11/02/2018

FASCICULE 1
MESURAGE
AFFICHAGE
COMMANDE
STOCKAGE
CLES EN MAIN !

.NETMF V4.3 - C# : EXEMPLES POUR LES CARTES NETDUINO

1.0.0

En cours de rédaction

philippemariano@gmail.com

Table des matières

Préface

Dérivé du Framework .NET de la société Microsoft, le Micro Framework .NET, en abrégé **NETMF**, est adapté au monde de l'embarqué et plus spécifiquement aux appareils ayant les ressources les plus restreintes. NETMF permet d'exécuter directement du code C# sur un microcontrôleur.

L'intérêt de cette technologie tient à :
- la simplicité du langage C#,
- la puissance et la convivialité de l'environnement de développement intégré (IDE) Visual Studio,
- la possibilité de réutiliser le code avec des cibles différentes (Netduino, BrainPad, FEZ PANDA, Gadgeteer, etc.) .

Ce qui rend parfois sa mise en œuvre un peu ardue :
- les exemples de code dispersés au sein des différentes communautés anglo-saxonnes.

Si vous ne connaissez pas le langage C#, il vous paraîtra verbeux. Ne vous laissez pas abuser, car il s'agit plutôt d'une expression de sa clarté. D'ailleurs l'IDE Visual Studio l'écrit pour vous ! Rien n'étant caché, les concepts de programmation-objet, de classe et d'instanciation sont parfaitement lisibles.

Ce premier fascicule est un recueil de programmes écrit en C#. Ils ciblent une carte Netduino de la société WILDERNESS LABS à laquelle sont connectés divers capteurs et actuateurs.
Ces programmes pourront être adaptés à d'autres cartes de développement telles que celles des familles Gadgeteer, FEZ et Brainpad de la société GHI Electronics.

Un deuxième fascicule est consacré à la communication sur un réseau local et à la réalisation de prototypes d'objets connectés.

N'hésitez pas à me contacter si vous avez des questions, des remarques, des critiques ou si vous avez décelé des erreurs.

Philippe Mariano

Se repérer dans le document

- La **partie 1** présente les principaux matériels et les logiciels utilisés dans la suite du document.
- La **partie 2** est un tutoriel décrivant la mise en œuvre du logiciel **Visual Studio 2015** et la programmation d'une carte Netduino.
- La **partie 3** rassemble divers exemples de code écrits en C# dans l'environnement de développement intégré Visual Studio.

Utiliser le document

- **Pour prendre en main l'IDE Visual Studio**
  Suivez le tutoriel. Il couvre la création et la configuration d'un projet, l'écriture du code d'un programme, son transfert dans la carte et son débogage in situ.
  Note : Une version en ligne, régulièrement mise à jour est disponible ici. [https://goo.gl/by4obJ]

- **Utiliser une fiche "Exemple"**
  - **Trouver un exemple illustrant un thème particulier ou l'utilisation d'un capteur dans le document**
    Un exemple décrivant un thème ou la mise en œuvre d'un capteur peut être trouvé à partir d'un titre dans la table des matières, d'un mot clé (nom de composant, fonctionnalité…) dans l'index ou d'une description en consultant l'**annexe 1**.

  - **Se repérer dans une fiche "Exemple"**
    L'organisation des fiches est décrite dans le chapitre 6.

  - **Analyser les différentes parties du code d'un exemple**
    Les différentes parties du code d'un exemple sont numérotées. Les numéros permettent de se repérer dans les commentaires qui l'accompagnent.

- **Tester le code d'un "Exemple"**
  L'implantation d'un programme dans la carte de développement est décrite dans le tutoriel. Certains répertoires "Projet Visual Studio", dont un récapitulatif est donné en *annexe 1*, contiennent un fichier "lisez-moi" et un répertoire "Doc_A_Consulter". Le fichier "lisez-moi" apporte des informations complémentaires. Le répertoire "Doc_A_Consulter" contient les documentations et la photo du montage à réaliser.

Ce qu'il est préférable d'avoir lu avant de tester les exemples de programme

- Le paragraphe 1.3 ci-dessous.
- Le chapitre 2 "Avant de commencer".
- Le chapitre 6 "Organisation d'une fiche exemple".

1 Avant de commencer

1.1 Introduction

Le Micro Framework .Net est un sous-ensemble du Framework .Net créé par **Microsoft open source**. **Netduino** est une plateforme open source, distribué par la société **WILDERNESS LABS**.
Les cartes de développement **Netduino** se positionnent entre une Arduino Uno et une Raspberry pi.
La programmation de leur **processeur 32 bits** ne repose pas sur un système d'exploitation comme Linux ou Windows 10 IoT mais sur le micro framework .**NET**. Il est possible de développer des applications capables d'accéder rapidement au matériel (comme des capteurs reliés à un bus I^2C) et de mettre en œuvre des fonctionnalités de plus haut niveau (accès à des fichiers, services web, etc.) avec une **approche multi threads** si nécessaire.

La véritable force de l'écosystème NETMF est le micro framework .NET, associé à l'IDE Visual Studio. Le micro framework .NET est un environnement de développement **open source**, complet et bien documenté. NETMF contient des fonctionnalités avancées telles que : les **protocoles de communication** (HTTP, SMTP, etc..), **la gestion de fichiers, XML, des bibliothèques graphiques, le multithreading**, un **émulateur configurable** pour simuler des capteurs, des bus de communication, etc.

1.2 Public visé

Ce recueil est destiné à des élèves préparant un baccalauréat S avec l'option sciences de l'ingénieur ou l'option ISN et aux élèves préparant un baccalauréat STI2D avec l'option Systèmes d'Information et Numériques.
Les exemples ont été rédigés pour servir de point de départ à la mise en œuvre de l'environnement .Net sur des cartes de prototypage rapide lors des travaux pratiques ou des projets.

Ce document pourra également servir aux enseignants de ces disciplines, désireux de se familiariser avec l'environnement .Net.

1.3 Organisation du document

Ce document se compose de trois parties :
- la première fait l'inventaire du **matériel et des logiciels**,
- la deuxième est un **tutoriel** permettant de "prendre en main" l'environnement de développement Visual Studio avec comme objectif d'implanter un programme sur une carte Netduino,
- la troisième constitue un **recueil d'exemples** documentés.

Les programmes sont écrits en **C#.** Ils permettent d'**acquérir** les informations délivrées par des capteurs analogiques ou numériques, de les **sauvegarder**, de les **traiter** et de les **communiquer** à des dispositifs d'affichage, ou à des préactionneurs.

Les programmes ont été compilés avec l'IDE **Visual Studio 2015.** Ils s'appuient sur le **.Net Micro Framework SDK v4.3** et le **Netduino SDK v4.3.2.1**. Ils ciblent les cartes **Netduino plus 2 ou Netduino 3.**

Les sources peuvent être recompilées pour d'autres cartes programmables en C# avec le micro framework .NET telles que les Gadgeteers et les cartes FEZ de la société **GHI Electronics**.

Les projets utilisant des classes n'appartenant pas au micro framework .NET sont maintenus sur **github**.

Ces classes sont rassemblées dans le NuGet "**MicroToolsKit**"

En annexe 1, un tableau associe la référence de l'exemple, le nom du répertoire du projet Visual Studio et un bref descriptif. Il précise également s'il existe un fichier "Lisezmoi.txt" décrivant le projet et des photos du montage. Il précise enfin si des classes spécifiques au projet sont maintenues sur **github**.

Exemple

	Visual Studio	**Description (CI ou module)**				
I2C_1	NetduinoPCF8574	I²C : Chenillard sur huit LED reliées à un port d'E/S **PCF8574**	X	☒	X	X

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio 2015.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

1.4 Téléchargements et liens

Les sources des projets sont accessibles au téléchargement sur le site STI de l'académie d'Orléans-Tours ou sur le wiki ***C# Embarqué Netduino***.

[http://goo.gl/7hxkKl]

Les classes développées pour les matériels utilisés dans ce document sont maintenues sur Github.

[https://goo.gl/DJXqs3]

Le wiki ***C# Embarqué Netduino*** pour télécharger les exemples de code et les fascicules**.**

[https://goo.gl/D6rcpv]

Le NuGet MicroToolsKit, téléchargeable à partir Visual Studio, rassemble les classes maintenues sur github.

[https://goo.gl/DJQKKh]

Le fil d'information ***C# embarqué : Netduino, GHI Fez, Quail (.NetMF)*** rend compte de l'actualité.

[https://goo.gl/2CQzNG]

1.5 Questions fréquentes

- **Généralités**

1. **Qu'est-ce qu'une carte Netduino ?**

   ***R** : Les cartes de prototypage rapide Netduino facilitent la réalisation d'applications multi threads dans le but d'acquérir, de traiter, de stocker, et de communiquer des données.*

- **Questions liées au matériel**

2. **Comment connecter une carte Netduino à un PC ?**
   ***R** : avec un câble USB type A mâle d'un côté et Micro USB Type B mâle de l'autre.*

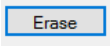
3. **Comment alimenter une carte Netduino ?**
   ***R** : Si l'énergie demandée par les périphériques branchés à la carte n'est pas trop importante, on peut se contenter du câble de programmation sinon il faut utiliser une alimentation universelle 1A ou 2A réglée sur 9V.*

4. **Comment connecter un capteur, un afficheur ou un pré actionneur à une carte Netduino ?**
   ***R** : Comme sur une carte Arduino. La carte Netduino est compatible avec la connectique d'une Arduino V3. Dans ce document les connexions se font par l'intermédiaire d'un shield Tinkerkit et de connecteurs KK3 ou KK4.*

5. **Quelles sont les précautions à prendre ?**
   R : Ne pas alimenter la carte avec une tension supérieure à 9V. Ne pas brancher ou débrancher les capteurs, les préactionneurs et les afficheurs sous tension.

6. **Comment effacer le programme dans le composant ?**
   ***R :** Utiliser l'utilitaire **MFDeploy** situé dans program -> Microsoft .Net Microframework 4.x*

   Régler Device sur USB puis clic sur 

- **Questions liées aux logiciels**

7. **Quel langage utilise-t-on pour programmer les cartes Netduino ?**
   ***R** :* **C#** (c'est le langage utilisé dans ce document) ou **Visual Basic**.

8. **Quels logiciels doit-on installer ?**
   ***R** : Visual Studio Community 2015, le .NET Micro Framework SDK v4.3* et son plug-in pour Visual Studio 2015 .NET MF plug-in for VS2015 *et le Netduino SDK v4.3.2.1 sur le PC. Il faut éventuellement faire une mise à jour du firmware de la carte Netduino. Un guide d'installation en français des outils logiciels (régulièrement mis à jour) est disponible au téléchargement ici.[http://goo.gl/SJI9O1]*
   *Le site WILDERNESS LABS (*https://www.wildernesslabs.co/*) fournit également ces informations (en anglais).*

   Remarque : Visual Studio Community 2015 est configuré en français à partir du menu Outils -> Options -> Paramètres internationaux -> Langue.

9. **Comment écrire, transférer et déboguer un programme ?**
   ***R** : Le chapitre 4 "Débuter avec C# et Visual Studio pour programmer une carte Netduino" constitue une première approche.*

10. ***Où peut-on trouver la documentation du micro framework .NET ?***
    ***R** : Un tableau des principales classes est donné en annexe 2. Celles-ci sont décrites sur le site msdn. [https://goo.gl/dIvfxy]*
    Dans l'IDE Visual Studio, sélectionner un mot dans le programme puis F1.

1.6 Problèmes rencontrés

- **Visual studio ne parvient pas à programmer la carte**

1. Il y a des erreurs dans le code.
   ***Solution** : corriger les erreurs*

2. La liaison ne s'établit pas
   ***Solution** : Appuyer sur le bouton Reset de la carte -> Lancez le transfert -> Relâcher le bouton Reset*

3. Le firmware de la carte est altéré.
   ***Solution** : réinstaller le firmware. Un guide d'installation en français des outils logiciels (régulièrement mis à jour) est disponible au téléchargement ici.[http://goo.gl/SJI9O1]*
   *Le site WILDERNESS LABS (*https://www.wildernesslabs.co/*) fournit également ces informations (en anglais).*

2 Les matériels

2.1 Les cartes de prototypage rapide

Une carte de prototypage rapide permet de valider l'étude d'un système électronique numérique sans développement matériel.
Quelques cartes "grand public" en 2017.

Figure 1 : Arduino Uno Rev3

Figure 2 : Netduino plus 2

Figure 3 : Raspberry pi V2

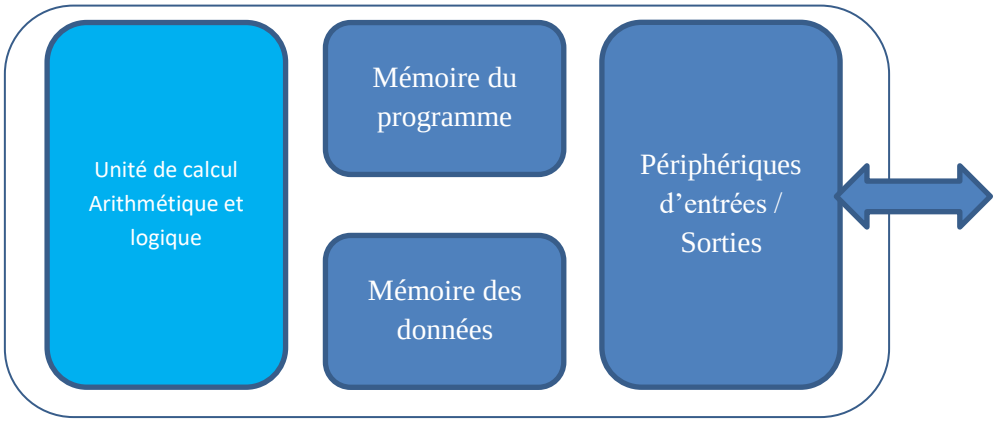
Une carte de prototypage rapide comprend généralement :
- un **microcontrôleur** dans lequel le programme sera implanté,
- une **connectique** lui permettant de communiquer avec des capteurs et des actuateurs via des ports d'entrées / sorties analogiques et numériques.

Pour les plus évoluées :
- une carte µSD,
- une liaison Ethernet et/ou wifi à 10/100 Mb,
- des ports USB,
- une sortie vidéo.

2.2 Le microcontrôleur

« Un microcontrôleur (en notation abrégée **µc**, ou **uc** ou encore **MCU** en anglais) est un *circuit intégré* qui rassemble les éléments essentiels d'un *ordinateur* : *processeur*, *mémoires* (*mémoire morte* pour le programme, *mémoire vive* pour les données), unités périphériques et interfaces d'*entrées-sorties*. » Source Wikipédia

Représentation fonctionnelle simplifiée



2.3 La connectique

Sur une carte de prototypage rapide, l'accès aux périphériques d'entrées / sorties se fait par l'intermédiaire de connecteurs. Il est ainsi possible de câbler des capteurs et des actuateurs soit directement, soit par l'intermédiaire de cartes d'extension (shields) .

3 Les cartes de prototypage Netduino

La carte Netduino plus 2 a la même configuration de broches que l'Arduino Uno Rev3. Elle est compatible avec un grand nombre de shields.

Un rapide comparatif entre une Arduino et une Netduino fait apparaître la puissance de la seconde par rapport à la première.

	Arduino Uno	Netduino 3
Fréquence CPU	(8bits) 16Mhz	(32bits) 168Mhz
SRAM	2kB	164+ kB
Flash	32kB	1408 kB
Réseau	Non	Ethernet 10MB
Prix (2017)	20€	65€ (*Mouser*)

La gamme complète des matériels est disponible sur ici.

Caractéristiques des principales cartes (celles sur lesquelles les exemples ont été testés sont entourées ci-dessous)

mainboard	netduino 3	netduino 3 ethernet	netduino 3 wi-fi
			
	more information »	more information »	more information »
processor and memory			
microcontroller	STMicro STM32F4	STMicro STM32F4	STMicro STM32F4
speed	168 MHz (Cortex-M4)	168 MHz (Cortex-M4)	168 MHz (Cortex-M4)
code storage	384 KB	1408 KB	1408 KB
ram	164+ KB	164+ KB	164+ KB
operating system	.NET Micro Framework 4.3	.NET Micro Framework 4.3	.NET Micro Framework 4.3
input and output			
networking	none	ethernet: 10/100 mbps	wi-fi: 802.11b/g/n including SSL/TLS 1.2 and WPA2
arduino shield compatibility	works with most arduino shields (some require .net mf drivers)	works with most arduino shields (some require .net mf drivers)	works with most arduino shields (some require .net mf drivers)
digital i/o	22 gpio, 6 pwm, 4 uart, i2c, spi	22 gpio, 6 pwm, 4 uart, i2c, spi	22 gpio, 6 pwm, 4 uart, i2c, spi
analog inputs	6 adc channels (12-bit)	6 adc channels (12-bit)	6 adc channels (12-bit)
storage	add-on: sd shields (up to 2 gb)	micro sd (up to 2 gb)	micro sd (up to 2 gb)
gobus ports	3 gobus ports	3 gobus ports	3 gobus ports

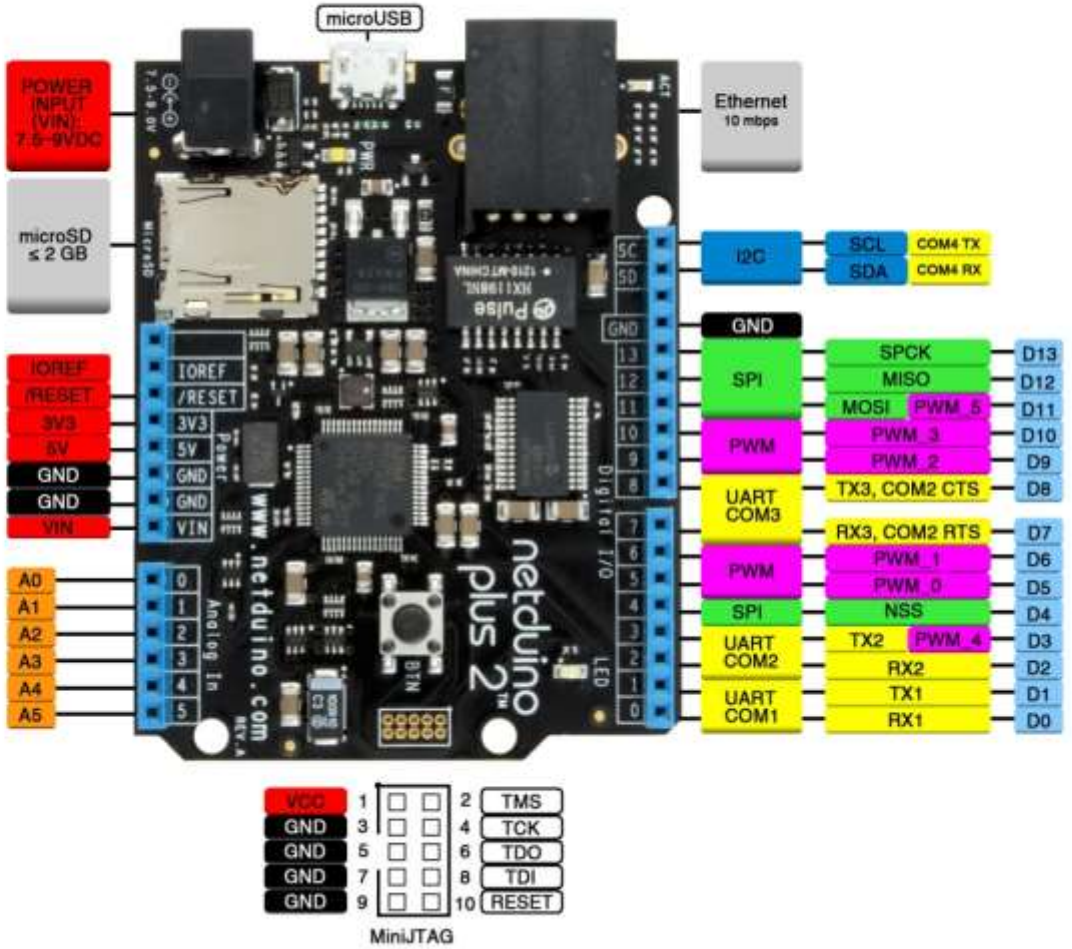
GPIO: General Purpose Input / Output port

Netduino Plus2

Version: 4.2.1.0

Processor and memory
* STMicro 32-bit microcontroller
* Speed: 168MHz, Cortex-M4
* Code Storage: 384 KB
* RAM: 100+ KB

Power
* if VIN is present, powered by VIN
* if VIN is not present, powered by USB
* maximum GPIO current: 25mA per pin
* maximum mcu output current: 125mA
* output: 5 VDC and 3.3 VDC regulated
* input/output: VIN is unregulated
* total max current
  (incl. GPIO, power rail, microSD, Ethernet):
  - ~500 mA @ 5 V (USB)
  - ~380 mA @ 7.5 V (VIN)
  - ~300 mA @ 9 V (VIN)
  - ~250 mA @12 V (VIN)
* digital i/o are 3.3 V--but 5 V tolerant
* IOREF: 3.3 V (allows shields to adapt to the voltage provided from the board)
* 12-bit ADC: converts an anolog input voltage from 0 V to IOREF to a digital from 0 to 4095
* erase pad has ben removed



gutworks

Version 1.2-Nov 27, 2012

Figure 4 : Identification des E/S de la carte Netduino Plus 2 (ou 3)

3.1 Le module Sensor Shield V2 Tinkerkit

Le module Sensor Shield V2 TinkerKit se connecte **sur** la carte Netduino plus 2 ou une carte Netduino 3. Il permet de raccorder facilement et sans soudure des capteurs et des actuateurs.

