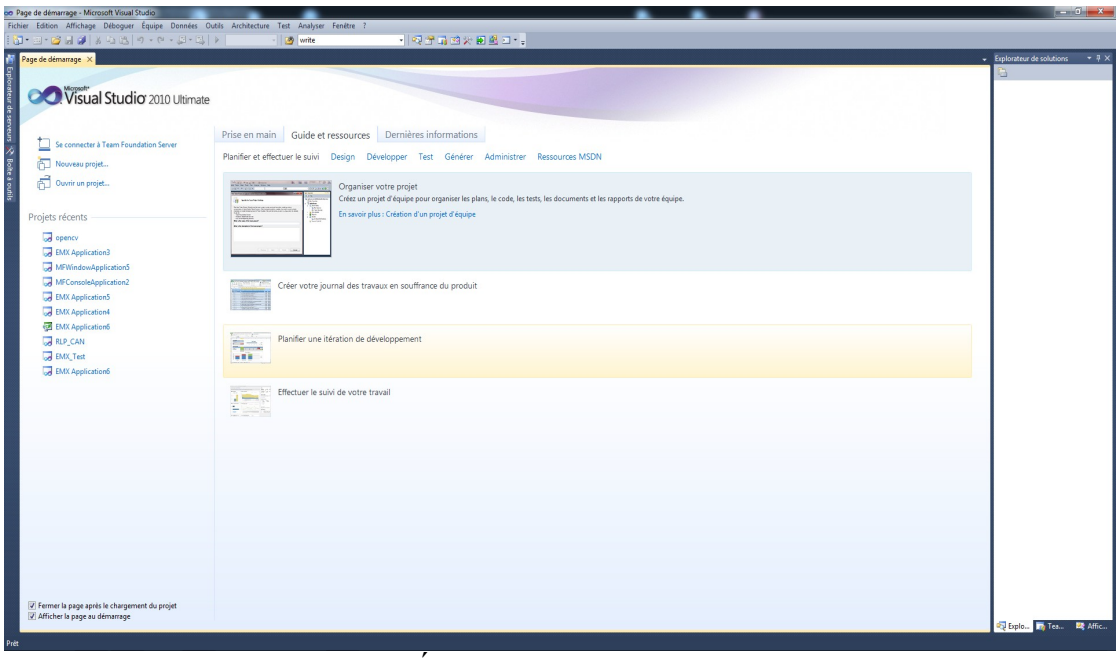


Illustration 5: Écran d'accueil de Visual Studio2010

Pour la programmation de ce projet nous utiliserons l'EDI de *Microsoft*. Cet EDI est très performant, il permet la création de projet pour diverses langages (C, C++, C#, VB). La conception de projet y est très simple, la totalité du logiciel est en français. Un des grands avantages et l'auto-complétion du code, c'est-à-dire que l'EDI se souvient de nos variables, de nos fonctions ou de nos méthodes et nous les propose en fonction de la situation et des premières lettres du mot que l'on utilise.

5.2. Configuration de Visual Studio pour la carte EMX

Avant de commencer l'utilisation de notre carte EMX, il faudra installer le .NET Micro Framework que vous trouverez à cette adresse **http://www.microsoft.com/netmf/default.mspx**. Afin de pouvoir utiliser toutes les capacités de la carte il sera impératif d'utiliser le SDK fourni par *GHI Electronics* disponible dans la section « download » de cette page **http://www.ghielectronics.com/product/129**. L'installation se fait automatiquement.

5.3. Création d'un nouveau projet

Maintenant que les deux SDK complémentaires sont installées, il va être possible de créer un nouveau projet. Pour cela démarrer Visual Studio et cliquer sur **Fichier → Nouveau → Projet** dans le menu de gauche, sélectionner **Visual C#** et **Micro Framework** et choisir, soit **Window Application** soit **Console Application.** Le premier vous offre un début de code gérant les boutons

de la carte, ceux-ci pouvant être problématiques avec l'utilisation des fonctions analogiques mais nous y reviendrons plus tard.

Maintenant nous avons notre projet composé de plusieurs fichiers dont le principal « Program.cs », c'est ici que nous écrirons notre code, c'est l'équivalent du fichier main.cpp en C++. Tous les fichiers en Csharp comportent l'extension « cs ».

Une étape importante avant de lancer notre premier programme est de configurer le « déploiement » du programme, c'est-à-dire utiliser l'émulateur de Visual Studio ou envoyer le programme sur la carte. Pour cela cliquer sur **Projet → Propriétés de <non du projet>** puis se rendre dans la dernière section **.NET micro framework** et choisir le type de sortie que l'on veux, à savoir que pour avoir la carte dans la section USB il faut préalablement l'avoir connectée.

6. Interface graphique

Dans cette section j'expliquerais mon travail dans le domaine de la création d'interface homme machine, c'est-à-dire la liaison entre l'écran tactile et la carte.

Cette partie est relativement délicate puisque le .NET Micro Framework n'accepte que des formes basiques, cercles, rectangles, importation d'images, points, lignes, textes. De ce fait, pour créer une interface harmonieuse comme celle du programme de présentation de la carte il faut créer des icônes avec un outil comme photoshop puis importer ces images et les placer sur l'écran. Il y a donc tout un travail préliminaire de design pour créer une application.

6.1. Configuration

Pour utiliser l'affichage de texte à l'écran il faut impérativement importer une police comportant l'extension .tinyfnt dans VisualStudio 2010, deux polices sont fournies avec le Micro Framework par défaut dans le dossier suivant **C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.1\Fonts** :

- NinaB.tinyfnt
- small.tinyfnt

Il est possible de créer d'autres polices d'écriture via l'outil **TFConvert.exe**, fourni dans **C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.1\Tools.**

Pour importer les polices d'écriture dans VisualStudio il vous faut, une fois votre projet crée, double-cliquer sur Resources.resx dans « l'explorateur de solutions », puis cliquer sur **Ajouter une ressource → Ajouter un fichier existant** et indiquer le fichier à importer, par exemple « small.tinyfnt ». La même manipulation sera à faire pour importer des images.

6.2. Programme

Il existe différentes façons d'afficher un texte ou une image à l'écran de la carte, soit par la création d'un bitmap, soit par l'utilisation d'un objet Image ou Texte. Voici ci-après un exemple de Bitmap.

Déclaration d'un Bitmap et chargement de la police small.tinyfnt

```
Bitmap LCD = New Bitmap(SystemMetrics.ScreenWidth, SystemMetrics.ScreenHeight);
            Font small = Resources.GetFont(Resources.FontResources.small);
```

Écriture simple de texte

```
            LCD.Clear();
            LCD.DrawText("Texte", small, Colors.Gray, 100, 100);
            LCD.DrawText("Texte2", small, Colors.Gray, 50, 50);
            LCD.Flush();
```

Je ne détaille pas ici les valeurs attendues par DrawTexte puisque celles-ci sont données par VisualStudio lorsqu'on écrit DrawText. Il est possible d'écrire plusieurs textes ou même d'écrire un texte dans une même fenêtre.

Avec les bitmaps il est possible de dessiner , des rectangles, des ellipses, du texte, des lignes, des points. Mais aussi gérer la transparence des bitmaps, appliquer une rotation, des dégradés, etc.

Une deuxième solution consiste à créer une « fenêtre » et d'y afficher un texte ou des images, c'est cette solution qui sera retenue dans le cadre d'interface complexe.

L'exemple suivant est tiré du code génère par un projet « Windows application ».

Affichage d'un texte stocker en ressource

```
  public Window CreateWindow()
        {
            //Creation d'une fenêtre à la taille de l'écran
            mainWindow = new Window();
            mainWindow.Height = SystemMetrics.ScreenHeight;
            mainWindow.Width = SystemMetrics.ScreenWidth;

            //Creation d'un objet text
            Text text = new Text();

            text.Font = Resources.GetFont(Resources.FontResources.small);
            text.TextContent = Resources.GetString(Resources.StringResources.String1);
            text.HorizontalAlignment =
Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
            text.VerticalAlignment =
Microsoft.SPOT.Presentation.VerticalAlignment.Center;

            //Ajout du text à l'écran
            mainWindow.Child = text;
            //Rend la fenêtre visible
            mainWindow.Visibility = Visibility.Visible;

            return mainWindow;
        }
```

Nous pouvons également créer avec cette méthode des objets de type Image. On peut ajouter autant d'éléments « enfants » à la fenêtre grâce, par exemple, à **mainWindow.Child = image**.

Grâce à ceci il suffit d'appeler les ressources stockées, comme par exemple, les chaines de caractère mais aussi les images.

Cette solution est celle adoptée par le programme de base de la carte EMX.