Exemple : Un capteur analogique raccordé au connecteur **IO du shield Tinkerkit** sera accessible sur la **broche 0** du connecteur **Analog In** de la carte **Netduino**. Son identification dans un programme C# (.NetMF) sera réalisé par **Cpu.AnalogChannel.ANALOG_0 (voir les tables ci-dessous).**

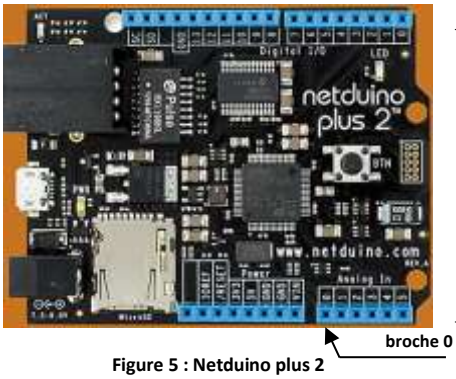


Figure 5 : Netduino plus 2

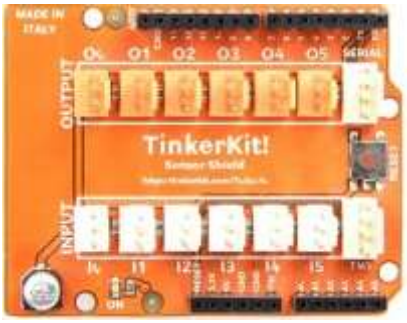


Figure 6 : Sensor Shield V2 Tinkerkit

3.2 Tables de correspondance des dénominations Netduino, Tinkerkit, IDE Visual Studio

Entrées / Sorties Numériques [Connecteur *Digital I/O*]

Connecteur **Digital I/O** sur la Netduino		**Shield Tinkerkit**	**Identification dans un programme C#**
GPIO n°	Secondary Features		
SC	SCL/UART 4TX	**TWI**	| v
SD	SDA/UART 4 RX		
13	SPCK	-	Pins.GPIO_PIN_D13
12	MISO		Pins.GPIO_PIN_D12
11	PWM / MOSI	**O0**	Pins.GPIO_PIN_D11
10	PWM	**O1**	Pins.GPIO_PIN_D10
9	PWM	**O2**	Pins.GPIO_PIN_D9
8	UART 3 TX / UART 2 CTS	-	Pins.GPIO_PIN_D8
7	UART 3 RX / UART 2 RTS		Pins.GPIO_PIN_D7
6	PWM	**O3**	Pins.GPIO_PIN_D6
5	PWM	**O4**	Pins.GPIO_PIN_D5
4		-	Pins.GPIO_PIN_D4
3	UART 2 TX / PWM	**O5**	Pins.GPIO_PIN_D3
2	UART 2 RX	-	Pins.GPIO_PIN_D2
1	UART 1 TX	**Serial**	Pins.GPIO_PIN_D1
0	UART 1 RX		Pins.GPIO_PIN_D0
-	LED	-	Pins.GPIO_ONBOARD_LED
	BP		Pins.GPIO_ONBOARD_SW1

Entrées / Sorties Numériques [Connecteur *Analog Input*]

Connecteur **Analog In** n° sur Netduino	**Shield Tinkerkit**	**Identification dans un programme C#**
0	**I0**	Pins.GPIO_PIN_A0
1	**I1**	Pins.GPIO_PIN_A1
2	**I2**	Pins.GPIO_PIN_A2
3	**I3**	Pins.GPIO_PIN_A3
4	**I4**	Pins.GPIO_PIN_A4
5	**I5**	Pins.GPIO_PIN_A5

Entrées Analogiques [Connecteur *Analog Input*]

Connecteur **Analog In** n° sur Netduino	**Shield Tinkerkit**	**Identification dans un programme C#**
0	**I0**	Cpu.AnalogChannel.ANALOG_0
1	**I1**	Cpu.AnalogChannel.ANALOG_1
2	**I2**	Cpu.AnalogChannel.ANALOG_2
3	**I3**	Cpu.AnalogChannel.ANALOG_3
4	**I4**	Cpu.AnalogChannel.ANALOG_4
5	**I5**	Cpu.AnalogChannel.ANALOG_5

Précautions à prendre lors de la configuration des entrées / sorties numériques.

Utiliser **OBLIGATOIREMENT** les classes WILDERNESS LABS comme cela est décrit dans les exemples sous peine de « planter » le firmware de la carte (il devra alors être réinstallé !).

Le « Guide d'installation des logiciels » est téléchargeable ici.

4 Les logiciels

4.1 Présentation

La véritable force de l'écosystème NETMF est l'association du Micro Framework .NET et de l'IDE Visual Studio. Le Micro Framework .NET est un environnement de développement complet et bien documenté. NETMF contient des fonctionnalités avancées telles que les protocoles de communication, la gestion de fichiers, XML, des interfaces graphiques, le multithreading, etc.

Les cartes Netduino se programment en **C#** (syntaxe C, langage perçu comme une amélioration de Java) ou en Visual Basic avec l'environnement de développement intégré (IDE) **Microsoft Visual Studio** (disponible dans les versions **Community ou professionnel)**. Il est nécessaire d'installer le Micro Framework .NET correspondant à la version de l'IDE et le SDK WILDERNESS LABS correspondant à la carte ciblée.

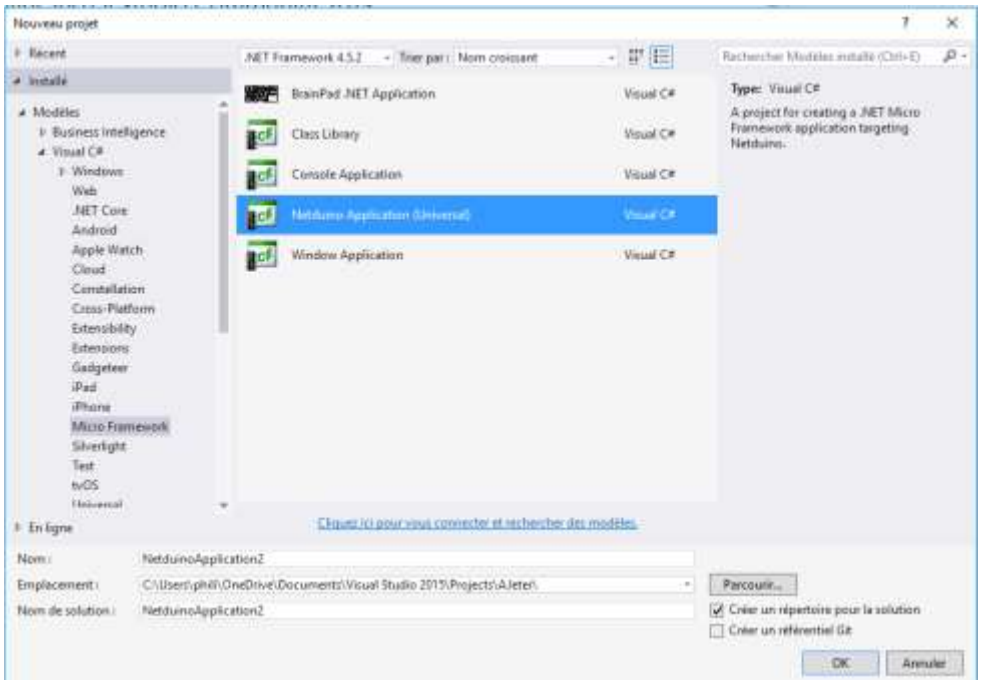


Figure 7 : Visual Studio - Nouveau projet

Fonctionnalités particulièrement appréciables dans l'IDE Microsoft Visual Studio

- Environnement de développement intégré complet pour créer des **applications Web**, **Windows Desktop et cross-plateforme** iOS, **Android** et Windows,
- **Coloration** syntaxique,
- **Autocompletion** (Intellisense),
- **Template** de code,
- **Debugger in situ** (exécution du programme en pas à pas dans la carte avec retour de la valeur des variables dans l'IDE)
- **Etc.**

Microsoft Visual Studio Community 2015

Visual Studio Community 2015

Téléchargez Visual Studio Community 2015, un IDE complet gratuit, des outils de développement mobiles cross-plateforme pour Windows, iOS et Android, et l'accès à des milliers d'extensions. Cette édition de Visual Studio est disponible sans frais pour les étudiants, les contributeurs open source, les startups, les développeurs indépendants et sous certaines conditions les PME.

Pour plus d'informations, consultez le guide du licensing.
Connectez-vous sous 30 jours avec votre compte Microsoft pour enregistrer votre produit.

Microsoft .NET Micro Framework

Netduino SDK

4.2 Installation

Un guide d'installation des outils logiciels régulièrement mis à jour est disponible au téléchargement ici.

5 Un premier programme en C# étape par étape

5.1 Préambule

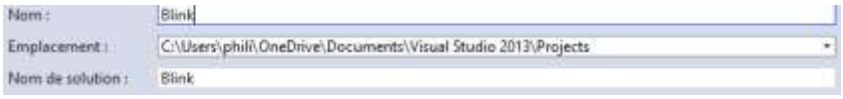
Pour mener à bien ce tutoriel vous devez disposer d'une carte **Netduino plus 2** (ou **Netduino 3**). Les outils logiciels nécessaires à sa programmation doivent être installés sur le PC. Le firmware de la carte doit être à jour. Si ce n'est pas le cas : suivez le « Guide d'installation des logiciels » téléchargeable ici.
Les vidéos du cours sur les **fondamentaux du langage C#**, accessibles sur le site *MVA*. *[http://goo.gl/Uckcib]* sont un excellent préalable ou un complément à ce tutoriel.

5.2 Prise en main de l'IDE Visual Studio

Objectif
Faire clignoter la Led de la carte Netduino !

Étape 1 : Créer un projet

- Ouvrez **l'IDE Visual Studio** en cliquant sur l'icône suivante : puis sélectionnez *Fichier -> Nouveau projet*.
- Dans la boîte de dialogue "*Nouveau projet*", sélectionnez :
  Modèles -> Visual C# -> Micro Framework et Netduino Application (Universal).
- Donnez le nom "*Blink*" à l'application puis cliquez sur Ok. L'emplacement identifié ci-dessous peut changer en fonction de la version du logiciel et de l'arborescence des répertoires du PC.



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

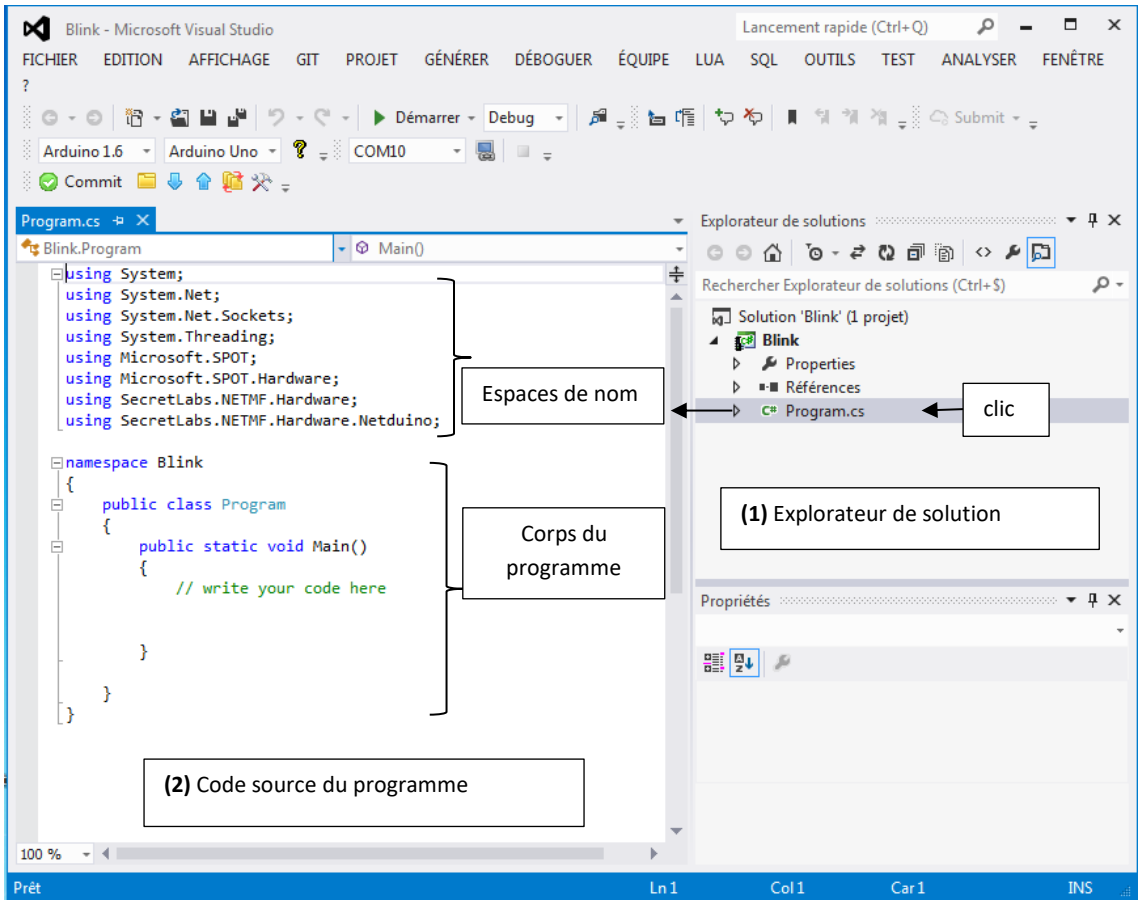


Figure 8 : Visual Studio - Projet

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

Étape 2 : Le code du programme

➢ **Déclarer les "Espaces de noms" et sélectionner les "bibliothèques" pour l'application**

Une partie du code a été renseigné par Visual Studio. C'est le cas de la liste des espaces de nom installés par défaut et de la structure minimum du corps de programme. Nous allons commencer par changer un de ces espaces de nom et la référence vers la bibliothèque lui correspondant.

Pour le moment l'espace de nom suivant : ***SecretLabs.NETMF.Hardware.Netduino*** n'est pas destiné à la bonne cible puisque nous avons une Netduino plus 2 (ou Netduino 3). Modifiez-le en ***SecretLabs.NETMF.Hardware.NetduinoPlus;*** dans le code source du programme.
Visual Studio le souligne en rouge spécifiant qu'il y a un problème. Cela est dû au fait qu'il ne trouve pas la bibliothèque correspondant à ce nouvel espace de nom. Une autre manipulation est nécessaire !

Dépliez le répertoire *Référence* dans l'explorateur de solution et supprimez *SecretLabs.NETMF.Hardware.Netduino* par un clic droit puis supprimer.

Pour ajouter la nouvelle bibliothèque : effectuez un clic droit sur *Références* puis *Ajouter une référence* et dans la liste sélectionnez : *SecretLabs.NETMF.Hardware.NetduinoPlus.* La nouvelle référence apparaît dans le répertoire *Références* et l'erreur disparaît dans le code.

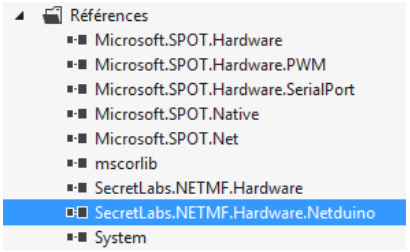


Figure 9 : Visual Studio - Références

➢ **Construire l'objet virtuel "LED"**

Dans la zone du code source, entrez la ligne ci-dessous à la place de : *// write your code here*

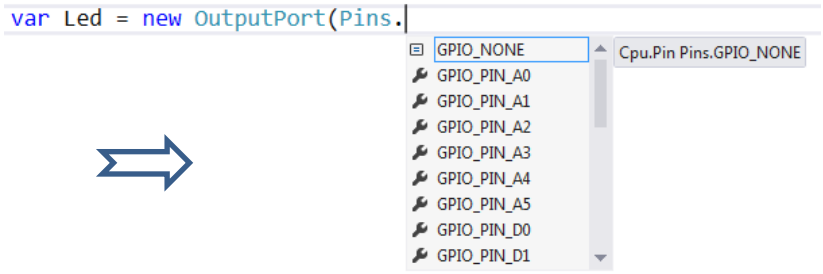
```
var LED = new OutputPort(Pins.ONBOARD_LED, false);
```

Remarque : Vous pouvez constater l'effet de l' **autocompletion** (**intellisense**) au fur et à mesure de la construction de la ligne. 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.

Signification de quelques icônes

	Classe
	Constante
	Enumération
	Méthode ou fonction
	Mot clé du langage
	Propriété

Exemple :



➢ **Écrire le corps du programme dans une boucle infinie**

Écrivez le mot *while* à la suite de la ligne précédente. Le logiciel vous propose un extrait de code (**template)**.

Appuyez **deux fois** sur la touche de tabulation. Le logiciel construit la structure while pour vous.

Complétez la structure ***while*** avec le code ci-dessous (les commentaires // peuvent être omis) :

```
while (true) // Constitue une boucle infinie
{ // début du bloc de code : ici la boucle while
  Led.Write(true); // Eclairer la Led
  Thread.Sleep(500); // pendant 0,5s
  Led.Write(false); // Eteindre la Led
  Thread.Sleep(500); // pendant 0,5s
} // fin du bloc de code : ici la boucle while
```

Étape 3 : Générer la solution

Dans l'éditeur, sélectionner Générer -> Générer la solution. S'il n'y a pas d'erreurs dans le code le logiciel indique : La génération a réussi (en bas et à gauche de la fenêtre). Vous allez maintenant pouvoir transférer le programme dans la carte Netduino plus 2.

Étape 4 : Transférer le programme et l'exécuter

- **Afficher la fenêtre de sortie**

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

  Dans l'éditeur, sélectionner : **Affichage -> Sortie**

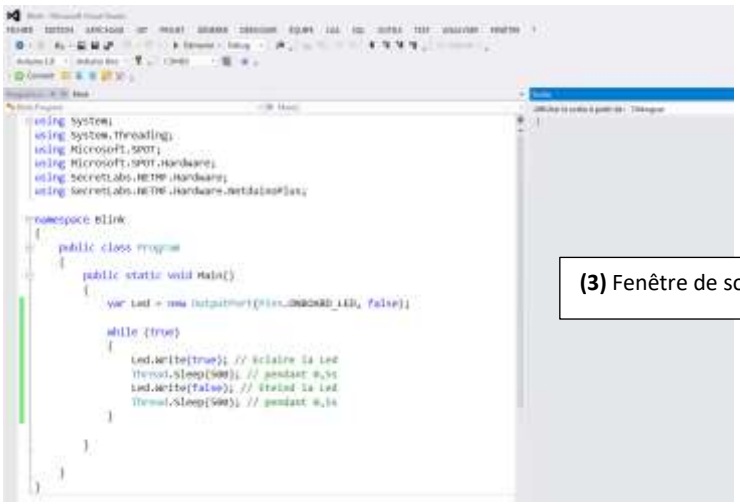


Figure 10 : Visual Studio - Fenêtre de sortie

- **Transférer et exécuter le programme dans la carte Netduino**

  Vérifiez que l'éditeur est en mode Debug. Pour transférer et exécuter le programme dans la carte, cliquez sur Démarrer ou appuyez 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.

F5

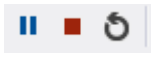
En cas de problème de transfert :

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

La LED de la carte doit clignoter.

Étape 5 : Tester le programme en mode pas à pas

Pour mettre un programme "au point" (déboguer), vous pouvez l'arrêter, le redémarrer ou le mettre en pause.

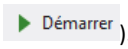
Arrêter
|
v
Mettre en pause ->  **<- Redémarrer**

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

Après avoir arrêté le programme, complétez le code comme ci-dessous.

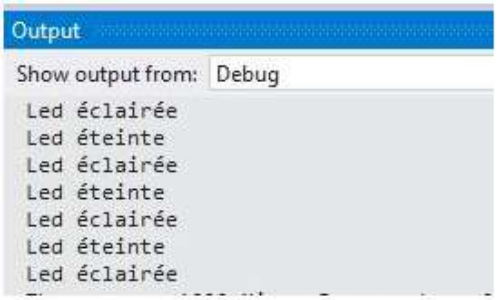
```
while (true)
{ // début du bloc de code : ici la boucle while
  Led.Write(true); /* Eclairer la LED */
  Debug.Print("LED éclairée"); // Affiche le texte entre guillemets dans la fenêtre de sortie
  Thread.Sleep(500); // Attente de 0,5s
  Led.Write(false); // Eteindre la Led
  Debug.Print("LED éteinte"); // Affiche le texte entre guillemets dans la fenêtre de sortie
  Thread.Sleep(500); // Attente de 0,5s
 } // fin du bloc de code : ici la boucle while
```

Placez un **point d'arrêt** en cliquant en face du mot ***while*** comme ci-dessous.

Relancez 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**.

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



Sources de l'exemple

Les sources de l'exemple (compilées avec **Visual Studio 2015 Community**) sont téléchargeables ici.

---

6 Organisation d'une "Fiche Exemple"

Les exemples de programme sont présentés sous la forme de fiches. La position des blocs sur la page peut changer en fonction de la place disponible.

Titre + descriptif

Matériel à utiliser

Connectique

Dans les commentaires du code.

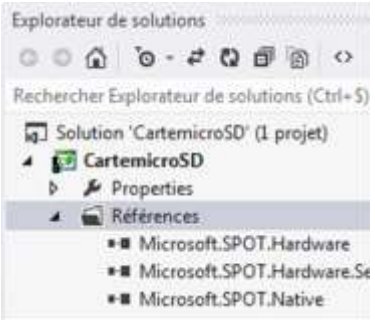
Explorateur de solution

Il doit contenir les références vers les bibliothèques nécessaires au projet.