7. Entrée analogique

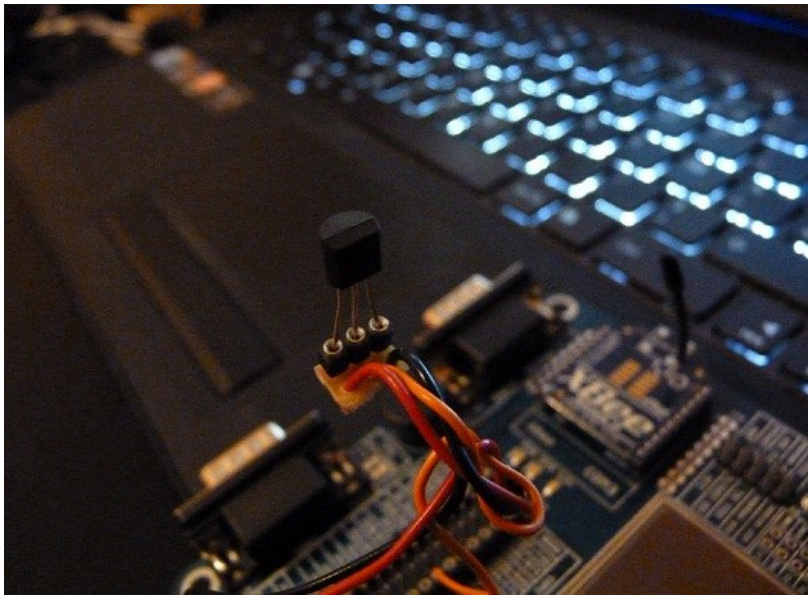


Illustration 6: Capteur de température LM35

L'idée sera ici de simuler un capteur renvoyant une tension analogique inférieure ou égale au maximum à +5V. Pour réaliser cette simulation nous utiliserons simplement un potentiomètre multi-tours de 10KOhm.

7.1. Programme

```
  public static void Main()
        {
            //Construction d'un objet Temp utilisant l'entrée analogique Ain7

            AnalogIn Temp = new AnalogIn(AnalogIn.Pin.Ain7);
            //boucle de test infinie
            while (true)
            {
                //Définition de l'échelle de mesure.
                Temp.SetLinearScale(0, 1023);

                int temp;
                //Lecture de la valeur appliqué à l'entrée.
                temp = Temp.Read();
                //On affiche dans le terminal de débogage la valeur que l'on
convertie en chaine.
                Debug.Print("Valeur = " + temp.ToString());
                Thread.Sleep(10);
                //Creation d'un nouveau bitmap à la taille de l'écran.
                Bitmap bmp = new Bitmap(SystemMetrics.ScreenWidth,
SystemMetrics.ScreenHeight);
                //Creation d'un rectangle qui fera office de barregraphe
                rectangle(Microsoft.SPOT.Presentation.Media.Color.Black,
                    1,
                    50, 50,
                    (temp/4), 20,
                    0, 0,
                    Microsoft.SPOT.Presentation.Media.Colors.Green,
                    50, 50,
                    Microsoft.SPOT.Presentation.Media.Colors.Red,
                    100 + (temp / 4), 110,
                    Bitmap.OpacityOpaque);
                //On affiche le bitmap
                bmp.Flush();
            }
        }
```

Ce programme a pour but d'afficher un bar graphe en couleur représentant la tension appliquée à la l'entrée analogique. Cela pourrait être un signal venant d'un compte-tours ou d'un accélérateur dans le cadre d'une voiture ou d'un kart par exemple.

Remarques

- La création du projet « Console application » est nécessaire car le projet « Window application » utilise les entrées analogiques pour les boutons de la carte, il en va de même pour l'écran tactile. Je n'ai pas trouvée de solution à ce problème et le support de *GHI Electronics* n'a pas pu me fournir de réponse. Ce problème est très handicapant, puisqu'il ne sera pas possible d'utiliser conjointement une interface graphique et les entrées analogiques. Il sera donc préférable d'utiliser des capteurs numériques ou utilisant un bus (CAN, spi, i²c).
- A la mise sous tension de la carte les entrées analogiques, qui peuvent également être des sorties, sont à 5V. Attention donc à ne connecter le capteur qu'après avoir charger le programme.

– La documentation de l'ARM7 de NXP donne comme information, une tension limite d'entrée de l'ADC[3] de +3,6V, mais il semblerais que cela fonctionne avec des tensions de +5V. Les plans du module EMX n'étant pas disponible, il n'est pas possible de justifier cette remarque.

Table 16. ADC static characteristics

V_{DDA} = 2.5 V to 3.6 V; T_{amb} = −40 °C to +85 °C unless otherwise specified; ADC frequency 4.5 MHz.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IA}	analog input voltage		0	-	V_{DDA}	V
C_{ia}	analog input capacitance		-	-	1	pF
E_D	differential linearity error	[1][2][3]	-	-	±1	LSB
$E_{L(adj)}$	integral non-linearity	[1][4]	-	-	±2	LSB
E_O	offset error	[1][5]	-	-	±3	LSB
E_G	gain error	[1][6]	-	-	±0.5	%
E_T	absolute error	[1][7]	-	-	±4	LSB
R_{vsi}	voltage source interface resistance	[8]	-	-	40	kΩ

Illustration 7: Caractéristique de l'ADC[2]

8. BUS Can

Le bus Can est un bus de communication série utilisée principalement pour les réseaux de terrain. Je ne m'étendrais pas ici sur les caractéristiques du bus[4] Can puisque celles-ci pourraient à elles seules faire l'objet d'un rapport, mais plutôt sur le coté hardware du bus Can.

8.1. Principe de communication

Le bus Can utilise pour communiquer une paire filaire différentielle composée de deux fils, CAN_H et CAN_L. Ce type de ligne est utilisé principalement pour s'affranchir des parasitages dû à l'environnement d'utilisation du bus Can (réseau de terrain). Le bus Can n'utilise pas les traditionnels niveaux 0 et 1 mais des niveaux dits dominants et récessifs.

Paramètres	CAN *low speed*	CAN *high speed*
Débit	125 kb/s	125 kb/s à 1 Mb/s
Nombre de nœuds sur le bus	2 à 20	2 à 30
Courant de sortie (mode émission)	> 1 mA sur 2,2 kΩ	25 à 50 mA sur 60Ω
Niveau dominant	CAN H = 4V CAN L = 1V	CAN H = 3,5 V CAN L = 1,5 V
Niveau récessif	CAN H = 1,75V CAN L = 3,25V	CAN H = 2,5 V CAN L = 2,5 V
Caractéristique du câble	30 pF entre les câbles de ligne	2*120Ω
Tensions d'alimentation	5V	5V

Illustration 8: Niveau bus Can[3]

3 *ADC : Analog Digital Converteur*

4 *Cours sur le bus Can de l'école national supérieur électronique, informatique et radiocommunications à cette adresse http://uuu.enseirb.fr/~kadionik/formation/canbus/canbus_enseirb.pdf.*

Pour ces tests la vitesse du bus Can sera de 250KBit/s la ligne sera donc de type CAN High Speed. Voici un exemple du principe dominant récessif :

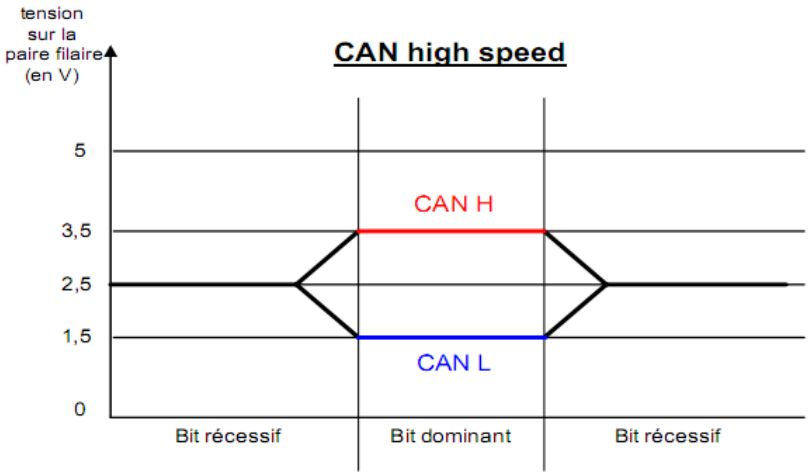


Figure 24 : Niveaux de tension du bus CAN *high speed*

Illustration 9: Niveau de tension du bus CAN High Speed[3]

8.2. Dongle CANUSB

Afin de faciliter la partie test du programme, j'ai utilisé un dongle CANUSB celui-ci utilisant aussi un connecteur Sub-D9, l'interconnexion des équipements est ainsi facilitée.

Ce dongle est vendu avec un logiciel nommé CAN Monitor Pro permettant d'envoyer et recevoir des trames Can, il est notamment possible de configurer ce logiciel en fonction de la vitesse du bus.