Pour en ajouter :

Clic droit sur « Référence »

➔ **Ajouter une référence**



➔ **Gérer les packages NuGet**

nuget

MicroToolsKit

Code C#

Organisé comme ci-dessous :

Espaces de nom
…

```
namespace <nom>
{
    // Autres classes, structures, énumérations etc..
    public class Program
     {
        public static void Main()
        {
       // Code de la méthode Main
             }

            //Autres méthodes statiques
      }
}
```

Classes spécifiques au projet

Eventuellement, un lien vers Github où sont maintenues les sources de certaines classes.

github SOCIAL CODING

7 Les entrées, sorties numériques

7.1 Généralités

« Dans un système à base de *microcontrôleur*, on appelle **entrées-sorties** les échanges d'informations entre le processeur et les périphériques qui lui sont associés. De la sorte, le système peut réagir à des modifications de son environnement, voire le contrôler. Elles sont parfois désignées par l'acronyme **I/O**, issu de l'anglais **I**nput/**O**utput ou encore **E/S** pour **e**ntrées/**s**orties. » Source Wikipédia

Pour éviter de faire référence à des valeurs électriques (tension ou intensité), on définit souvent l'état d'un signal numérique en utilisant la logique booléenne.
- **true** (« 1 » logique) correspondra par exemple à 5V ou 3,3V
- **false** (« 0 » logique) correspondra à 0V.

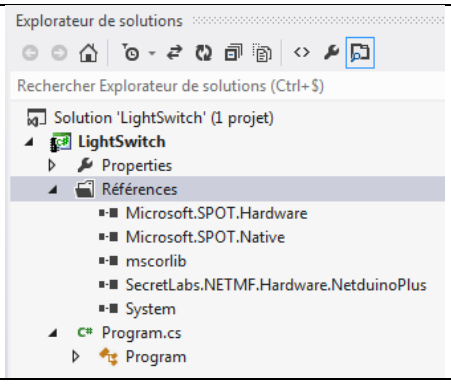
Le nombre de broches d'un microcontrôleur est limité. Il est fréquent d'avoir plusieurs fonctionnalités sur une même broche.

Une sortie numérique conserve la dernière information logique. Elle se comporte comme une **mémoire**.

7.2 Les entrées, sorties numériques des cartes Netduino

Un programme implanté dans une carte Netduino accède à ses entrées, sorties numériques par l'intermédiaire des bibliothèques (assemblys) ci-dessous :

- NETMF : Microsoft.SPOT.Hardware
- Secret Labs : SecretLabs.NETMF.Hardware.NetduinoPlus

Espaces de noms	Références
`using Microsoft.SPOT.Hardware;` `using SecretLabs.NETMF.Hardware.NetduinoPlus;`	

Les connexions **D13 à D0** peuvent être configurées en **entrée ou en sortie numérique**. On voit sur la copie d'écran ci-dessous qu'elles partagent la connectique avec d'autres fonctionnalités (PWM, SPI etc.). Ces fonctionnalités sont abordées dans la suite du document.

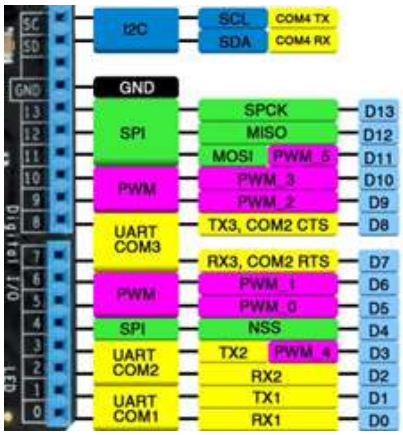


Figure 11 : Netduino plus 2 – Entrées ou sorties numériques

7.3 Les sorties numériques

7.3.1 Précautions d'utilisation

Une sortie numérique délivre **très peu de puissance** (quelques centaines de **mW**). Il n'est donc pas possible de la relier directement à un actionneur (moteur). Il est nécessaire de placer une interface de puissance (hacheur, relais) entre elle et l'actionneur à commander.

7.3.2 Objet logiciel « OutputPort »

7.3.2.1 Construction d'un objet Outputport

L'accès à une sortie numérique "physique" nécessite la construction d'un objet logiciel **OutputPort**. L'état logique de la sortie est contrôlé par une méthode **Write()**.

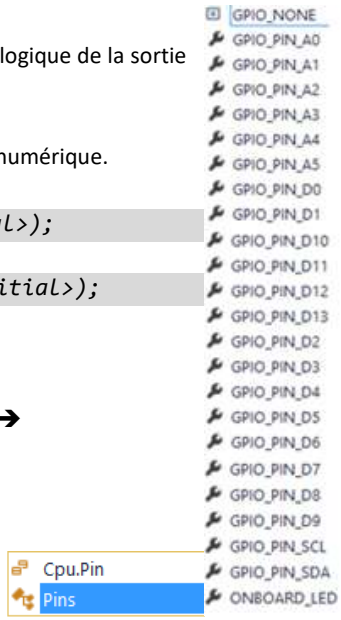
En créant un objet **OutputPort**, le code ci-dessous **configure** une connexion de la carte Netduino en sortie numérique.

```
var <nom> = new OutputPort(<sortie>, <etat_logique_initial>);
```

ou

```
OutputPort <nom> = new OutputPort(<sortie>, <etat_logique_initial>);
```

var et **new** sont des mots-clés du langage C#. **OutputPort** est un mot clé de NETMF**.**
<nom> est le nom que le programmeur attribue à l'objet.
<sortie> est le nom que **WILDERNESS LABS** attribue à la connexion. ➔
<etat_logique_initial> est la valeur que prendra la sortie à la mise sous tension (**true** ou **false**)



Exemple : Initialisation de la LED de la carte

```
var led = new OutputPort(Pins.ONBOARD_LED, false);
```

7.3.2.2 Contrôle de l'état logique d'une sortie numérique

Une sortie numérique peut prendre l'état **true** ou **false**. Cet état est contrôlé avec la méthode Write.

```
<nom>.Write(<etat_logique>);
```

<nom> est le nom que le programmeur attribue à l'objet.
<etat_logique> est la valeur à attribuer à la sortie (**true** ou **false**)

Exemple :

```
led.Write(false);   // Extinction de la LED de la carte
```

7.3.3 Sorties numériques « 3-états » (Tri-state)

Les sorties 3-états sont soit connectées, soit (déconnectées). Dans un système de traitement numérique, il peut être nécessaire de brancher les sorties de différents circuits sur la même ligne (Bus).

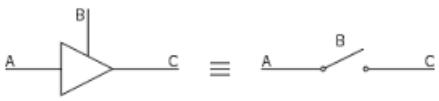


Figure 12 : Sortie trois états

Développement et exemple dans la prochaine révision du document

7.3.4 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
E/S_1	BlinkingLed	- Sortie numérique : faire clignoter la LED de la carte Netduino				
E/S_2a	LightSwitch	- E/S numériques : commander une LED avec un bouton-poussoir.				
E/S_2b	NetduinoClav4Digilent	- E/S numériques : commander une LED avec un clavier **Digilent 4BP**	☒			☒
E/S_3	Netduino_EasyStepperMot	- Sorties numériques : commander un moteur pas à pas avec une carte **EasyStepper Driver Motor V4.4**	☒	☒	☒	

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

Pour plus d'informations, consulter la description de la classe **OutputPort** sur MSDN

7.4 Les entrées numériques

Les entrées numériques permettent de détecter un état logique « 0 » ou « 1 ».

7.4.1 Précautions d'utilisation

Les entrées numériques sont **fragiles**. Elles ne supportent ni les décharges **électrostatiques** ni les **surtensions**. Il ne faut ni les toucher ni leur appliquer une tension supérieure à 5V ou inférieure à 0V. Une entrée numérique utilisée dans un programme ne doit pas être laissée "en l'air" (non connectée) car elle prendra alors un état logique aléatoirement et le comportement du programme deviendra imprévisible.

Exemple : Acquisition de l'état (ouvert ou fermé) d'un bouton-poussoir.

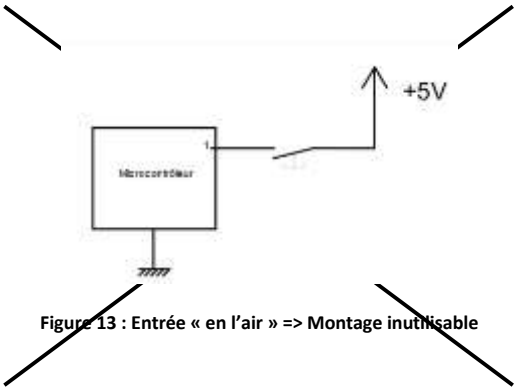


Figure 13 : Entrée « en l'air » => Montage inutilisable

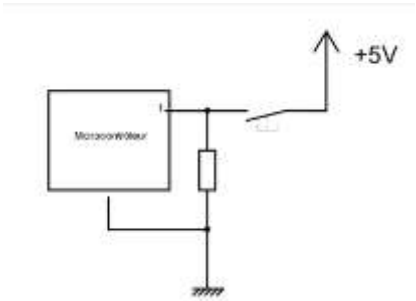


Figure 14 : Résistance de rappel (pull down) => Montage correcte

Remarque : le processeur de la carte Netduino dispose de résistances de rappel internes pouvant être connectées par le logiciel.

7.4.2 Objet logiciel « InputPort »

7.4.2.1 Construction d'un objet InputPort

L'accès à une entrée numérique "physique" nécessite la construction d'un objet logiciel **InputPort**.
Sa lecture se fait avec une méthode **Read**.

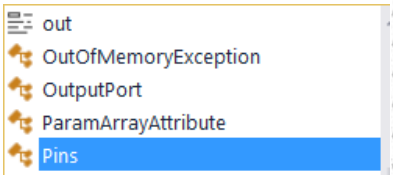
En créant un objet **InputPort**, le code ci-dessous **configure** une connexion de la carte Netduino en entrée numérique.

```
var <nom> = new InputPort(<nom_entrée>, <filtre>,<résistance de rappel>);
```

ou

```
InputPort <nom> = new InputPort(<nom_entrée>, <filtre>,<résistance de rappel>);
```

var et **new** sont des mots-clés du langage C#. **InputPort** est un mot clé de NETMF.



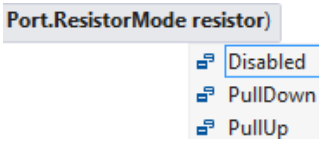
<*nom*> est le nom que le programmeur attribue à l'objet.
<*nom_entrée*> est le nom que **WILDERNESS LABS** attribue à la connexion.

<*filtre*> permet d'activer ou non le filtrage de l'entrée (**true** ou **false**).
<*résistance de rappel*> permet de connecter une résistance à l'intérieur du composant.

Exemple :

```
var BTN = new InputPort(Pins.ONBOARD_SW1, false, Port.ResistorMode.Disabled);
```

Remarques : Le paramètre <*résistance de rappel*> peut prendre les valeurs ci-contre.
Son choix nécessite une analyse du circuit électronique.



Liens vers un article expliquant la notion de résistance de rappel. [https://goo.gl/Yf4lxM]

7.4.2.2 Lecture d'une entrée numérique

La lecture d'une entrée numérique se fait avec la méthode **Read**.

```
var <nomvariable> = <nom>.Read();
```

ou

```
var <nomvariable> = <valeur initiale> ;
<nomvariable> = <nom>.Read();
```

<nomvariable> est le nom que le programmeur attribue à l'objet recevant le résultat de la lecture (**true** ou **false**).
<nom> est le nom que le programmeur attribue à l'objet.
<valeur initiale> : true ou false.

Exemple 1

```
var etatBTN = false ; // Partie déclarative
        …
etatBTN = BTN.Read() ; // Partie exécutive
```

Exemple 2:

```
var etatBTN = BTN.Read() ; // Partie exécutive
```

7.4.3 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
E/S_2a	LightSwitch	E/S numériques : commander une LED avec un bouton-poussoir.				
E/S_2b	NetduinoClav4Digilent	E/S numériques : commander une LED avec un clavier **Digilent 4BP**	☒			☒

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

Pour plus d'informations, consulter la description de la classe **InputPort** sur MSDN

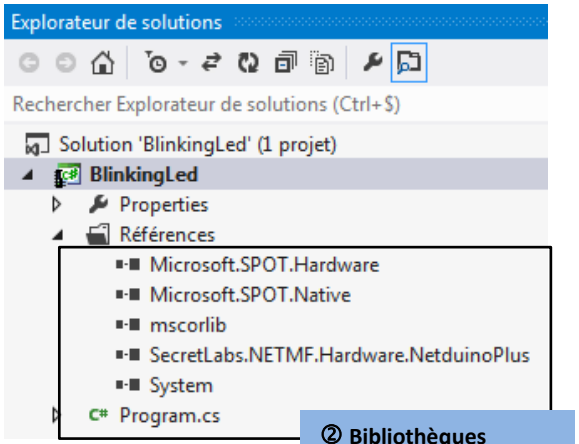
7.4.3.1 E/S_1 : faire clignoter la « LED » de la carte Netduino !

Matériels
- Carte Netduino

Code C#

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace BlinkingLed
{
    public class Program
    {
        public static void Main()
        {③
         ⑤ var ledPort = new OutputPort(Pins.ONBOARD_LED, false);


         while (true)
           { ⑥
             ledPort.Write(true);    // Eclairer la Led  ⑦
             Debug.Print("Led éclairée"); ⑧
             Thread.Sleep(500);      // Attendre 500 ms ⑨
             ledPort.Write(false);   // Eteindre la Led
             Debug.Print("Led éteinte");
             Thread.Sleep(500);      // Attendre 500 ms
          }⑩
        }①①
    }
}
```

① **Espaces de noms**



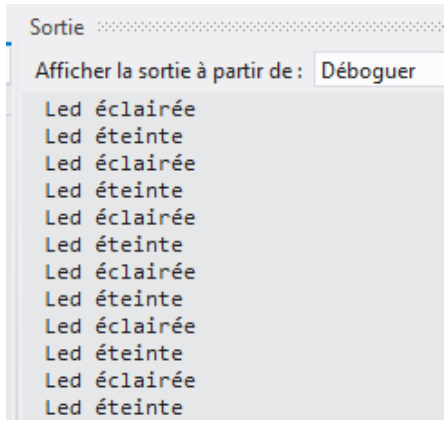
② **Bibliothèques**

④ La construction de l'objet LED semble complexe mais l'**autocomplétion** (**IntelliSense**) simplifie le travail !

Utiliser les classes **SecretLabs** pour la déclaration d'un objet OutputPort.

Pour illustrer l'utilisation du **débogueur !**

Test du programme en mode pas à pas
F10 : Step Over
F11 : Step into



.net Micro Framework

① OutputPort Class
- OutputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool)
- Write(bool)

Explications détaillées de l'exemple E/S_1

① Les **espaces de noms** résolvent le problème du nommage des classes. Deux classes peuvent avoir le même nom si elles se situent dans des espaces de noms différents. Ils permettent donc d'organiser les nombreuses classes du micro framework .Net. Leur déclaration derrière le mot clé ***using*** évite de réécrire le nom complet lors de leur utilisation.

Exemple : La déclaration `using SecretLabs.NETMF.Hardware.NetduinoPlus;`
permet d'écrire `OutputPort(Pins.ONBOARD_LED, false);`
sinon, il serait nécessaire d'écrire `SecretLabs.NETMF.Hardware.NetduinoPlus.OutputPort(Pins.ONBOARD_LED, false);`

② Les **bibliothèques** (*assembly*) sont des fichiers compilés (.dll) contenant des classes. Lorsqu'on utilise les fonctionnalités d'une classe dans un programme, elle doit être présente dans le répertoire "*Référence*" du projet. La documentation de la bibliothèque de classes du micro framework .Net est accessible à partir du site MSDN. [https://goo.gl/dIvfxy]

③ { => En C# une **accolade ouvrante** définit le **début d'un bloc de code**. Ici le début du programme principal (main).

④ L'**autocompletion** (**intellisence**) permet de réduire le taux d'erreur lors de l'écriture du code.

⑤ `var ledPort = new OutputPort(Pins.ONBOARD_LED, false);`
Création d'un objet de type **Port** configuré en sortie (OutputPort). On précise à la broche (Pins) à laquelle est reliée la LED de la carte (ONBOARD_LED) qu'elle doit se comporter comme une sortie tout ou rien (OutputPort). Dans la suite du programme, la référence à cet objet se fera avec le nom ledPort.

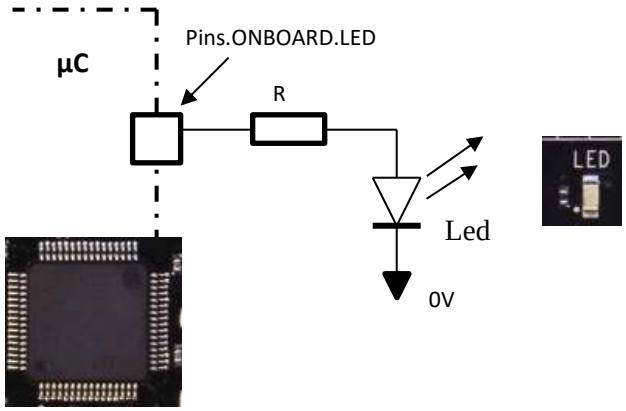


Figure 15 : Association réel - virtuel

⑥ { => Début de la boucle infinie *while(true)* dans laquelle "tourne" le programme.

⑦ La classe OutputPort contient plusieurs méthodes. **Write** est l'une d'entre elles. Elle permet d'écrire un état logique "1" (**true**) ou un état logique "0" (**false**) sur la broche qui lui est associée (ici ledPort).
Lorsque *ledPort.Write(true);* est exécuté, la LED s'éclaire. La structure électronique connectée à la broche étant une **mémoire** : la LED reste éclairée tant qu'on ne l'éteint pas !

⑧ Le code `Debug.Print("LED éclairée");` affiche le texte " LED éclairée" dans la fenêtre de sortie.

⑨ Le code `Thread.Sleep(500);` produit une attente (temporisation) de 0,5s.

⑩ } => En C# une accolade fermante définit la **fin d'un bloc de code**. Ici la fin de la boucle infinie.

A une accolade ouvrante doit correspondre une accolade fermante.

①① } => Fin du programme principal (main).

⑪ Lien vers le site MSDN (description de la classe OutputPort).

7.4.3.2 E/S_2a : commander une LED avec un bouton-poussoir

Matériels
- Carte Netduino

Code C#

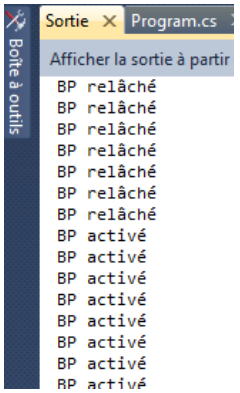
```
using System.Threading; ①
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace LightSwitch
{
  public class Program
  {
    public static void Main()
    {③
     var BTN = new InputPort(Pins.ONBOARD_SW1, false, Port.ResistorMode.Disabled);④
     var led = new OutputPort(Pins.ONBOARD_LED, false);⑤

     while (true)
        { ⑥
           if (BTN.Read()) ⑦
            {
               led.Write(true); ⑧
               Debug.Print("BP activé");⑨
            }
           else
            {
               led.Write(false);
               Debug.Print("BP relâché");
            }
           Thread.Sleep(100); ⑩  // 100 milliseconds
     }
    }
  }
}
```

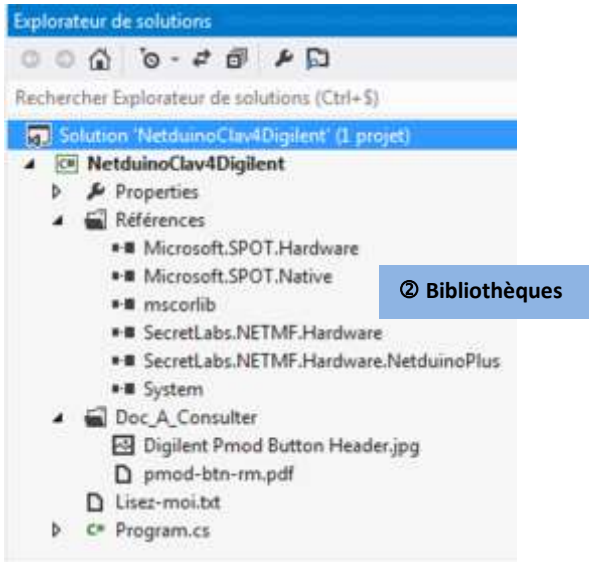
Utiliser les classes **SecretLabs** pour la déclaration des objets InputPort et OutputPort.

Utilisation du **débogueur**

Test du programme en mode pas à pas
F10 : Step Over
F11 : Step into

① InputPort Class
InputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool, Microsoft.SPOT.Hardware.Port.ResistorMode)
Read()
① OutputPort Class
OutputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool)
Write(bool)





Explications détaillées de l'exemple E/S_2a

①②③⑤⑥⑧⑨⑩: Commentaires identiques à ceux de l'exemple **E/S_1**

④ Création d'un objet de type **Port** configuré en **entrée** (InputPort).
On précise à la broche (Pins) à laquelle est relié le bouton-poussoir de la carte (ONBOARD_SW1) qu'elle doit se comporter comme une entrée tout ou rien (InputPort). Dans la suite du programme, la référence à cet objet se fera avec le nom BTN.

⑦ La classe InputPort contient plusieurs méthodes. **Read** est l'une d'entre-elles. Elle permet de lire l'état logique de la broche du µC sur laquelle est connecté le bouton-poussoir (BTN). BTN.Read() renvoie true ("1") ou false ("0") selon l'état du bouton-poussoir (appuyé ou non).
If(){} else {} est un test. Si le bouton-poussoir est appuyé alors la led s'éclaire sinon elle s'éteint.

ⓘ Lien vers le site MSDN (description des classes OutputPort et InputPort).

7.4.3.3 E/S_2b : commander une LED avec un clavier Digilent !

Figure 16 : Digilent Pmod Button Header

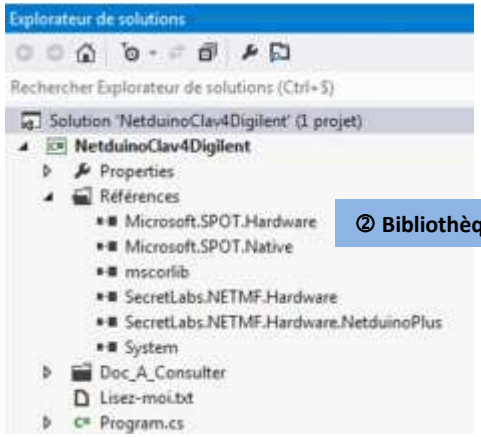
ⓘ InputPort Class
InputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool, Microsoft.SPOT.Hardware.Port.ResistorMode)
Read()
ⓘ OutputPort Class
OutputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool)
Write(bool)

Code C#

```
using System; ①
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace NetduinoClav4Digilent
{
   public class Program
   {
        public static void Main()
    {③
      var BTN0 = new InputPort(Pins.GPIO_PIN_A0, false, Port.ResistorMode.Disabled); ④
      var BTN1 = new InputPort(Pins.GPIO_PIN_A1, false, Port.ResistorMode.Disabled);
      var BTN2 = new InputPort(Pins.GPIO_PIN_A2, false, Port.ResistorMode.Disabled);
      var BTN3 = new InputPort(Pins.GPIO_PIN_A3, false, Port.ResistorMode.Disabled);
      var Led1 = new OutputPort(Pins.GPIO_PIN_D5, false); ⑤

    while (true)
      {⑥

        if (BTN0.Read() || BTN1.Read() || BTN2.Read() || BTN3.Read())⑦
        {
          Led1.Write(true);
        }
        else
        {
          Led1.Write(false);
        }

          Thread.Sleep(100);
      }
    }
  }
}
```

Utiliser les classes **SecretLabs** pour la déclaration des objets InputPort et OutputPort.

Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2
- Clavier Digilent Pmod Button Header
- E-Block Led



② Bibliothèques

Explications détaillées de l'exemple E/S_2b

①②③⑤⑥⑧⑨⑩: Commentaires identiques à ceux de l'exemple **E/S_1**

④ Création d'un objet de type **Port** configuré en **entrée** (InputPort).
On précise à la broche (Pins) à laquelle est relié le bouton-poussoir BTN0 du module Digilent (GPIO_PIN_A0) qu'elle doit se comporter comme une entrée tout ou rien (InputPort). Dans la suite du programme, la référence à cet objet se fera avec le nom BTN0.

⑦ La classe InputPort contient plusieurs méthodes. **Read** est l'une d'entre-elles. Elle permet de lire l'état logique des broches du µC sur lesquelles sont connectés les boutons poussoirs (BTNx). BTN.Read() renvoie true ("1") ou false ("0") selon l'état du bouton poussoir (appuyé ou non).
If(){} else {} est un test. || est le connecteur logique OU. Si un ou plusieurs boutons poussoirs sont appuyés alors la led s'éclaire sinon elle s'éteint.

ⓘ Lien vers le site MSDN (description des classes OutputPort et InputPort).

7.4.3.4 E/S_3 : commander un moteur pas à pas ITC-CNC-2 avec une carte EasyDriverStepperMotor V4.4

7.4.3.4.1 Généralités

« Les moteurs **pas à pas** permettent de convertir directement un signal électrique numérique en un positionnement angulaire de caractère incrémental. Chaque impulsion envoyée par le système de commande au module de puissance se traduit par la rotation d'un pas du moteur. La résolution angulaire d'un moteur pas à pas va de **4 à 400 pas.** » Mémotech sciences de l'ingénieur

Figure 17 : CI Allegro 3967 - MotorDriver - EasyDriver stepper Motor V4.4

7.4.3.4.2 Commande des moteurs pas à pas

Figure 18 : Moteur pas à pas ITC-CNC-2

- **Commande unipolaire**

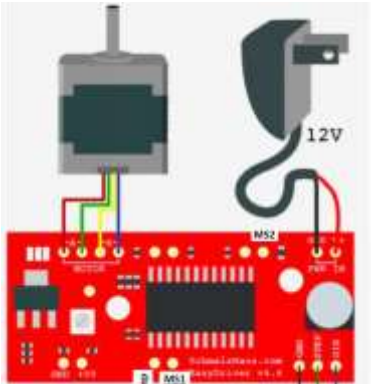
« Chaque phase du moteur est alimentée dans un seul sens. C'est-à-dire que le courant traversant une phase est soit nul, soit positif. L'électronique de commutation est alors plus simple et plus économique, mais les performances du moteur sont moindres. Les moteurs pouvant être pilotés en unipolaire doivent avoir 6 ou 8 fils. » Mémotech sciences de l'ingénieur

- **Commande bipolaire**

« Chaque phase du moteur peut être alimentée par un courant positif, négatif ou nul. Ainsi, tout le cuivre du moteur est utilisé : les performances du moteur sont accrues. Cependant la commande électrique est plus complexe. Les moteurs pouvant être pilotés en mode bipolaire doivent avoir 4 ou 8 fils. » Mémotech sciences de l'ingénieur

7.4.3.4.3 Interface de puissance EasyDriver stepper Motor v4.4

Cette carte de commande permet de contrôler facilement un **moteur pas-à-pas bipolaire** jusqu'à 750 mA par phase. Elle fonctionne par défaut avec **8 micro-pas**.



Elle est basée sur le circuit **Allegro 3967** et permet de contrôler un moteur pas-à-pas (par ex: **ITC-CNC2**) très facilement à partir d'un microcontrôleur (deux broches pour le mode, une broche pour la direction et une pour la vitesse).

Le module permet de fonctionner en mode **pas complet**, **1/2 pas**, **1/4 de pas** ou **1/8 de pas** (mode par défaut).

MS_1	MS_2	Résolution
L	L	Full step
H	L	Half step
L	H	Quarter step
H	H	OneEighth step

DIR	Direction
H	Forward
L	Backward

- **Alimentation**: 7 à 30 Vcc maxi
- **Commande**: via un microcontrôleur 0 - 5 Vcc (mode, direction et vitesse)
- **Sortie**: 150 à 750 mA maxi par phase
- **Dimensions**: 48 x 21 x 18 mm
- **Schéma:** EasyDriver v4.4
- **Fiche technique:** EasyDriver Stepper Motor Driver
- **Distributeur**: GOTRONIC

7.4.3.4.4 miniE shield v1.3

Cette carte, vendu sous la forme d'un kit, peut accueillir l'interface de puissance EasyDriver stepper Motor v4.4 et un module horloge temp réel (RTC). Son utilisation simplifie la connexion du moteur pas à pas au module de puissance. On rajoutera trois straps (MS2 relié à D11, MS1 relié à D10, Enable relié à D3) pour limiter l'énergie consommée et disposer de tous les modes de fonctionnement.

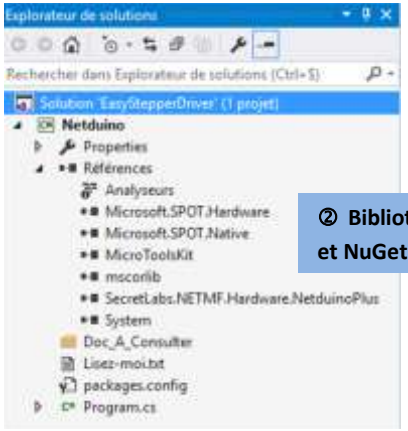
http://shop.airiclenz.com/

Figure 19 : miniE shield v1.3

7.4.3.4.5 Code C#

Matériels
- Carte Netduino
- miniE shield v1.3
- Module EasyDriver Stepper Motor
- Moteur pas à pas ITC-CNC-2

miniE shield v1.3



② **Bibliothèques et NuGet**

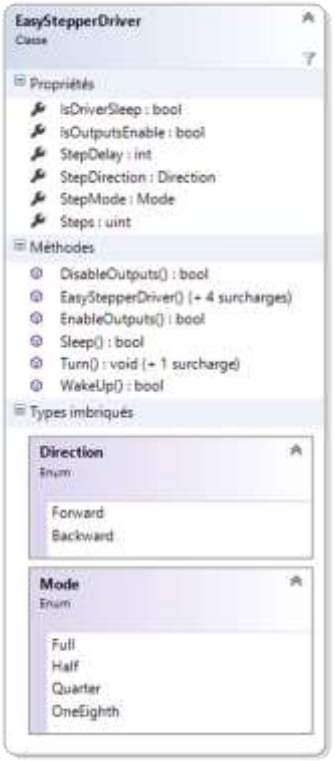
```
using System;  ①
using System.Threading;
using Microsoft.SPOT;
using SecretLabs.NETMF.Hardware.NetduinoPlus;
using Microtoolskit.Hardware.MotorDrivers;

namespace TestNetduinoStepper
  {
   public class Program
    {
      public static void Main()
      { ③ // Programme de test d'un moteur pas à pas ITC-VNC-2
          // avec une carte EasyDriver V4.4 (N = 200 pas – U = 12V – C = 200g/cm) ④
          var time = 2000; UInt16 delay = 2; UInt32 nbpas = 200; ⑤
          var stepper = new EasyStepperDriver(Pins.GPIO_PIN_D13, Pins.GPIO_PIN_D12, Pins.GPIO_PIN_D10, Pins.GPIO_PIN_D11); ⑥

          while (true)
          {// Exemple d'utilisation de la méthode Turn() et des propriétés Steps, StepMode, StepDirection et StepDelay

           Debug.Print("Full Forward"); // 360° for ITC-VNC-1 motor
           Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
           " time= " + stepper.StepDelay + "ms" + "\n");
           stepper.Turn(nbpas, EasyStepperDriver.Direction.Forward, delay, EasyStepperDriver.Mode.Full); Thread.Sleep(time);

           Debug.Print("Half Backward"); // 180° for ITC-VNC-1 motor
           Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
           " time= " + stepper.StepDelay + "ms" + "\n");
           stepper.Turn(nbpas, EasyStepperDriver.Direction.Backward, delay, EasyStepperDriver.Mode.Half); Thread.Sleep(time);


           Debug.Print("Quater Forward"); // 90° for ITC-VNC-1 motor
           Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
           " time= " + stepper.StepDelay + "ms" + "\n");
           stepper.Turn(nbpas, EasyStepperDriver.Direction.Forward, delay, EasyStepperDriver.Mode.Quarter); Thread.Sleep(time);

           Debug.Print("OneEighth Backward"); // 45° for ITC-VNC-1 motor
           Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
           " time= " + stepper.StepDelay + "ms" + "\n");
           stepper.Turn(nbpas, EasyStepperDriver.Direction.Backward, 1, EasyStepperDriver.Mode.OneEighth); Thread.Sleep(time);


           Debug.Print("Full Forward"); // 45° for ITC-VNC-1 motor
           Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
           " time= " + stepper.StepDelay + "ms" + "\n");
           stepper.StepMode = EasyStepperDriver.Mode.Full; stepper.StepDirection = EasyStepperDriver.Direction.Backward;
           stepper.Turn(25); Thread.Sleep(2 * time);
    }
   }
  }
}
```

⑦

Page Web de la classe EasyStepperDriver

MicroToolsKit

①



Connexions à la carte Netduino : **DIR -> Di13** **Step -> Di12** ***MS_2 -> Di11** ***MS_1 -> Di10** ***/Enable ->Di3** ***/Sleep -> Di2**

*** Option**

Explications détaillées de l'exemple E/S_3

① Les **espaces de noms** résolvent le problème du nommage des classes. Deux classes peuvent avoir le même nom si elles se situent dans des espaces de noms différents. Ils permettent donc d'organiser les nombreuses classes du micro framework .Net. Leur déclaration derrière le mot clé ***using*** évite de réécrire le nom complet lors de leur utilisation.

Exemple : La déclaration `using SecretLabs.NETMF.Hardware.NetduinoPlus;`
permet d'écrire `OutputPort(Pins.ONBOARD_LED, false);`
sinon, il serait nécessaire d'écrire `SecretLabs.NETMF.Hardware.NetduinoPlus.OutputPort(Pins.ONBOARD_LED, false);`

② Les **bibliothèques** (*assembly*) sont des fichiers compilés (.dll) contenant des classes. Lorsqu'on utilise les fonctionnalités d'une classe dans un programme, elle doit être présente dans le répertoire "*Référence*" du projet. La documentation de la bibliothèque de classes du micro framework .Net est accessible à partir du site MSDN. [https://goo.gl/dIvfxy]

④ Caractéristiques partielles de la carte Easy stepper Driver Motor et du moteur ITC-CNC-V2.

⑤
```
var time = 2000;      // Attente entre deux mouvements (ms)
UInt16 delay = 2;  // Attente entre deux déplacements angulaires (ms)
UInt32 nbpas = 200; // Nombre de pas du moteur
```

⑥ Construction de l'objet stepper : contrôle la carte EasyDriver V4.4

DIR -> Di13 : commande du sens de rotation
↑**Step** -> Di12 : le moteur tourne d'un pas à chaque front montant
MS_1 -> Di10 : MS1 et MS2 règlent la résolution angulaire (1/8 par défaut)
MS_2 -> Di11 : Exemple : $MS_1MS_2 = 00$ pour ITC-CNC-2 => r = 360°/200 = 1,8°
/Sleep -> Di2 : Actif à l'état bas. Diminue la consommation du circuit hors utilisation ("1" logique par défaut)
/Enable ->Di3 : Actif à l'état bas. Désactive les sortie ("1" logique par défaut)

MS_1	MS_2	Résolution
L	L	Full step
H	L	Half step
L	H	Quarter step
H	H	OneEighth step

DIR	Direction
H	Forward
L	Backward

Figure 20 : EasyDriver - Stepper Motor Driver

```
var stepper = new EasyStepperDriver(Pins.GPIO_PIN_D13, Pins.GPIO_PIN_D12, Pins.GPIO_PIN_D10, Pins.GPIO_PIN_D11) ;
```

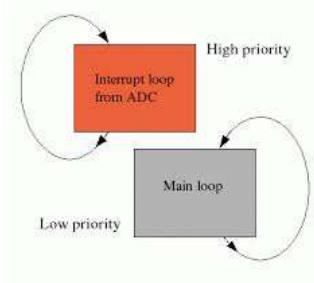
⑦
```
Debug.Print("Full Forward"); // 360° for ITC-VNC-1 motor
Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " + stepper.StepDirection +
" time= " + stepper.StepDelay + "ms" + "\n");
stepper.Turn(nbpas, EasyStepperDriver.Direction.Forward, delay, EasyStepperDriver.Mode.Full);
Thread.Sleep(time); // Attente entre deux mouvements
```

ⓘ Description de la classe EasyStepperDriver maintenue sur Github.

Moteur pas à pas ITC-CNC-2
(N = 200 pas – U = 12V – C = 200g/cm)

7.5 Les interruptions

7.5.1 Généralités



Une **interruption** est un arrêt temporaire de l'exécution normale d'un programme par le processeur afin d'exécuter un autre programme (appelé service d'interruption).

L'interruption est provoquée par une cause externe (action sur un bouton-poussoir, mesure réalisée par un capteur, horloge temps réel, etc.).

On utilise les interruptions afin de permettre des **communications non bloquantes** avec des périphériques externes.

Une interruption tient compte de l'état logique présent sur une broche. Couramment, on la déclenchera sur le **front montant,** le **front descendant, ou chacun des fronts** d'un signal logique.

7.5.2 Précautions d'utilisation

Une interruption sera reconnue si le signal présente des fronts "propres". Il faudra donc s'assurer de la qualité du signal.
Les figures ci-dessous représentent un signal transmis à la fermeture du contact d'un anémomètre. Le signal de gauche n'est pas utilisable, car, à cause du rebondissement du contact, il contient quatre fronts montants au lieu d'un seul comme dans le cas du signal de droite.

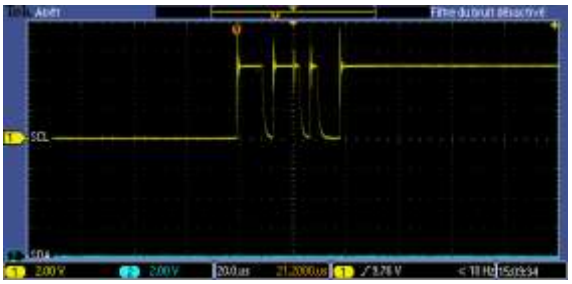


Figure 21 : Rebondissement du contact => signal inutilisable (nécessite un filtrage)

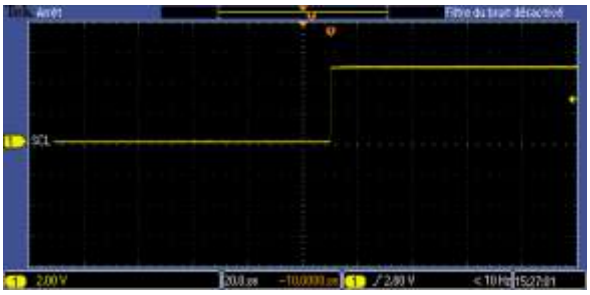


Figure 22 : Front montant « propre » => signal utilisable

7.5.3 Objet logiciel « InterruptPort »

7.5.3.1 Construction d'un objet « InterruptPort »

La détection d'une interruption apparaissant sur une entrée "physique" nécessite la construction d'un objet logiciel **InterruptPort**.

Le code ci-dessous permet d'associer une interruption à une entrée numérique de la carte Netduino.

```
var <nom> = new InterruptPort(<entrée>, <résistance de rappel>, <front de déclenchement>);
```

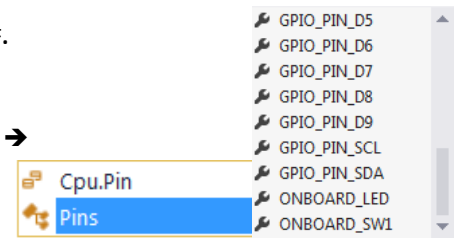
ou

```
InterruptPort <nom> = new InterruptPort(<entrée>, <résistance de rappel>, <front de déclenchement>);
```

var et **new** sont des mots-clés du langage C#. **InterruptPort** et **Pins** sont des mots-clés de NETMF**.**

<nom> est le nom que le programmeur attribue à l'objet auquel on attache une interruption.
<entrée> est le nom que **WILDERNESS LABS** attribue à la connexion.
<résistance de rappel> permet de connecter une résistance à l'intérieur du composant.
<front de déclenchement> précise sur quel front du signal l'interruption sera déclenchée.

➔



Exemple

```
var BTN = new InterruptPort(Pins.GPIO_PIN_D11, true, Port.ResistorMode.Disabled,Port.InterruptMode.InterruptEdgeBoth);
```

7.5.3.2 Gestionnaire d'interruption

Une interruption déclenche l'exécution d'une **routine (sous-programme)** si elle est attachée à un **gestionnaire**. Ceci est réalisé par le code ci-dessous.