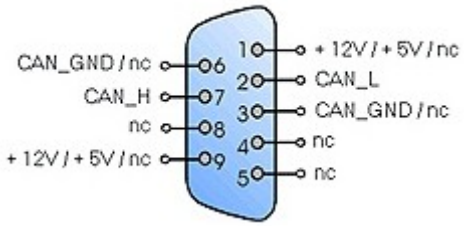


Illustration 10: Brochage du dongle CANUSB[4]

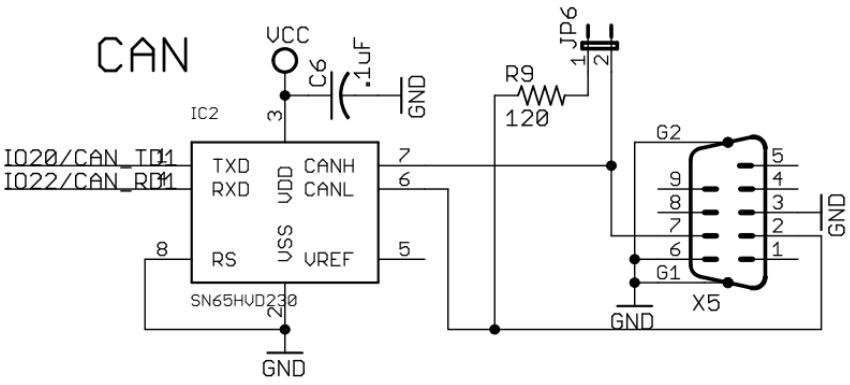


Illustration 11: Brochage du bus CAN de la carte de développement[1]

Le connecteur de la carte et du dongle étant tous les deux des embases mâles, j'ai réalisé un câble femelle/femelle afin des les interconnecter, il n'a pas été nécessaire de mettre de résistances de fin de ligne car la longeur du câble ne crée pas de rebond de données.

Comme nous pouvons le voir sur ces deux schémas les connecteurs respectent le même câblage.

8.3. Programme

La programmation du bus n'étant pas très détaillée, notamment sur la réception des données, il m'a fallu un certain temps avant de trouver la solution de programmation adéquate.

De plus la class[5] Can développer par GHI est en plein changement, par conséquent un message stipulant que la classe est obsolète apparaît dans le débugger.

```
public static void Main()
        {
            int T1, T2, BRP;

            //Cette partie est donnée par GHI Electronics
            //Elle sert à configurer la vitesse du Bus CAN de la carte EMX
            ////////////////////////////////////////////////////////////////////////////////
////////////////
            // Bitrate 250Kbps
            // CLK = 72 Mhz, with BRP = 12 -> 6Mhz CAN clock
            // 6Mhz/250Kbps = 24 TQ
            // T1 = 16 minus 1 for sync = 15
            // T2 = 8
            // 15 + 1 + 8 = 24 TQs which is what we need
            ////////////////////////////////////////////////////////////////////////////////
////////////////
            BRP = 12;
            T1 = 15;
            T2 = 8;

            //Ouverture du channel1 à 250Kb/s
            CAN can = new CAN(CAN.Channel.Channel_1, (uint)(((T2 - 1) << 20) | ((T1 - 1)
<< 16) | ((BRP - 1) << 0)));

            //Declaration d'un message can pour l'envoie et la réception
            CAN.Message message = new CAN.Message();
```

Suite du programme page suivante.

5 *Class : fichier Csharp.*

```
            //Utilisation du message pour remplir des tableau avec les valeurs reçu
            CAN.Message[] msgList = new CAN.Message[] { message };
            CAN.Message[] recMsg = new CAN.Message[] { message };

            message.data[0] = 1;
            message.data[1] = 2;
            message.data[2] = 3;
            message.data[3] = 4;
            message.data[4] = 5;
            message.data[5] = 6;
            message.data[6] = 7;
            message.data[7] = 8;

            message.DLC = 8;
            message.ArbID = 0xD6;
            message.isEID = true;
            message.isRTR = false;

            int sentCount;
            //Test pour savoir si l'on peu parler
            if (can.IsTxBufferAvailable())
                sentCount = can.PostMessages(msgList);
            while (true)
            {
                //On attend un envoie de message
                while (can.GetRxQueueCount() == 0) ;
                //On stock le message reçu
                can.GetMessages(recMsg);
                //On affiche l'id
                Debug.Print("Id : "+(recMsg[0].ArbID).ToString());
                //Et les données reçu
                for (int i = 0; i < 8; i++)
                {
                    Debug.Print("data["+i+"]: "+(recMsg[0].data[i]).ToString()+" ");
                }

            }
```

On envoie ici un message avec pour id D6 (en hexadécimal) avec des données sur 8 bits 1,2,3,4,5,6,7,8.

Puis on attend un message et on affiche ces informations le stockage des informations se fait toujours dans l'ordre suivant:

Id, DLC(taille des données), Données. Il suffit donc d'aller chercher les informations à la bonne place dans le tableau grâce à .ArbID ou .data[i], mais on peut aussi savoir si EID ou RTR sont à vrai ou faux avec recMsg[0].isEID par exemple.

Remarque

- Il n'a pas été possible de configurer le bus Can de la carte à 500Kb/s, malgré les indications données par GHI. Le test sur la voiture prévu par le planning n'a donc pas pu être réalisé.
- La déclaration de tableau pour les messages Can est un peu déroutant, mais il est imposé par la Class de GHI.

9. Module WiFI

9.1. Configuration

Un module Wifi optionnel peut être adjoint à la carte EMX afin d'accroitre sa connectivité. Le but de cette partie sera de connecter la carte EMX à un routeur WiFi.

La Class WiFi développer par GHI permet une connexion à presque tous les réseaux WiFi puisqu'elle accepte tous les types d'encryption .

Microsoft fournis avec son framework une suite de logiciels, dont un permet la configuration réseau de la carte. Il se trouve dans **C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.1\Tools\Employeuse** . Grâce à ce logiciel il va nous être possible de configurer l'adresse mac que la carte devra utiliser, son adresse ip, le réseaux auquel elle devra se connecter etc. Vous trouverez les options de configuration dans **Target → Configuration → Network.**

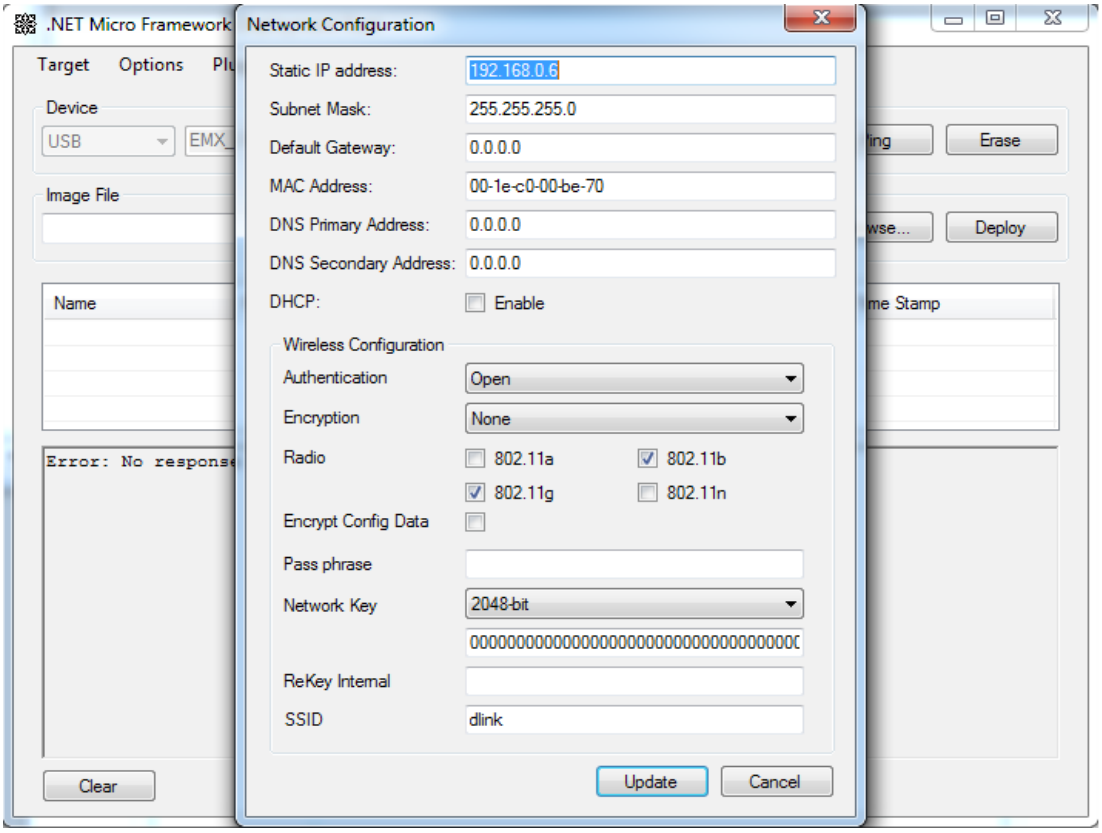


Illustration 12: Écran de configuration réseau de MFDeploy.exe

Vous pouvez voir ici toutes les informations qu'il faudra configurer pour la connexion, attention toutefois de ne pas activer le DHCP puisque celui-ci ne semble pas fonctionner. Les informations tels que l'adresse ip, le ssid, l'encryptions, le passphrase pourrons également être configurés dans le programme grâce à la class WiFi.