```
<nom.OnInterrupt> += new NativeEventHandler (<nom_Routine_Interruption>);
```

new est un mot clé du langage C#. ***OnInterrupt*** **et NativeEventHandler** sont des mots-clés de NETMF**.**
<nom> est le nom que le programmeur attribue à l'objet auquel on a attaché une interruption lors de sa construction.
<nom_Routine_Interruption> est le nom du sous-programme déclenché par l'interruption

Exemple :

```
BTN.OnInterrupt += new NativeEventHandler(IntButton_OnInterrupt);
```

7.5.3.3 Routine d'interruption

Le sous-programme *<nom_Routine_Interruption>* sera exécuté lors de la détection d'une interruption.

Exemple :

```
static void IntButton_OnInterrupt(uint port, uint state, DateTime time)
{
        // Code du sous-programme
}
```

7.5.3.4 La construction du gestionnaire et de la routine d'interruption est simplifiée par Visual Studio

En entrant le code de l'exemple ci-dessous :

BTN.OnInterrupt+= 

On obtient

```
public static void Main()
{
    // write your code here
    InterruptPort BTN = new InterruptPort(Pins.GPIO_PIN_A0, true, Port.ResistorMode.Disabled,
                                                                Port.InterruptMode.InterruptEdgeHigh);

    BTN.OnInterrupt += BTN_OnInterrupt;

}

static void BTN_OnInterrupt(uint data1, uint data2, DateTime time)
{
    throw new NotImplementedException();
}
```

Génération automatique du nom de la routine d'interruption

Génération automatique de la routine d'interruption

Remarque : data1 (ici le Port), data2 (ici son état), time (le temps écoulé entre le déclenchement et la gestion de l'interruption)

7.5.4 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
INT_1	LightSwitchINT	E numérique : commander une LED avec un bouton-poussoir.				
Timer_2	NetduinoAnemometre	Timer : Mesure de la vitesse du vent avec un kit wheather Sensor Assembly p/n80422				☒

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

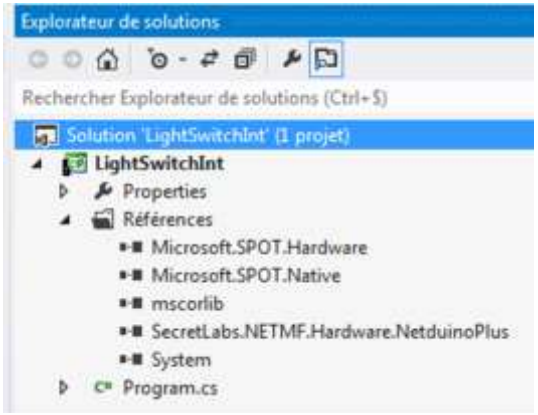
Pour plus d'informations, consulter la description de la classe **InterruptPort** sur MSDN

7.5.4.1 INT_1 : Commander une LED avec un bouton-poussoir !

Matériels
- Carte Netduino plus 2
- Sensor Shield TINKERKIT V2
- Si Clavier Digilent Pmod Button Header (modification du programme nécessaire)

ou

Figure 23 : Digilent Button Header Module



② **Bibliothèques**

Code C#

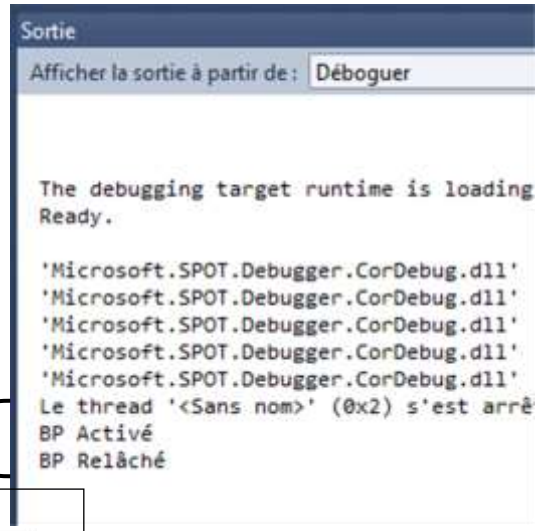
```
using System;                                                    ①
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace LightSwitchInt
{
     public class Program
  {
    static OutputPort LED; ④

     public static void Main()
     {
         var BTN = new InterruptPort(Pins.ONBOARD_SW1,true, Port.ResistorMode.Disabled, Port.InterruptMode.InterruptEdgeBoth); ⑤
         var LED = new OutputPort(Pins.ONBOARD_LED, false); // La broche produit une interruption sur chaque front (montant et descendant) ⑥

         // Déclaration d'un gestionnaire d'interruption
         BTN.OnInterrupt += new NativeEventHandler(IntButton_OnInterrupt); ⑦

         Thread.Sleep(Timeout.Infinite); // Après l'initialisation, Main n'est plus utilisé
     }

     static void IntButton_OnInterrupt(uint port, uint state, DateTime time) // Routine d'interruption ⑧
    {
      LED.Write(state != 0);
           if (state == 0)
           {
               Debug.Print("BP Relâché");
           }
           else
           {
               Debug.Print("BP Activé");
           }
     }
  }
}
```

Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S. ③

Utilisation du **débogueur**
Test du programme en mode pas à pas

F10 : Step Over
F11 : Step into

Une action est affichée seulement lors d'une interruption.



Explications détaillées de l'exemple INT_1

①②③: Commentaires identiques à ceux de l'exemple E/S_1

④ : Pour être visible dans la routine d'interruption, l'objet LED est déclaré **static** à l'extérieur de main.

⑤ : Création d'un objet de type **Port** configuré en **entrée** (InputPort). On précise à la broche (Pins) à laquelle est relié le bouton-poussoir de la carte (ONBOARD_SW1) qu'elle doit se comporter comme une entrée tout ou rien (InputPort). Dans la suite du programme, la référence à cet objet se fera avec le nom BTN. Une interruption est attachée à BTN.

⑥ : Création d'un objet de type **Port** configuré en **sortie** (OutputPort). On précise à la broche (Pins) à laquelle est reliée la Led de la carte (ONBOARD_LED) qu'elle doit se comporter comme une sortie tout ou rien (OutputPort). Dans la suite du programme, la référence à cet objet se fera avec le nom LED.

⑦ Déclaration d'un gestionnaire d'interruption.

⑧ Routine d'interruption.

ⓘ Description de la classe **InterruptPort**.

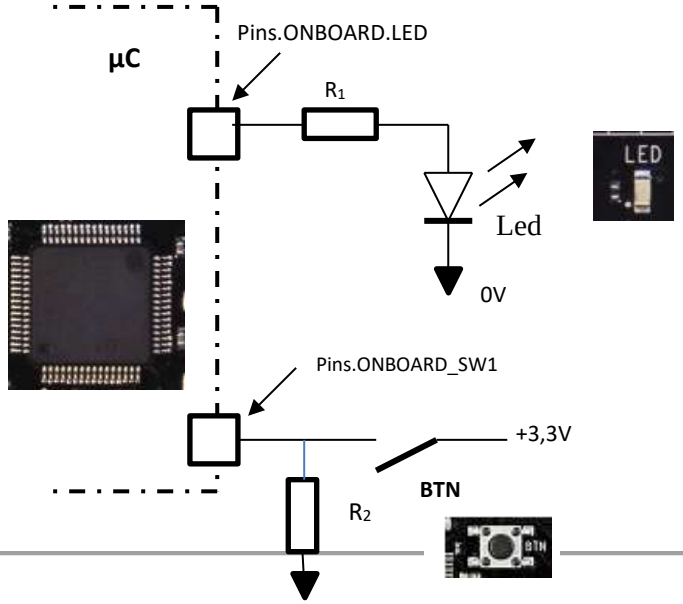


Figure 24 : Association réel - virtuel

Remarque : Contrairement au bouton-poussoir BTN de la carte, le module Digilent dispose de circuits anti-rebond.

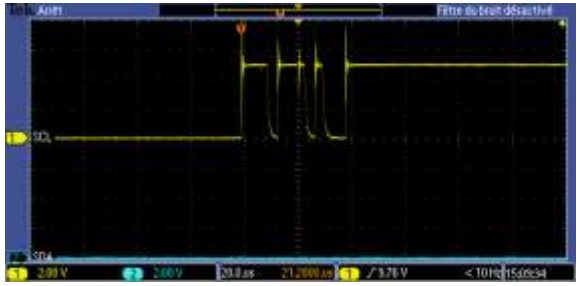


Figure 25 : Rebonds d'un contact

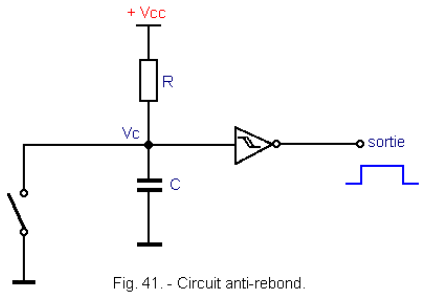


Fig. 41. - Circuit anti-rebond.

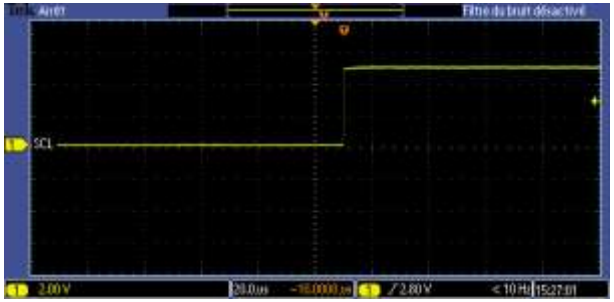
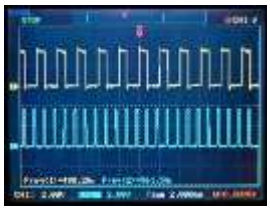


Figure 26 : Elimination des rebonds à la sortie du montage

7.6 Sortie PWM (MLI)

7.6.1 Généralités

Un signal modulé en largeur d'impulsion (MLI ou PWM*) peut être obtenu à partir d'un signal périodique (Clk sur le schéma ci-dessous) de fréquence fixe $F_{clk} = 1/T_{clk}$.
En appliquant ce signal à l'entrée d'un compteur, on obtient un signal numérique M (codés sur n bits) pouvant évoluer entre 0 et $2^n - 1$. La représentation de $M_{(t)}$ est appelée **rampe numérique**.
En appliquant $M_{(t)}$ et un signal constant N_i (codé sur n bits) à un comparateur numérique, on obtient un signal binaire $S_{i(t)}$ de période $T = (2^n - 1).T_{clk}$ dont le temps t_1 (durée à l'état logique « 1 ») est réglé avec la valeur de N_i.

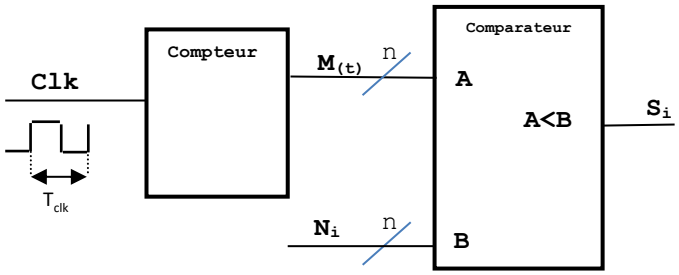


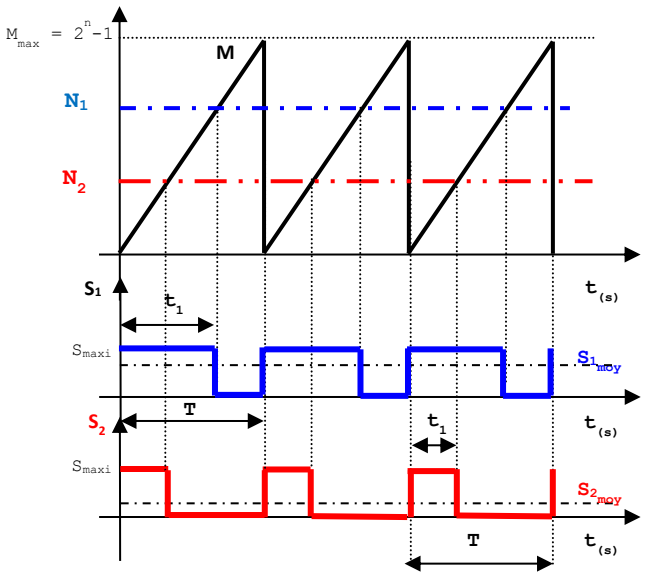
Signaux PWM (MLI)

On appelle $\alpha = t_1/T$ le rapport cyclique du signal $S_{i(t)}$. On montre que la valeur moyenne $<S_i>$ de $S_{(t)}$ est égale au produit de α par S_{maxi}.

On donne ci-dessous le schéma de principe d'une structure générant un signal M.L.I et les chronogrammes de $S_{i(t)}$ pour deux valeurs particulières de N_i (N_1 et N_2).

***PWM** : Pulse with Modulation





Clk : signal logique périodique de fréquence F_{clk}

M : valeur numérique évoluant dans l'intervalle $0 \leq M \leq 2^n-1$, codée sur n bits.

N_i : valeur numérique constante codée sur n bits.

Comparateur : si (A<B) alors S_i <- S_{maxi} sinon S_i <- 0

S_i : signal logique de rapport cyclique réglable

$$S_{moy} = \alpha . S_{max}$$

7.6.2 Les sorties PWM des cartes Netduino

Les connexions **D11 à D9** et **D6, D5 et D3** peuvent être configurées **en sortie numérique PWM**.

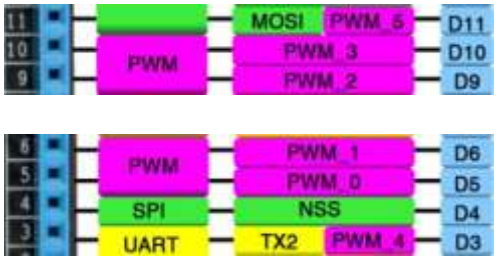


Figure 27 : Netduino plus 2 - Sorties PWM

7.6.3 Précautions d'utilisation

Identiques à celles décrites dans le paragraphe "Sorties numériques".

7.6.4 Objet logiciel « PWM »

7.6.4.1 Description d'un objet PWM

L'accès à une sortie PWM "physique" nécessite la construction d'un objet logiciel **PWM**.

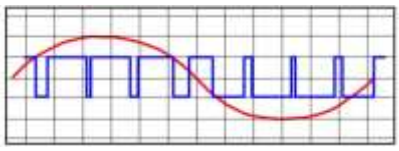
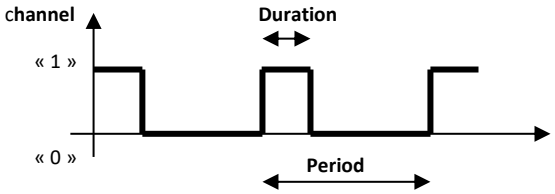


Figure 28 Signal PWM

Les propriétés d'un objet « PWM » du MicroFramework .NET v4.3 sont rassemblées dans le tableau associé à la figure ci-dessous.



Frequency = 1/Period

DutyCycle = Duration/Period

Type	Nom de la propriété	Repère utilisé dans la suite du document	Description
PWMChannels	**channel**	<Sortie>	Nom **WILDERNESS LABS** attribué à la connexion (pin) sur laquelle sortira le signal PWM.
double	**Frequency**	*<Fréquence>*	Fréquence **en Hz** attribuée au signal. (Entre **20Hz et 500kHz** avec une **Netduino plus 2).**  En dessous de 20Hz le firmware peut se « planter » et devra être réinstallé.
double	**DutyCycle**	*<Rapport Cyclique>*	Rapport cyclique attribué au signal. **Entre 0 et 1.**
uint	**Period**	<Période>	Période du signal (en **ms ou µs**). L'unité de temps est réglée par le facteur d'échelle **Scale**.
uint	**Duration**	<Durée_t1>	Durée du signal à l'état haut (« 1 »). L'unité de temps est réglée par le facteur d'échelle **Scale**.
ScaleFactor	**Scale**	<Echelle>	Unité de temps pour *Period* et *Duration* : - Millisecondes (ms) - Microsecondes (µs) - Nanosecondes (ns) : non utilisable avec une carte Netduino plus 2
bool	**invert**	<Inverse>	Inverse (**true)** ou non (**false)** l'état logique du signal.

Deux constructeurs permettent d'initialiser un objet PWM.

7.6.4.2 Constructeur à quatre paramètres

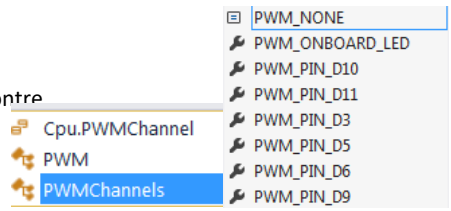
Ce constructeur permet de régler le signal PWM à partir de la fréquence et du rapport cyclique souhaités.

```
var <nom> = new PWM(<Sortie>, <Fréquence>, <Rapport Cyclique>,<Inverse>);
```

ou

```
PWM <nom> = new PWM(<Sortie>, <Fréquence>, <Rapport Cyclique>, <Inverse>);
```

var et **new** sont des mots-clés du langage C#. **PWM** est un mot clé de NETMF**.**
<nom> est le nom que le programmeur attribue à l'objet.
<Sortie> est le nom **WILDERNESS LABS** de la connexion. Les sorties possibles sont identifiées ci-contre



Exemple

```
var duty = 0.4; // Rapport cyclique entre 0 et 1
var Freq = 10000; // Fréquence en Hz
var pwm = new PWM(PWMChannels.PWM_ONBOARD_LED, Freq, duty, false);
```

- **Réglage de la fréquence du signal**

La fréquence du signal est contrôlée en affectant une valeur réelle (type **double**) à la propriété **Frequency** de l'objet logiciel.

```
<nom>.Frequency = <valeur>;
```

Exemple

```
pwm.Frequency = 1000;   // fréquence réglée à 1KHz
```

- **Réglage du rapport cyclique**

Le rapport cyclique du signal est contrôlé en affectant une valeur réelle (type **double**) à la propriété **DutyCycle** de l'objet logiciel.

```
<nom>.DutyCycle = <valeur>;
```

<nom> est le nom que le programmeur attribue à l'objet.
<valeur> est la valeur du rapport cyclique (comprise entre 0 et 1).

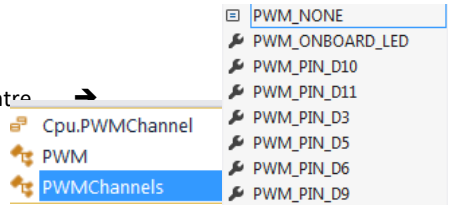
Exemple
```
pwm.DutyCycle = 0.4;   // rapport cyclique réglé à 40%
```

7.6.4.3 Constructeur à cinq paramètres

Ce constructeur permet de régler le signal PWM à partir de la durée à l'état haut et de la période souhaitées.

```
var <nom> = new PWM(<Sortie>, <Période>, < Durée_t1>, <Echelle>, <Inverse>);
```
ou
```
PWM <nom> = new PWM(<Sortie>, <Période>, < Durée_t1>, <Echelle>, <Inverse>);
```

var et **new** sont des mots-clés du langage C#. **PWM** est un mot clé de NETMF.
<nom> est le nom que le programmeur attribue à l'objet.
<Sortie> est le nom **WILDERNESS LABS** de la connexion. Les sorties possibles sont identifiées ci-contre ➔
<Echelle> est l'unité de temps pour la durée à l'état haut et la période du signal.



Exemple
```
var scale = PWM.ScaleFactor.Microseconds;
uint duration = 1000;
uint period = 10000;
var EscMotBas = new PWM(PWMChannels.PWM_PIN_D11, period, duration, scale, false);
```

- **Réglage du facteur d'échelle**

Avec une carte Netduino, le facteur d'échelle (unité de temps) des propriétés *Duration* et *Period* peut prendre les valeurs ms et µs.

```
<nom>.Scale = <valeur>;
```

Exemple
```
EscMotBas.Scale = PWM.ScaleFactor.Microseconds;
```

- **Réglage de la durée à l'état haut du signal**

La durée à l'état haut du signal est contrôlée en affectant une valeur entière à la propriété **Duration** de l'objet logiciel. L'unité de temps est réglée par la propriété **Scale**.

```
<nom>.Duration = <valeur>;
```

Exemple
```
EscMotBas.Scale = PWM.ScaleFactor.Microseconds;
EscMotBas.Duration = 1000 + (uint)(reglage * 10);
// Si reglage varie entre 0 et 100 alors Duration varie entre 1000µs et 2000µs
```

- **Réglage de la période du signal**

La période du signal est contrôlée en affectant une valeur entière à la propriété **Period** de l'objet logiciel.