9.2. Programmation

```
namespace MFWindowApplication5
{

    public sealed class Class1
    {
        static public bool wifi_event = false;
        static public bool wifi_last_status = false;
        static public bool network_is_read = false;

        static public ManualResetEvent NetworkAvailablityBlocking = null;
        public int Flag;
        //Création d'une méthode Connect()
        public int Connect()
        {

            if (!WiFi.IsEnabled)
            {
                WiFi.Enable(SPI.SPI_module.SPI2, (Cpu.Pin)33, (Cpu.Pin)19); //
Configuration des entrées/sortie à utilisé
                                                                            //pour la
gestion du module Wifi.

            }

            // WiFi configuration, donnée par GHI.
            NetworkInterface[] netif = NetworkInterface.GetAllNetworkInterfaces();
            Wireless80211 WiFiSettings = null;
            for (int index = 0; index < netif.Length; ++index)
            {
                if (netif[index] is Wireless80211)
                {
                    WiFiSettings = (Wireless80211)netif[index];

                }
            }

            if (WiFiSettings.Ssid != "dlink") // Configuration de l'AP.
            {
                WiFiSettings.Ssid = "dlink";
                WiFiSettings.PassPhrase = "";
                WiFiSettings.Encryption = Wireless80211.EncryptionType.None;
                Wireless80211.SaveConfiguration(new Wireless80211[] { WiFiSettings },
false); //Sauvegarde de la configuration

//pour le prochain reboot de la carte

            }
```

Suite du code précédent.

```
  NetworkAvailablityBlocking = new ManualResetEvent(false);
            if (!WiFi.IsLinkConnected)
            {
                Debug.Print("Recherche du WiFi");
                NetworkAvailablityBlocking.Reset();
                while (!NetworkAvailablityBlocking.WaitOne(5000, false))
                {
                    if (!WiFi.IsLinkConnected)
                    {
                        Debug.Print("Connexion échoué");
                        Flag = 0;
                        return Flag;
                    }
                    else
                        break;
                }
            }
            Flag = 1;
            Debug.Print("Connexion réussie");
            Debug.Print("IP: " + WiFiSettings.IPAddress);


            //Thread.Sleep(Timeout.Infinite);
            return Flag;
        }
    }
}
```

J'ai crée une class spécifique pour la configuration et la connexion du Wifi avec une seule méthode Connect() cette class est dans fichier Class1.cs

Sur la page suivant vous trouverez une partie du programme utilisant cette class.

```
private void OnButtonUp(object sender, RoutedEventArgs evt)
        {
            ButtonEventArgs e = (ButtonEventArgs)evt;


            Debug.Print(e.Button.ToString());
            //test du bouton gauche
            if (e.Button == Button.VK_LEFT)
            {
                int connexion;
                //Declaration d'un nouvelle objet
                Class1 test = new Class1();
                //On lance la connexion et on stock la valeur de retour dans connexion
                connexion = test.Connect();
                //On test la valeur et on affiche le résultat à l'écran.
                if (connexion == 1)
                {
                    Text text = new Text();
                    text.Font = Resources.GetFont(Resources.FontResources.small);
                    text.TextContent =
Resources.GetString(Resources.StringResources.String2);
                    text.HorizontalAlignment =
Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
                    text.VerticalAlignment =
Microsoft.SPOT.Presentation.VerticalAlignment.Center;
                    mainWindow.Child = text;
                }
                else
                {
                    Text text = new Text();
                    text.Font = Resources.GetFont(Resources.FontResources.small);
                    text.TextContent =
Resources.GetString(Resources.StringResources.String3);
                    text.HorizontalAlignment =
Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
                    text.VerticalAlignment =
Microsoft.SPOT.Presentation.VerticalAlignment.Center;
                    mainWindow.Child = text;
                }

            }
        }
```

J'utilise le code de base généré par la création d'une « Windows application » je récupère ensuite l'appuie sur le bouton gauche afin de lancer la connexion et d'afficher à l'écran de la carte le statut de la connexion.