```
<nom>.Period = <valeur>;
```

Exemple
```
EscMotBas.Scale = PWM.ScaleFactor.Microseconds;
EscMotBas.Duration = 10000;  // Période réglée à 10000µs
```

7.6.4.4 Démarrage et arrêt du signal PWM

```
<nom>.Start() ;
<nom>.Stop() ;
```

Exemple :
```
pwm.Start();   // Démarrage du signal
…
pwm.Stop();   // Arrêt du signal
```

7.6.5 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
PWM_1	NetduinoPWM	PWM : Faire varier la luminosité d'une LED				
PWM_2	NetduinoArdumoto	PWM : Faire varier la vitesse d'un moteur à CC	☒			☒
PWM_3	NetduinoServo	PWM : Régler la position d'un servomoteur de modélisme	☒			
PWM_4	NetduinoEscBrushless	PWM : Régler la fréquence de rotation d'un moteur brushless				

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

Pour plus d'informations, consulter la description de la classe **PWM** sur MSDN

7.6.5.1 PWM_1 : Faire varier la luminosité d'une LED

Fonctionnement : La luminosité de la LED croit de 10% à 90% par pas de 1% puis repasse à 10%

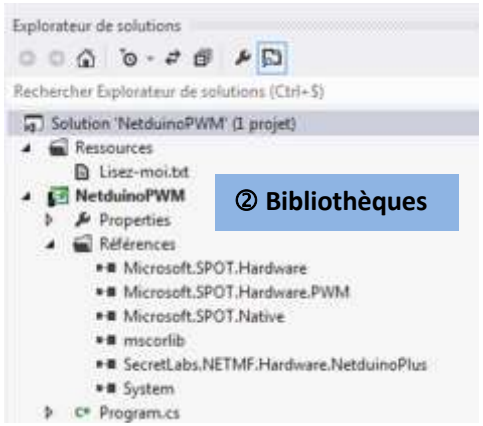
Code C#

```
using System.Threading; ①
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TestNetduinoPWM
{
    public class Program
    {
        public static void Main()
        {
            var duty = 0.4; // ⚠ Rapport cyclique : doit être compris entre 0 et 1
            var Freq = 10000; // Fréquence en Hz
            var pwm = new PWM(PWMChannels.PWM_ONBOARD_LED, Freq, duty, false); ④

            pwm.Start();⑤

            while (true)
            {
                for (var i = 10.0; i < 90.0; i++) ⑥
                 {
                    pwm.DutyCycle = i/100; ⑦
                    Debug.Print("i: "+ i + " pwm: "+ pwm.DutyCycle.ToString("N2")); ⑧
                    Thread.Sleep(10);
                 }
            }
        }
    }
}
```

Matériels
- Carte Netduino

PWM Class
- Dispose()
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, uint, uint, Microsoft.SPOT.Hardware.PWM.ScaleFactor, bool)
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, double, double, bool)
- Start(Microsoft.SPOT.Hardware.PWM[])
- Start()
- Stop(Microsoft.SPOT.Hardware.PWM[])
- Stop()
- Duration
- DutyCycle
- Frequency
- Period
- Pin
- Scale

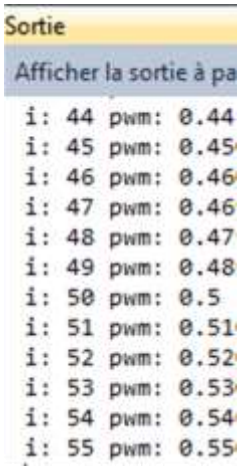
⚠ Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S. ③



② Bibliothèques

Rapport cyclique visualisé dans le **débogueur**.

Test du programme en mode pas à pas
F10 : Step Over
F11 : Step into



Explications détaillées de l'exemple PWM_1

①②③ : Commentaires identiques à ceux de l'exemple E/S_1.

④ Création d'un objet de type **PWM**. On précise qu'à la mise sous tension, la sortie PWM : PWMChannels.PWM_ONBOARD_LED produira un signal de fréquence 10kHz et de rapport cyclique 40%. Dans la suite du programme, la référence à cet objet se fera avec le nom pwm.

⑤ Démarrage du signal PWM. **Attention à ne pas l'oublier** !

⑥ Boucle « Pour… » : fait évoluer la valeur de la variable i de 10% à 90% par pas de 1%.

⑦ Réglage du rapport cyclique avec le résultat du calcul i/100.

⑧ Affichage de i et du rapport cyclique dans le débogueur.

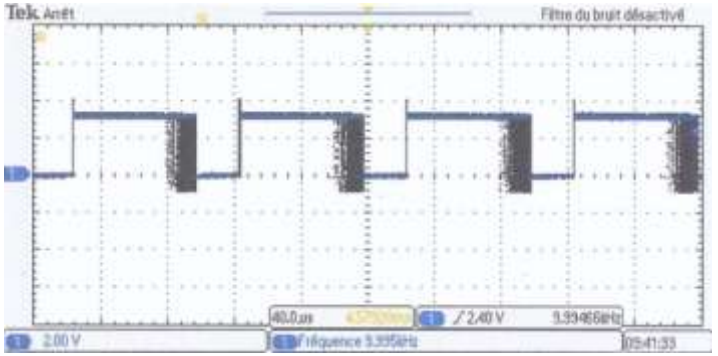


Figure 29 : Chronogramme - Signal PWM

7.6.5.2 PWM_2 : Faire varier la vitesse d'un moteur à courant continu

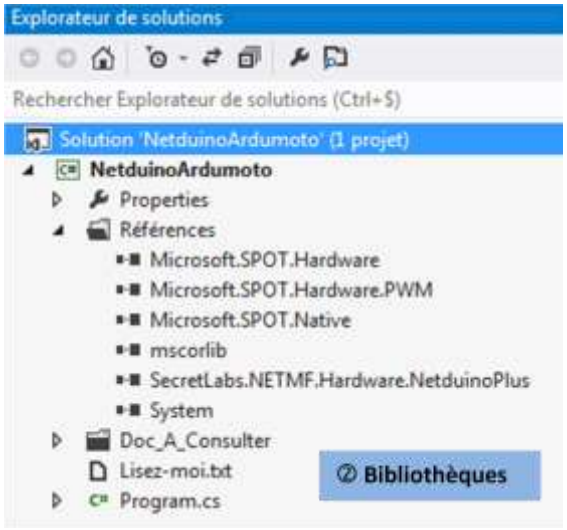
Matériels
- Carte Netduino
- Shield **Ardumoto** + connectique
- 1 seul moteur Lynxmotion GHM-16 (contrairement à la photo !)

Figure 30 : Shield Ardumoto

Code C#

```
using System; ①
using System.Threading;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TestNetduinoArdumoto
{
    public class Program
    {
        public static void Main()
        {
            var duty = 0.1; // Rapport cyclique entre 0 et 1
            var freq = 1000; // Fréquence du signal en Hz

            // Moteur 12V connecté sur la voie B de la carte Ardumoto
            var PWMB = new PWM(PWMChannels.PWM_PIN_D11, freq, duty, false); ④
      var DIRB = new OutputPort(Pins.GPIO_PIN_D13, true); ⑤

      PWMB.Start();⑥

      while (true)
            {// Rampe d'accélération

                for (int i = 0; i < 9; i++) ⑦
                {
                    PWMB.DutyCycle = duty + 0.1 * i; ⑧
                    Thread.Sleep(500); ⑨
                }
                Thread.Sleep(4000); // Palier à vitesse constante ⑩
                PWMB.DutyCycle = 0.1; ①①
            }
        }
    }
}
```

⚠ Utiliser **impérativement** les classes Secret Labs pour la déclaration des E/S. ③



PWM Class
- Dispose()
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, uint, uint, Microsoft.SPOT.Hardware.PWM.ScaleFactor, bool)
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, double, double, bool)
- Start(Microsoft.SPOT.Hardware.PWM[])
- Start()
- Stop(Microsoft.SPOT.Hardware.PWM[])
- Stop()
- Duration
- DutyCycle
- Frequency
- Period
- Pin
- Scale

.net Micro Framework

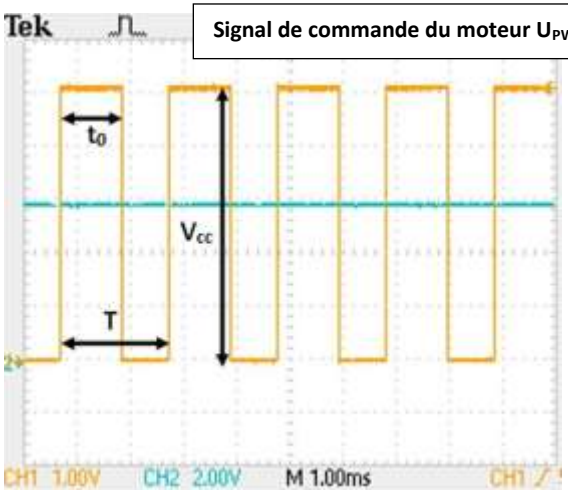
Connectique

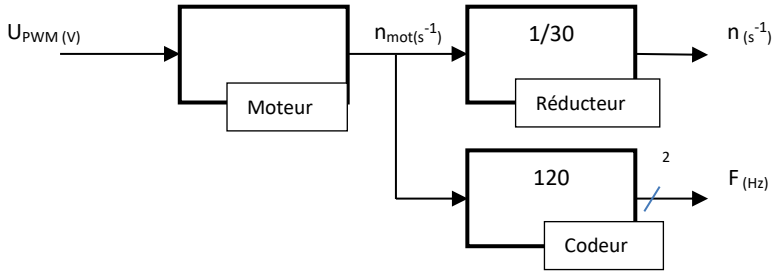
Netduino	Ardumoto	Moteur
Di3	PWMA	A
Di12	DIRA	
Di11	PWMB	B
Di13	DIRB	

Motoréducteur GHM16 équipé d'un codeur E4P-120-079-HT

Motoréducteur Lynxmotion GHM-16	
• Voltage : 6-12vdc • RPM : 200 • Couple : 0.78Kg-cm (10.83oz-in) • Réduction : 30:1 • Poids : 0.34 livres (154g) • Diamètre extérieur : 37mm • Diamètre (essieu) : 6mm	
Encodeur en quadrature lynxmotion **E4P-120-079-HT**	
• Alimentation : 5V Cycles par révolution: 120 • Comptes en quadrature par révolution: 480 • Fréquence: 30kHz Red = +5vdc Black = Ground Green = Output A Yellow = Output B	 

Signal de commande du moteur U_{PWM}





Explications détaillées de l'exemple PWM_2

①②③: Commentaires identiques à ceux de l'exemple E/S_1

④ Création d'un objet PWM. Contrôle la fréquence de rotation du moteur.

⑤ Création d'un objet OutputPort. Contrôle le sens de rotation du moteur.

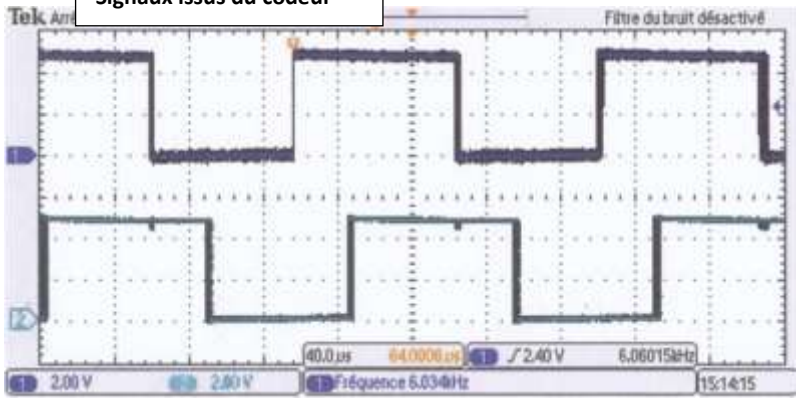
⑥ Démarrage du signal PWM. **Attention à ne pas l'oublier !**

⑦ Accélération réalisée avec une boucle Pour… ⑧Augmente la fréquence de rotation du moteur par pallier de 10% entre 10% et 90% ⑨ toutes les 0,5s.

⑩ Vitesse constante pendant 4s.

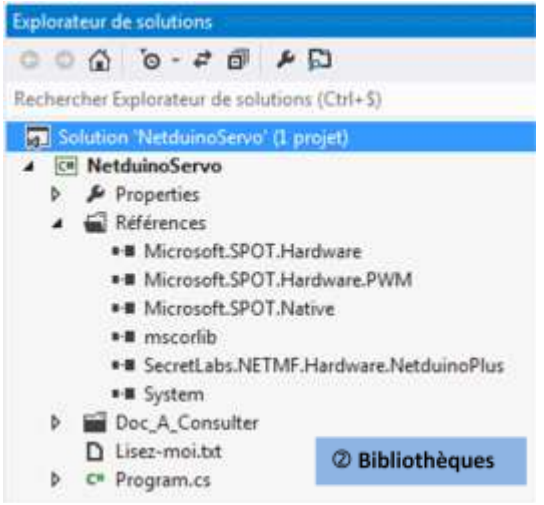
①① Retour à une fréquence de rotation correspondant à un rapport cyclique de 10%.

Signaux issus du codeur



7.6.5.3 PWM_3 : Commande d'un servomoteur de modélisme

Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2
- Servomoteur de modélisme type FUTABA S3003

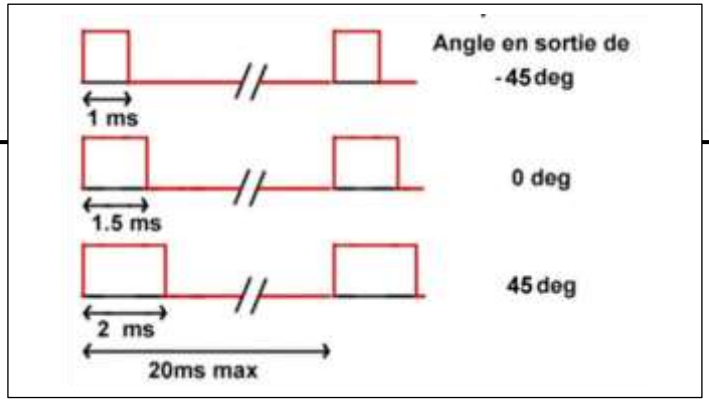


Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace NetduinoServo
{
    public class Program
    {
        public static void Main()
        {
           var duty = 0.05; // Rapport cyclique : doit être compris entre 0 et 1
            var freq = 50; // Fréquence en Hz min pour ce type de servomoteur

            // Configuration de l'E/S numérique 3 de la carte
            // Netduino en sortie PWM
            // Connexion du servomoteur au connecteur O5 de la carte
            // TINKERKIT
            PWM Servo = new PWM(PWMChannels.PWM_PIN_D3,freq,duty,false);

            Servo.Start();

             while (true)
             { // Déplacement angulaire de 90°
                 for (var i = 0; i < 6; i++)
                 {
                     Servo.DutyCycle = duty + i*0.01;
                     Thread.Sleep(1000);
                 }

                 Servo.DutyCycle = duty;

             }
        }
    }
}
```

Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S numériques et PWM.

PWM Class
- Dispose()
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, uint, uint, Microsoft.SPOT.Hardware.PWM.ScaleFactor, bool)
- PWM(Microsoft.SPOT.Hardware.Cpu.PWMChannel, double, double, bool)
- Start(Microsoft.SPOT.Hardware.PWM[])
- Start()
- Stop(Microsoft.SPOT.Hardware.PWM[])
- Stop()
- Duration
- DutyCycle
- Frequency
- Period
- Pin
- Scale



Explications détaillées de l'exemple PWM_3
Voir PWM_2

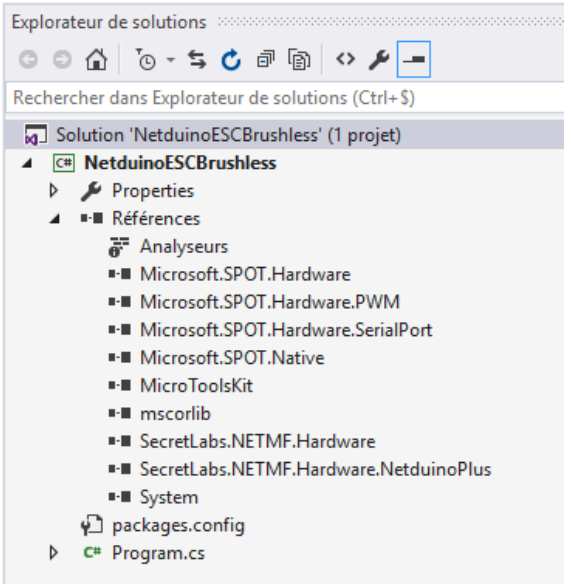
7.6.5.4 PWM_4 : Régler la fréquence de rotation d'un moteur brushless

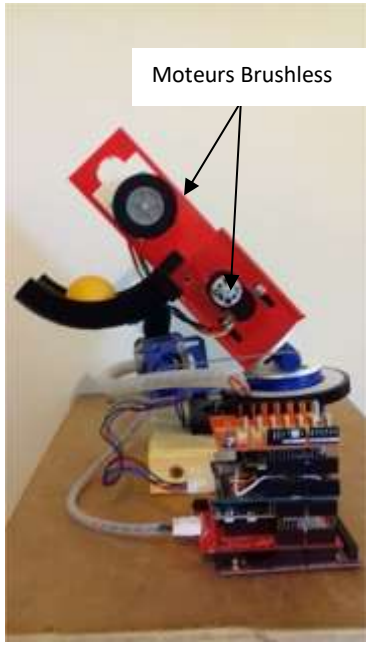
Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2
- Moteur Brushless ROXXY C22-20-20 1330 KV
- Régulateur GROUPNER S3055

Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S numériques et PWM.

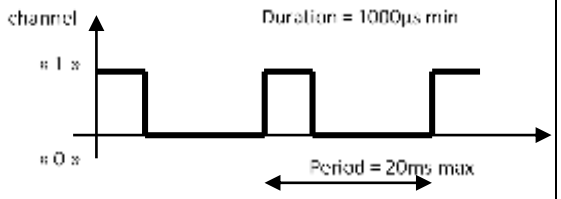
Code C#

```
using System.Threading;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

using Microtoolskit.Hardware.Displays;

namespace NetduinoESCBrushless
{
    public class Program
    {
        public static void Main()
        {
            // Afficheur à connecter sur O5 de Tinkerkit
            var lcd = new ELCD162();

            // Potentiomètre de réglage de la fréquence de rotation du moteur bas
            var potMotBas = new AnalogInput(Cpu.AnalogChannel.ANALOG_0);
            potMotBas.Scale = 100;

            // Paramètres de réglage pour le régulateur GROUPNER S3055
            var scale = PWM.ScaleFactor.Microseconds;
            uint duration = 1000;
            uint period = 10000;

            // Signal de commande pour le moteur Bas (O0 Tinkerkit)
            var EscMotBas = new PWM(PWMChannels.PWM_PIN_D11, period, duration, scale, false);

            // Initialisations
            EscMotBas.Start(); lcd.Init(); lcd.ClearScreen();

            while (true)
            {
                lcd.ClearScreen();
                double reglage = potMotBas.Read();
                EscMotBas.Duration = 1000 + (uint)(reglage * 10);
                lcd.PutString("t1MotB = " + EscMotBas.Duration.ToString("F0") + "us");
                Thread.Sleep(200);
            }
        }
    }
}
```





Lanceur de balle de ping-pong



Explications détaillées de l'exemple PWM_4
Voir PWM_2

8 Les entrées analogiques

Pour plus d'informations, consulter la description de la classe **AnalogInput** sur MSDN

8.1 Généralités

*« Le terme **analogique** désigne les phénomènes, appareils électroniques, composants électroniques et instruments de mesure qui représentent une information par la variation d'une grandeur **physique** (ex. : une **tension électrique**). Ce terme provient du fait que la mesure d'une valeur naturelle (ou d'un élément de **signal électrique** ou électronique) varie de manière **analogue** à la source.*

Un signal électrique est une grandeur électrique dont la variation dans le temps transporte une information, d'une source à une destination
*La grandeur électrique que l'on considère pour la transmission et le traitement du signal peut être directement la **différence de potentiel** ou l'intensité d'un courant électrique.*
*On distingue généralement les signaux électriques par la nature de l'information qu'ils transmettent. Dans un **signal analogique**, la valeur de la grandeur électrique peut varier arbitrairement dans les limites définies par le canal de transmission. Dans le cas général, cette grandeur varie dans le temps, **s = F(t)**. » Wikipédia*

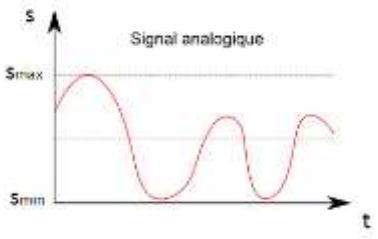
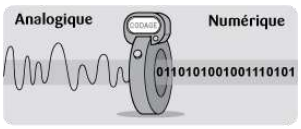


Figure 31 : Signal analogique

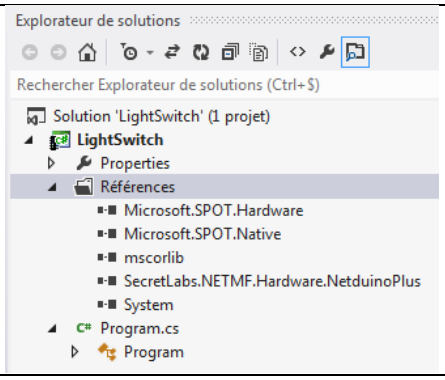


Pour être traité par un microcontrôleur, un signal analogique doit être délivré à une entrée dite "analogique". Cette entrée est connectée à un convertisseur analogique numérique qui se charge de numériser le signal analogique.

8.2 Les entrées analogiques des cartes Netduino

Un programme implanté dans une carte Netduino accède à ses entrées analogiques par l'intermédiaire de la bibliothèque (assembly) ci-dessous :

- NETMF : Microsoft.SPOT.Hardware

Espaces de noms	Références
`using Microsoft.SPOT.Hardware;`	

Les connexions **A5 à A0** peuvent être configurées en **entrée analogique**.

Ces entrées sont reliées au convertisseur analogique numérique (**CAN**) du microcontrôleur par l'intermédiaire d'un multiplexeur analogique.

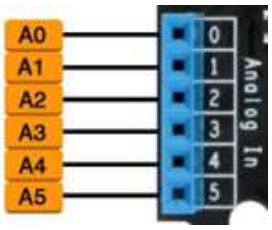
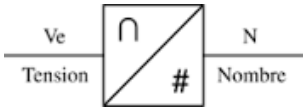


Figure 32 : Netduino plus 2 – Entrées analogiques

<u>Caractéristiques du CAN accessible sur la carte Netduino plus 2</u>

Tension pleine échelle **VPE = 3,3V** résolution **n = 12bits**



$$N = Ve\frac{2^n}{VPE}$$

8.3 Précautions d'utilisation

Les entrées analogiques sont **fragiles**. Elles ne supportent ni les décharges **électrostatiques** ni les **surtensions**. Il ne faut ni les toucher ni leur appliquer une tension supérieure à **5V** ou inférieure à **0V**.

8.4 Objet logiciel « AnalogInput»

8.4.1 Construction d'un objet AnalogInput

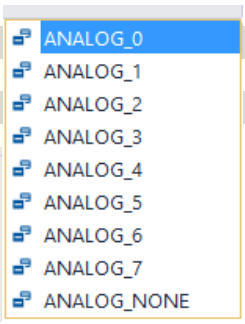
L'accès à une entrée analogique "physique" nécessite la construction d'un objet logiciel **AnalogInput**. La lecture d'une entrée analogique se fait avec une méthode **ReadRaw()** ou une méthode **Read().**

En créant un objet **AnalogInput**, , le code ci-dessous **configure** une connexion de la carte Netduino en entrée analogique.

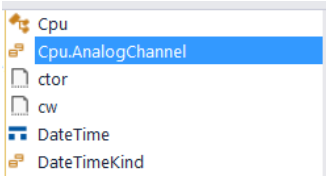
```
var <nom> = new AnalogInput(<entrée>);
```

ou

```
AnalogInput <nom> = new AnalogInput(<entrée>);
```



var et **new** sont des mots-clés du langage C#. ***AnalogInput*** est un mot clé de NETMF**.**
<nom> est le nom que le programmeur attribue à l'objet.
<entrée> est le nom que **NETMF** attribue à la connexion. ➔



Exemple :

```
var Potentiometre = new AnalogInput(Cpu.AnalogChannel.ANALOG_0);
 // Un potentiomètre est connecté à l'entrée A0 de la carte Netduino
```

8.4.2 Lecture d'une entrée analogique

La valeur numérique issue de la conversion analogique numérique dépend de la méthode utilisée **ReadRaw()** ou **Read()**.

- **Méthode ReadRaw()**

ReadRaw renvoie la valeur brute issue du convertisseur analogique numérique.

```
var <nomvariable> = <nom>.ReadRaw();
```

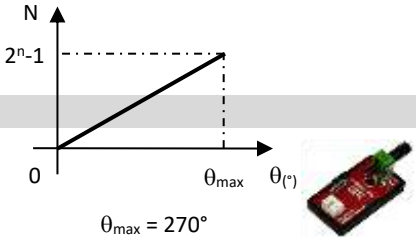
ou

```
var <nomvariable> = <valeur initiale> ;
    <nomvariable> = <nom>.ReadRaw
```

<nomvariable> est le nom que le programmeur attribue à la variable dans laquelle sera mémorisé le résultat de la lecture (compris **entre 0 et 2^n-1**). (n : résolution du CAN)
<nom> est le nom de l'objet sur lequel s'applique la méthode ReadRaw.

Exemple : Mesure de la position angulaire d'un potentiomètre

```
var N = 0;
N = Potentiometre.ReadRaw();
```



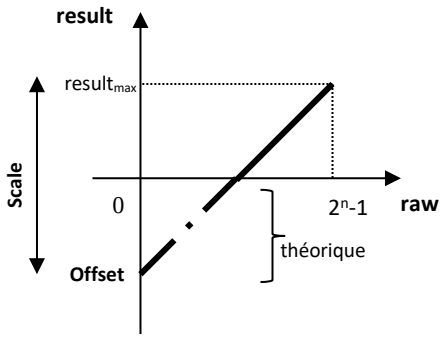
θ_{max} = 270°

```
// La valeur de la position angulaire du potentiomètre
// (Angle) est connue à partir de la relation ci-dessous

Angle = (θmax * N) / 2^n = (270 * N) / 4096;
```

- **Méthode Read()**

Read effectue une mise **à l'échelle** de la valeur brute issue du convertisseur analogique numérique.

result = raw * alpha + K. (documentation Microsoft)



$$result = \frac{(result_{max} - Offset)}{2^n} * raw + Offset$$

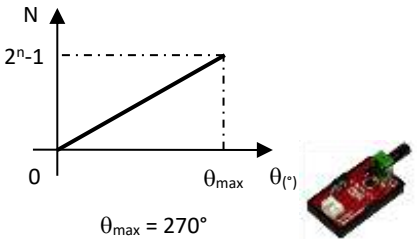
raw : valeur issue du convertisseur analogique numérique (entier compris entre 0 et $2^n - 1$)
result: résultat de la mesure (image d'une grandeur physique)
result$_{max}$: Valeur de la grandeur physique mesurée pour raw = 2^n-1
alpha = (result$_{max}$ – Offset) / 2^n = Scale / 2^n
K = Offset

Le coefficient alpha se règle avec l'attribut **Scale**. **Scale** = result$_{max}$ – Offset (alpha = 1 par défaut)

Le coefficient K se règle avec l'attribut **Offset**. **K = Offset** (K = 0 par défaut)

Exemple: Mesure de la position angulaire d'un potentiomètre
On pose result = θ
Offset = 0 et Scale = θ_{max} = 270

```
Potentiometre.Scale = 270 ;
Angle = Potentiometre.Read() ;
```



θ_{max} = 270°

8.5 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
Analog_1	NetduinoPot	Régler la fréquence de clignotement d'une LED avec un potentiomètre.	☒			
Analog_2	MesureAngle	Mesurer une position angulaire avec un potentiomètre		☒	☒	☒
Analog_3	NetduinoThermoGHI	Mesurer la température ambiante avec un module GHI FEZ thermomètre.				
Analog_4	NetduinoGirouette	Mesurer la direction du vent avec un kit wheather Sensor Assembly p/n80422	☒	☒	☒	☒

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

8.5.1 Analog_1 : Régler la fréquence de clignotement d'une LED avec un potentiomètre !

Code C#

```
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace NetduinoPot
{
    public class Program
    {
        public static void Main()
        {
            var led = new OutputPort(Pins.ONBOARD_LED, false);

            // Le potentiomètre est connecté à l'entrée I0 du shield Tinkerkit (CAN 12 bits) (Limiter la tension à 3,3V)
            var Pot = new AnalogInput(Cpu.AnalogChannel.ANALOG_0 );

            while (true)
            {
                var N = Pot.ReadRaw()/4; // Résultat sur 10 bits
                Debug.Print(N.ToString());
                led.Write(true);
                Thread.Sleep(N);
                led.Write(false);
                Thread.Sleep(N);
            }

        }
    }
}
```

Led

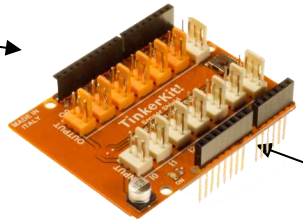
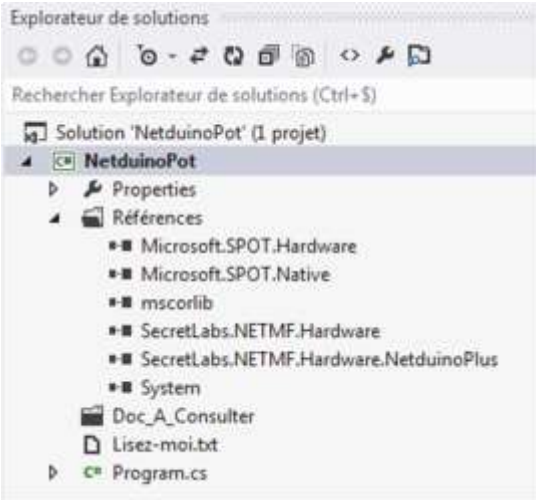


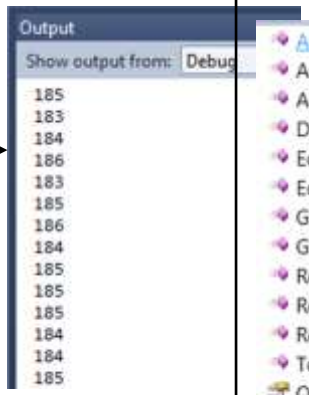
Figure 33 : Module GHI FEZ Potentiomètre



⚠ Utiliser **impérativement** la classe NETMF pour la déclaration de l'entrée analogique et la classe Secret Labs pour la déclaration de la sortie numérique.

Utilisation du **debugger** pour visualiser la valeur renvoyée par le CAN en fonction de la position du potentiomètre.

Pas à pas
F10 : Step Over
F11 : Step into



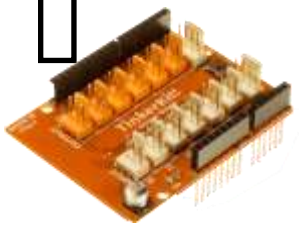
AnalogInput Class

- AnalogInput(Microsoft.SPOT.Hardware.Cpu.AnalogChannel)
- AnalogInput(Microsoft.SPOT.Hardware.Cpu.AnalogChannel, int)
- AnalogInput(Microsoft.SPOT.Hardware.Cpu.AnalogChannel, double, double, int)
- Dispose()
- Equals(object, object)
- Equals(object)
- GetHashCode()
- GetType()
- Read()
- ReadRaw()
- ReferenceEquals(object, object)
- ToString()
- Offset
- Pin
- Precision
- Scale

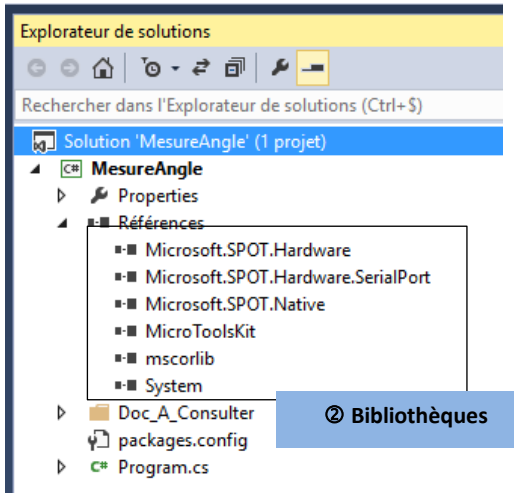
Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2
- Module Led GHI
- Module FEZ Potentiomètre

8.5.2 Analog_2 : M RS 232 sition angulaire

Module GHI FEZ
Potentiomètre



Shield TINKERKIT V2
sur Netduino plus 2



② **Bibliothèques**

Code C#

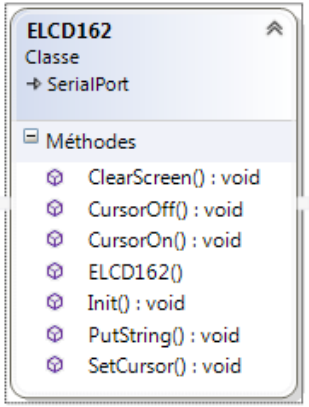
```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using Microtoolskit.Hardware.Displays;

namespace MesureAngle
{
    public class Program
    {③
        public static void Main()
        {
            // Constantes
            const UInt16 delay = 500;   // 500ms

            // Variables
            var N = 0;
            var Angle = 0.0; // Le mot clé var laisse le compilateur définir le type de la variable (entier pour N et réel pour Angle)

            // Configuration des E/S
            // http://msdn.microsoft.com/en-us/library/hh421132.aspx (Description de la classe AnalogInput)
            AnalogInput Potentiometre = new AnalogInput(Cpu.AnalogChannel.ANALOG_5); // Connecté sur I5 de la carte Tinkerkit ⑤
            // http://webge.github.io/ELCD162/ (Description de la classe ELCD162)
            ELCD162 Lcd = new ELCD162(); // Connecté sur O5 Tinkerkit ⑦

            Lcd.Init(); Lcd.ClearScreen(); Lcd.CursorOff();⑧

            while (true)
            { ⑥
                N = Potentiometre.ReadRaw(); ⑨
                Angle = (270 * N) / 4096; ⑩
                Lcd.ClearScreen();
                Lcd.PutString("Angle= " + Angle.ToString() + "deg");
                Debug.Print("N= " + N.ToString() + "    " + Angle.ToString() + "°");
                Thread.Sleep(delay);
            }
        }
    }
}
```

① **Espaces de noms**

④ La construction de l'objet Potentiomètre semble complexe mais l'**autocomplétion** (**IntelliSense**) simplifie le travail !

nuget
MicroToolsKit

Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2
- Afficheur 2x16 + module ELCD162 COMFILE Série
- Module FEZ Potentiomètre



github SOCIAL CODING
⑪ Page Web de la classe ELCD162

Autre solution avec la méthode Read

```
Potentiometre.Scale = 270; // Mise à l'échelle

while (true)
{
    Angle = Potentiometre.Read(); // Lecture et calcul de l'angle
   …
```

Explications détaillées de l'exemple An_2

Schéma fonctionnel

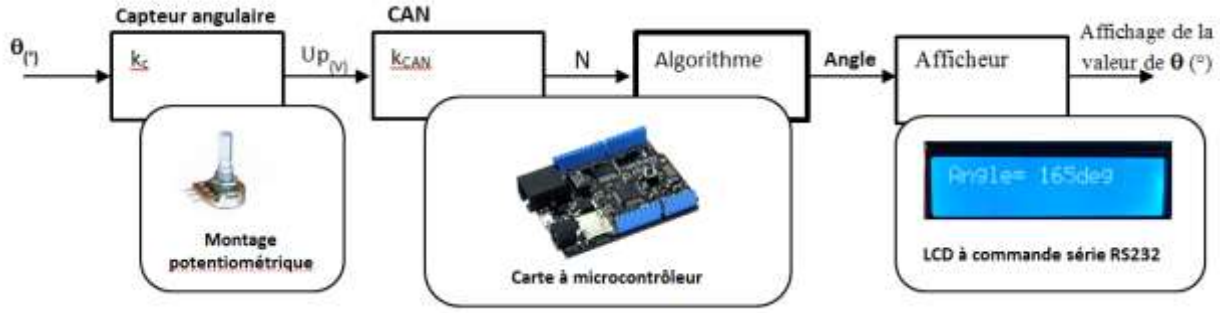


Figure 34 : Schéma fonctionnel - Mesure Angle

kc = VPE/θ_{max} et k_{CAN} = 2^n/VPE

Algorithme

Algorithme MesureAngle
variables
N ← 0 : entier sur 16 bits
Angle ← 0 : réel

début
Lire(N)
Angle ← $\frac{\theta_{max}}{2^n}$**N**
Ecrire(Angle)
fin

Commentaires

① ② ③ ④ ⑥Commentaires identiques à ceux de l'exemple E/S_1.
⑤ Création d'un objet Potentiomètre.
⑦ Création d'un objet LCD à partir de la classe ELCD162 maintenue à jour sur GitHub.
⑧ Initialisation de l'afficheur.
⑨ Lecture de la valeur issue du convertisseur analogique numérique.
⑩ Mise à l'échelle (calcul de l'angle).

ⓘ Lien vers le site GitHub (description de la classe ELCD162).

8.5.3 Analog_3 : Mesurer la température ambiante avec un module GHI FEZ thermomètre

Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using Microtoolskit.Hardware.Displays;

namespace NetduinoThermoGHI
{
    public class Program
    {
        public static void Main()
        {
            // Constantes
            const UInt16 delay = 1000;    // En ms

            // Variables
            var N = 0;
            var Temperature = 0.0;

            // Configuration des E/S
            // http://msdn.microsoft.com/en-us/library/hh421132.aspx (Description de la classe AnalogInput)
            AnalogInput Thermometre = new AnalogInput(Cpu.AnalogChannel.ANALOG_5); // Connecté sur I5 Tinkerkit
            // http://webge.github.io/ELCD162/ (Description de la classe ELCD162)
            ELCD162 Lcd = new ELCD162(); // Connecté sur O5 Tinkerkit

            Lcd.Init(); Lcd.ClearScreen(); Lcd.CursorOff();
            Thermometre.Scale = 82.5; Thermometre.Offset = -25;

            while (true)
            {
                Temperature = Thermometre.Read();
                Debug.Print("N=" + N + "    " + "Temp=" + Temperature.ToString("F1"));
                Lcd.ClearScreen();
                Lcd.PutString("T=" + Temperature.ToString("F1"));
                Thread.Sleep(delay);
            }
        }
    }
}
```

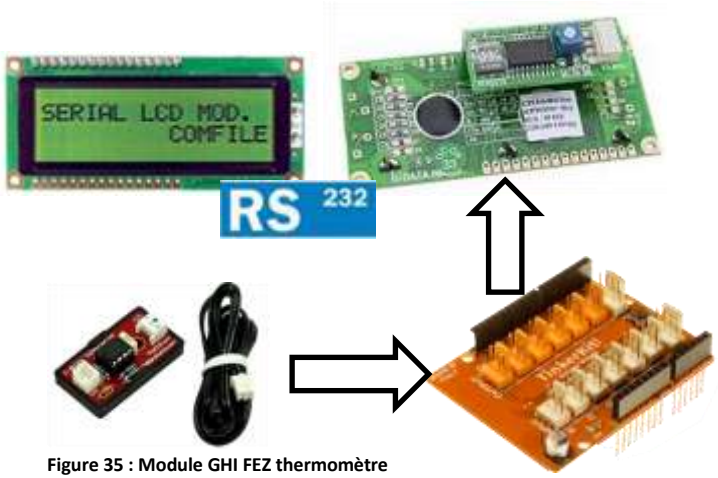
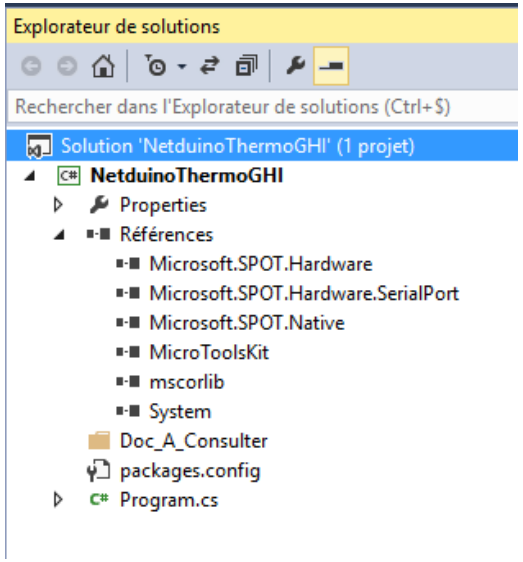


Figure 35 : Module GHI FEZ thermomètre

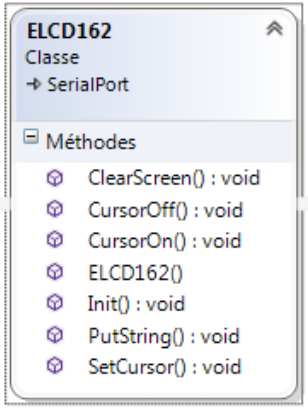
[Shield TINKERKIT V2](#)
sur Netduino plus 2



Matériels

- [Module GHI FEZ thermomètre](#)
- Carte Netduino
- Sensor [Shield TINKERKIT V2](#)
- Afficheur 2x16 + [module ELCD162 COMFILE Série](#)

[MicroToolsKit](#)



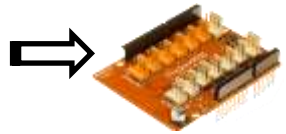
ⓘ [Page Web de la classe ELCD162 + Code de l'exemple](#)

8.5.4 Analog_4 : Mesurer la direction du vent avec un kit Weather Sensor Assembly p/n 80422

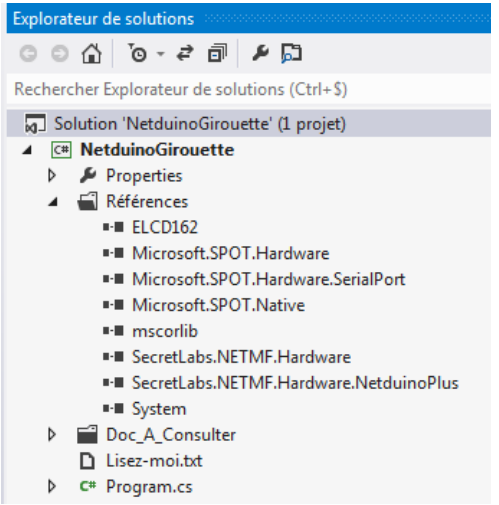
Code C#

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;


using Microtoolskit.Hardware.Displays;

namespace NetduinoGirouette
{
    public class Program
    {
        public static void Main()
        {
            // Variables
            var Angle = 0; // Position angulaire de la girouette

            // Girouette connectée à l'entrée I0 de la carte Tinkerkit
            AnalogInput Girouette = new AnalogInput(Cpu.AnalogChannel.ANALOG_0);
            Girouette.Scale = 3.3; // Mise à l'échelle (Tension d'alimentation = 3,3V)

             // LCD à connecter sur la sortie O5 de la carte Tinkerkit
             // Documentation de la classe ELCD162 : https://github.com/WebGE?tab=repositories
             // Création et initialisation d'un objet afficheur série ELCD-162 (par défaut : COM2, 19200, None, 8, 1)
            ELCD162 Lcd = new ELCD162(); Lcd.Init(); Lcd.ClearScreen();

      while (true)
            {
                var U_Girouette = Girouette.Read();

                if ((U_Girouette >= 0) && (U_Girouette < 0.45)){Angle = 90;}
                else if ((U_Girouette >= 0.45) && (U_Girouette < 0.76)) { Angle = 135; }
                else if ((U_Girouette >= 0.76) && (U_Girouette < 1.21)) { Angle = 180; }
                else if ((U_Girouette >= 1.21) && (U_Girouette < 1.76)) { Angle = 45;  }
                else if ((U_Girouette >= 1.76) && (U_Girouette < 2.28)) { Angle = 225; }
                else if ((U_Girouette >= 2.28) && (U_Girouette < 2.70)) { Angle = 0;   }
                else if ((U_Girouette >= 2.70) && (U_Girouette < 2.95)) { Angle = 315; }
                Else Angle = 270;

                Lcd.ClearScreen(); Lcd.PutString("Angle=" + Angle + "deg");
                Debug.Print("Tension = " + U_Girouette.ToString("N2") + "  Angle= " + Angle);
                Thread.Sleep(1000);
            }
        }
    }
}
```

Wheather Sensor



Shield TINKERKIT V2
sur Netduino plus 2



nuget

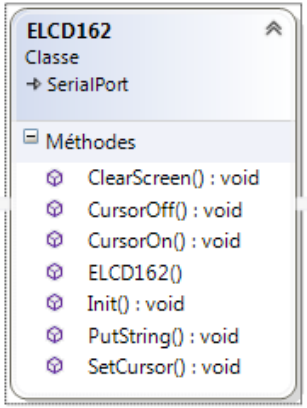
MicroToolsKit

Matériels

- Carte Netduino plus 2
- Sensor Shield TINKERKIT V2
- Weather Sensor Assembly p/n 80422
- Afficheur 2x16 + module ELCD162 COMFILE Série

github SOCIAL CODING

ⓘ **Page Web de la classe ELCD162**



Extrait de la documentation

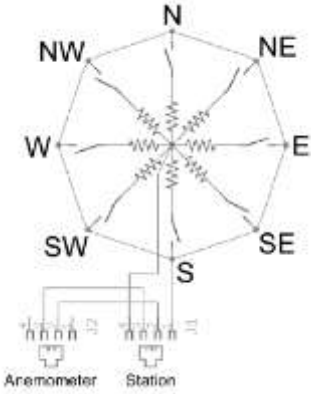
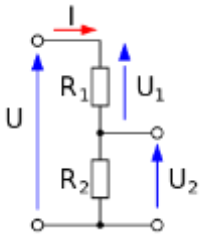


Schéma équivalent pour une position de la girouette



Alimentation U = 3,3V
R1 = 10kΩ
R2 = $R_{Girouette}$

U2 = Tension mesurée par le CAN de la carte Netduino.