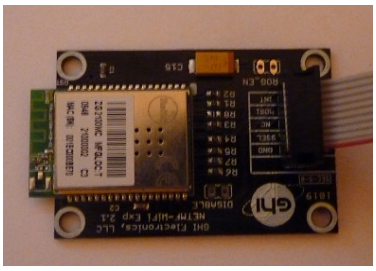


Illustration 13: Module additionnel Wifi

Remarque

- J'ai du réaliser ces tests avec un autre routeur WiFi que celui de l'univ-tours puisque ce dernier n'accepte pas les connexions sans le serveur DHCP activé.
- Avec cette partie de code j'ai pu comprendre le fonctionnement des class en Csharp et de ce fait réaliser un programme propre et adaptable.

Conclusion

J'ai pu constater au cours de ce projet que toutes les fonctionnalités de la carte ne sont pas exploitables, ceci est principalement du à la jeunesse du framework. En effet le .NET Micro Framework 4.1 venant « tout juste » de sortir, GHI n'a pas encore tout à fait prit en charge ce framework. Ce qui par exemple empêcher de tester le bus i²c car les fonctions étaient adaptées à l'ancien framework et n'avaient pas été mises à jour. De plus, plusieurs documentations présentent des explications et du code plus adapté au nouveau framework, il est donc difficile de trouver de la document à jour pour cette carte. Ces problèmes seront résolus avec le temps.

Le fait d'apprendre le Csharp en pratiquant et sans base concrète n'a pas été facile et m'a ralentit dans ce projet, en effet le Csharp est un langage de très haut niveau, très complet et par conséquent il aurait fallu beaucoup plus de temps pour pouvoir l'exploiter au mieux malgré les nombreuses soirées et weekends passés à me documenter. La gestion de l’écran tactile en particulier est relativement ardu.

Le cahier des charges à globalement été rempli, je n'ai pas pu réaliser les tests sur la voiture, ni fais d'interface graphique pour regrouper tous les bouts de code par manque de temps et de connaissance. J'ai pu parcontre tester le port série via un module Xbee. Le fait de travailler sur ce projet m'a conforter dans mon choix de m'orienter vers de la programmation de système embarqué pour mes études futures.

Résumé

Nous avons pu voir dans le début de ce document la configuration de Visual Studio et la façon de le configurer afin de pouvoir créer un projet compatible avec la carte de développement. Mais également quelques notions de Csharp et .NET Micro Framework.

La conception graphique d'une application pour l'embarqué se décompose en deux phases. Un travail préalable de design afin de créer des icônes, des boutons ou d'autres éléments d'interaction homme/machine d'une part, et un travail de programmation d'autre part, puisqu'il faut ensuite relier ces objets aux diverses interactions, comme l'écran tactile ou les boutons.

Les entrées analogiques sont relativement simples à utiliser comme sur tout les micro-processeurs. Il est d'ailleurs possible sur cette carte de définir la plage de conversion du convertisseur analogique-numérique.

La carte nous permet également d'utiliser le bus Can relativement simplement via les constructeurs et les méthodes fournies par GHI. Il m'a donc été possible de la tester grâce au dongle CANUSB et au logiciel Can monitor pro.

Enfin la partie programmation du module WiFI, fut la partie la plus délicate à comprendre et à mettre en œuvre, de plus il a fallu utiliser un routeur spécifique pour la connexion de la carte puisque celui de l'univ-tours n'acceptait pas une adresse ip fixée.

La mise en œuvre du bus i²c n'a pas pu être effectuée faute de class compatible avec le nouveau framework

231 mots

Index des illustrations

Bibliographie

[0].<http://www.thierry-lequeu.fr/data/DATA434.HTM> (septembre2010).
[1].<http://www.ghielectronics.com/product/129> (septembre2010).
[2].<http://www.nxp.com/documents/data_sheet/LPC2478.pdf> (novembre2010).
[3].<http://uuu.enseirb.fr/~kadionik/formation/canbus/canbus_enseirb.pdf> (octobre2010).
[4].<http://www.lextronic.fr/P466-module-canusb.html> (octobre2010).

Annexes

Prévisionnel/Réel	37		38		39		40		41		42		43		44		45		46	
Apprentissage du langage C#	X	X	X	X	X	X		X		X		X		X		X		X		X
Développement de l'application test					X	X														
Test et prise en main de la carte EMX			X	X	X	X	X	X												
Développement application bus CAN							X		X	X		X		X						
Tests pratiques avec voiture test									X											
Développement application I/O											X	X	X	X				X		X
Mise en œuvre WFI													X		X		X			
Développement de l'application finale																	X		X	
Test et modifications																X		X	X	X