Explications détaillées de l'exemple Analog_4

Développement dans la prochaine révision du document

9 La communication série

Pour transmettre une information, il faut un véhicule. Si nous utilisons communément la voix et l'écriture pour communiquer entre nous, les ordinateurs s'accommodent plutôt de **signaux** à deux états possibles !

Le véhicule de l'information (caractère "A") est donc un **signal binaire.**

Comme chacun le sait, un véhicule doit emprunter une voie de communication pour se déplacer. L'air est à la voix ce que le papier est à l'écriture !
Sous une forme électrique, les signaux affectionnent particulièrement les câbles métalliques, mais ils peuvent également être transmis dans l'air ou dans la matière sous une forme optique (lumière), acoustique (vibration), électromagnétique (onde), etc. Dans tous les cas la voie de communication est appelée **médium** ou **média**.
Sur de nombreux PC, le médium utilisé entre le clavier et l'unité centrale est encore un câble électrique !

Entre les équipements informatiques, la transmission de l'information se fait sur un **médium** sous la forme d'un **signal binaire**.

Mais un signal binaire ne peut prendre que deux états possibles "0" ou "1". Alors, comment représenter un caractère 'A' parmi les lettres de l'alphabet, les chiffres, etc.
Tous simplement en utilisant plusieurs "0" et plusieurs "1" qui assemblés, constituent un **code**.

Un code constitué de **n bits** peut représenter 2^n caractères différents. Il existe de nombreux codes binaires. L'un d'entre eux, appelé **code ASCII,** est universellement utilisé pour représenter les caractères alphanumériques.

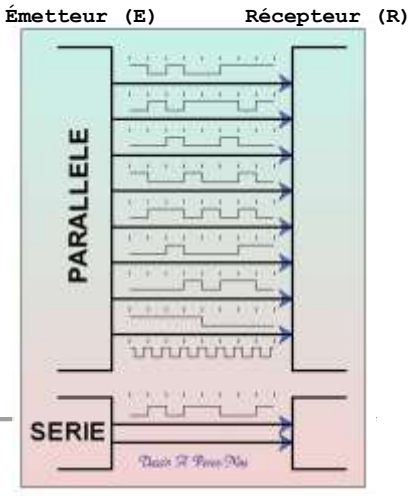
Exemple : En ASCII le caractère "A" est représenté par $01000001_{(2)}$ ou $41_{(16)}$.

Il existe **deux techniques** pour véhiculer les huit bits (**octet**) de ce code :

- Soit on les transmet simultanément (**en parallèle**) et alors au moins huit fils sont nécessaires dans le cas d'un médium filaire (plus un fil de masse (0V) !)

- Soit on les transmet les uns après les autres (**en série**) et alors un ou deux fils suffisent (plus un fil de masse (0V) !)

La transmission en parallèle ne concerne que les transmissions sur de très courtes distances (exemple : bus IEEE 488).



La transmission des données en série concerne donc la **majorité des communications** entre les équipements.

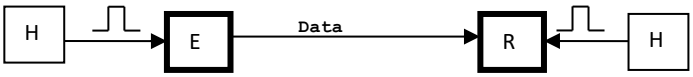
Pour que le **récepteur** « comprenne » les données qui lui sont transmises, il doit être **synchronisé** avec l'émetteur**.** C'est-à-dire « lire » les données à la **même vitesse** que l'émetteur les « écrit ».

Selon le type d'équipement utilisé la transmission série peut être **asynchrone** ou **synchrone**.

9.1 Communication série asynchrone

La synchronisation est faite en créant un signal d'horloge au niveau de l'émetteur et un signal d'horloge au niveau du récepteur. Chaque équipement possède sa propre horloge. La synchronisation des horloges se fait à chaque début de transfert.

On dit alors que la transmission est **ASYNCHRONE**.



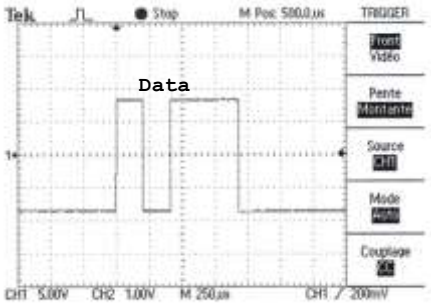


Figure 36 : Trame RS232

9.1.1 UART

9.1.1.1 Généralités

Un **UART**, pour ***Universal Asynchronous Receiver Transmitter***, est un émetteur-récepteur asynchrone universel.

Développement dans la prochaine révision du document

9.1.1.2 La communication série asynchrone des cartes Netduino

Développement dans la prochaine révision du document

9.1.1.3 Objet logiciel « SerialPort »

Développement dans la prochaine révision du document

9.1.1.4 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
UART_1	NetduinoUART	UART : Transmettre une valeur numérique via une liaison (**RS232)**	☒			☒
UART_2	NetduinoELCD_162	UART : Utiliser un afficheur LCD à commande série (**ELCD-162)**	☒	☒	☒	☒
UART_3a	NetduinoXBee_E	UART : Transmettre des données avec un module (**XBee)**	☒			☒
UART_3b	NetduinoXBee_R	UART : Recevoir des données avec un module (**XBee)**	☒	☒	☒	☒

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

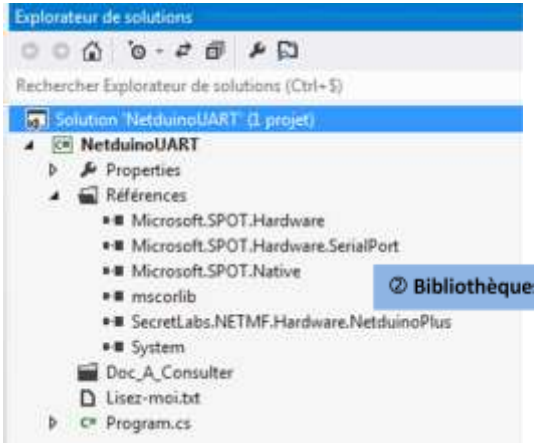
La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

Pour plus d'informations, consulter la description de la classe **SerialPort** sur MSDN

9.1.1.4.1 UART_1 : Transmettre une valeur numérique via une liaison RS232

Matériels
- Carte Netduino plus 2
- Sensor Shield TINKERKIT V2
- Module Digilent PmodRS232
- Câble RS232

Figure 37 : Module Digilent PmodRS232



② Bibliothèques

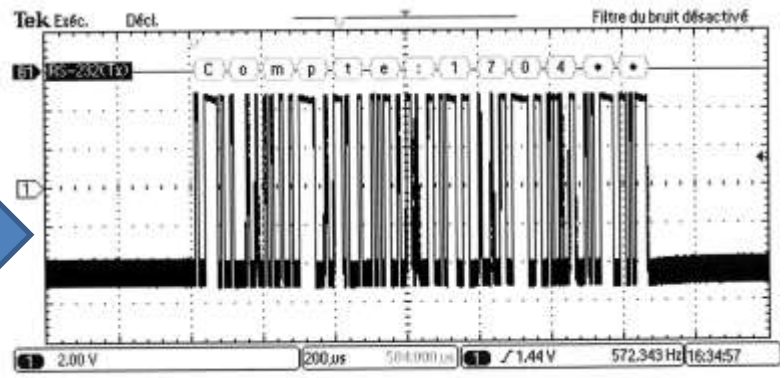
Code C#

```
using System.Text;
using System.Threading;
using System.IO.Ports;
using Microsoft.SPOT;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace NetduinoUART
{
    public class Program
    {
        public static void Main()
        {
        // Convertisseur TTL <--> RS232 relié au connecteur Serial de la carte TINKERKIT
            SerialPort UART = new SerialPort(SerialPorts.COM1, 115200);
            var counter = 0;
            UART.Open();

            while (true)
            {
                // Création d'une chaine de caractères
                string counter_string = "Compte:" + counter.ToString() + "\r\n";
                // Conversion de la chaîne en octets
                byte[] outBuffer = Encoding.UTF8.GetBytes(counter_string);
                // Envoie des octets au port série
                UART.Write(outBuffer, 0, outBuffer.Length);
                Debug.Print(counter_string);

                // Incrémentation du compteur
                counter++;
                // Attente de 100ms entre deux envois
                Thread.Sleep(100);
            }
        }
    }
}
```

Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S.

COM1
COM2
COM3
COM4

Débits binaires
2400
9600
19200
38400
57600
115200



Utilisation du **debugger** pour visualiser la valeur transmise.
Pas à pas
F10 : Step Over
F11 : Step into

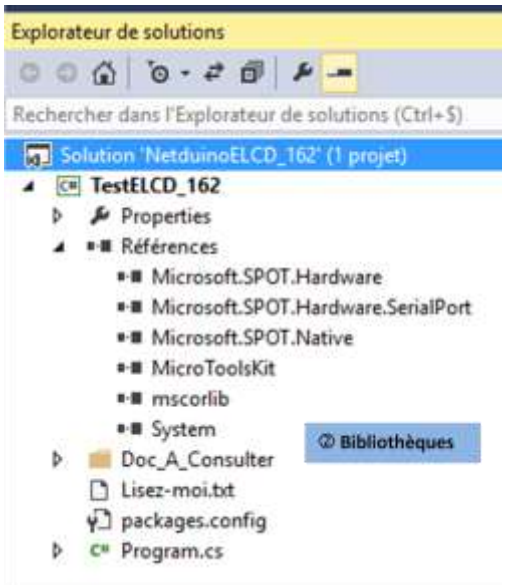
SerialPort Class (partielle)

- Close()
- DiscardInBuffer()
- DiscardOutBuffer()
- Flush()
- Open()
- Read(byte[], int, int)
- Seek(long, System.IO.SeekOrigin)
- SerialPort(string, int, System.IO.Ports.Parity, int, System.IO.Ports.StopBits)
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)
- SerialPort(string, int)
- SerialPort(string)

9.1.1.4.2 UART_2 : Utiliser un afficheur LCD à commandes séries (Module COMFILE ELCD-162)

Afficheur 2x16 équipé d'un
module ELCD162 COMFILE Série (19200b/s)

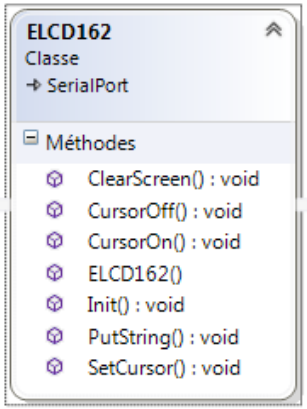
Shield TINKERKIT V2
sur Netduino plus 2



Code C#

```
using System.Threading;
using Microtoolskit.Hardware.Displays;

namespace NetduinoELCD_162
{
    public class Program
    {
        public static void Main()
            { // Documentation de la classe SerialELCD162 http://webge.github.io/ELCD162/
              // Pour accéder au COM2, relier l'afficheur au connecteur O5 de la carte Tinkerkit

             // Valeur à afficher
             var counter = 0;
             // Création d'un objet afficheur série ELCD-162 (par défaut : COM2, 19200, None, 8, 1)
             ELCD162 display = new ELCD162();

            // Initialisation de l'afficheur
             display.Init();display.ClearScreen();display.CursorOff();

             while (true)
             {
                 // Création d'une chaine de caractères
                 string counter_string = "Compte:" + counter.ToString() ;
                 // Envoie des octets au port série de l'afficheur
                 display.PutString(counter_string);
                 // Incrémentation du compteur
                 counter++;
                 // Attente de 1s entre deux envois
                 Thread.Sleep(1000); display.ClearScreen();
             }
        }
    }
}
```

Matériels
- Carte Netduino plus 2
- Sensor Shield TINKERKIT V2
- Afficheur 2x16 + module ELCD162 COMFILE Série



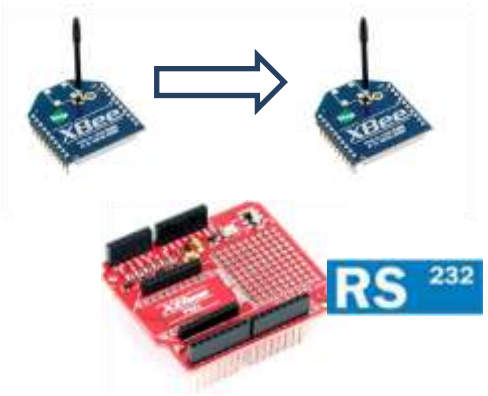
Page Web de la classe ELCD162

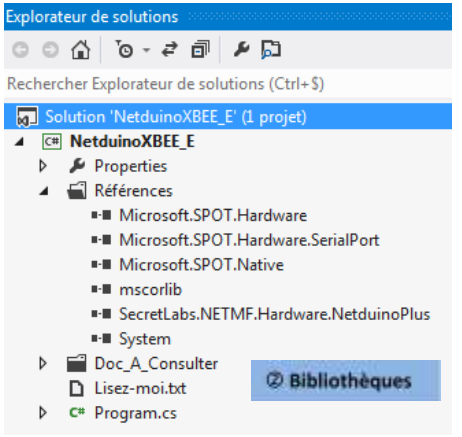
MicroToolsKit

9.1.1.4.3 UART_3a : Transmettre des données avec des modules XBEE (Emission/Réception)

SerialPort Class (partielle)

- Close()
- DiscardInBuffer()
- DiscardOutBuffer()
- Flush()
- Open()
- Read(byte[], int, int)
- Seek(long, System.IO.SeekOrigin)
- SerialPort(string, int, System.IO.Ports.Parity, int, System
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)
- SerialPort(string, int)
- SerialPort(string)

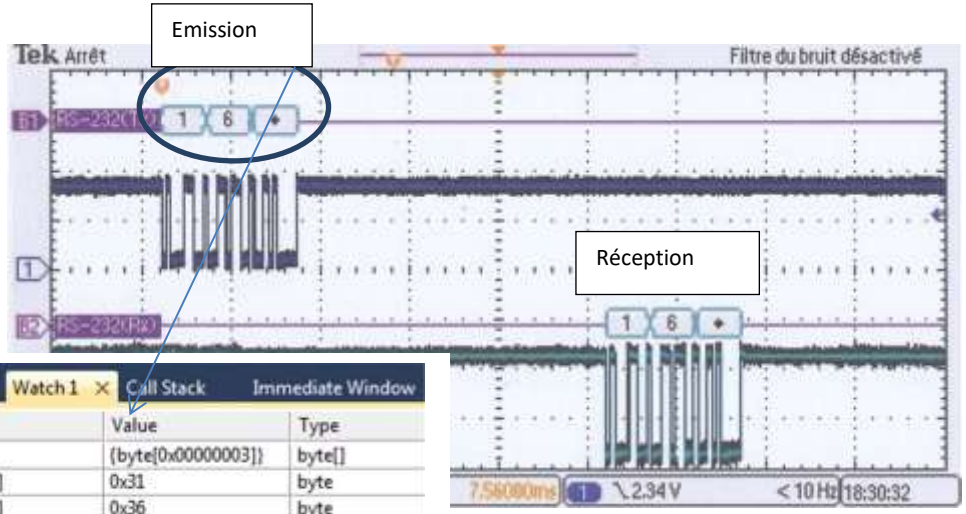




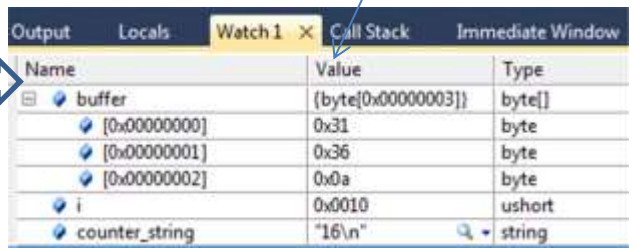
Code C#

```
using System;
using System.Text;
using System.IO.Ports;
using System.Threading;


using SecretLabs.NETMF.Hardware.NetduinoPlus;


namespace NetduinoXBEE_E
{
    public class Program
    {
        public static void Main()
        { // Documentation de la classe SerialPort http://msdn.microsoft.com/en-us/library/system.io.ports.serialport(v=vs.102).aspx
            var i = 0;// Valeur à transmettre

            // Emetteur
            // Création d'un port série pour la communication avec le module XBEE installé sur le shield Sparkfun. XBEE_RX sur Di0 et XBEE_TX sur Di1.
            SerialPort xbee_E = new SerialPort(SerialPorts.COM1, 9600, Parity.None, 8, StopBits.One);

            // Création d'une chaîne de caractère et d'un buffer d'émission
            string counter_string; byte[] OutBuffer;

            xbee_E.Open(); //Ouverture du port série

            while (true)
            {
                for (i = 0; i <= 65535; i++)
                {   // \n = <LF> (délimite la fin de la donnée transmise)
                    counter_string = i.ToString() + "\n";
                    outBuffer = Encoding.UTF8.GetBytes(counter_string);
                    xbee_E.Write(outBuffer, 0,outBbuffer.Length);
                    Thread.Sleep(2000); // Envoi toutes les 2s

                }
            }
        }
    }
}
```

Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S.



Transmission de "16 <CR>"



Matériels
- Carte Netduino
- Shield XBEE

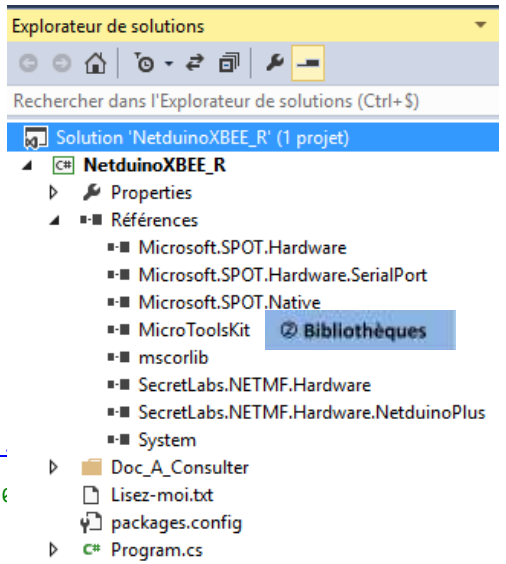
9.1.1.4.4 UART_3b : Transmettre des données avec des modules XBEE (Emission/Réception)

Code C#

```
using System.Text;
using System.IO.Ports;

using SecretLabs.NETMF.Hardware.NetduinoPlus;

using Microtoolskit.Hardware.Displays;

namespace NetduinoXBEE_R{
    public class Program{
        public static void Main(){
        // Documentation de la classe SerialPort http://msdn.microsoft.com/en-us/library/system.io.ports.serialport(v=vs.102)
        // Récepteur (Message = chaîne de caractères représentant une valeur comprise entre 0 et 65535 suivi de \n)
        // Création d'un port série pour la communication avec le module XBEE installé sur le shield Sparkfun. XBEE_RX sur Di6
         SerialPort xbee_R = new SerialPort(SerialPorts.COM1, 9600, Parity.None, 8, StopBits.One);

         // Pour accéder au COM2, relier l'afficheur au connecteur KK O5 de la carte Tinkerkit
        // Création d'un port série logiciel pour l'afficheur LCD
        ELCD162 display = new ELCD162();

        // Création du buffer de réception
        byte[] inBuffer = new byte[6] { 0, 0, 0, 0, 0, 0 };

        // Initialisation du LCD
        display.Init(); display.ClearScreen(); display.CursorOff();
        int pos = 0; // position dans inBuffer
        xbee_R.Open(); // Ouverture du port série COM1

        while (true)
        {
          bool Aff = false;
          // Copie du message reçu dans inBuffer
          do
           {
             xbee_R.Read(inBuffer, pos, 1);
             if (inBuffer[pos]!=0x0a) {
               pos++; // Avance la position dans inBuffer
             }        // si le caractère fin de ligne '\n' n'est pas atteint
             else{
              pos = 0; // Réinitialise la position dans inBuffer
              Aff = true; // Autorise l'affichage de la donnée
             }
          } while (xbee_R.BytesToRead > 0);}
```

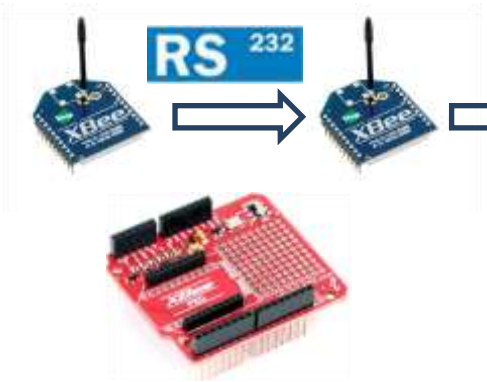
SerialPort Class (partielle)

- Close()
- DiscardInBuffer()
- DiscardOutBuffer()
- Flush()
- Open()
- Read(byte[], int, int)
- Seek(long, System.IO.SeekOrigin)
- SerialPort(string, int, System.IO.Ports.Parity, int, System.IO.Ports.StopBits)
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)
- SerialPort(string, int)
- SerialPort(string)



nuget

MicroToolsKit



Afficheur 2x16 équipé d'un [module ELCD162 COMFILE Série](#) (19200b/s)

RS 232

Matériels

- Carte Netduino
- Shield XBEE
- Afficheur 2x16 + module ELCD162 COMFILE Série

github SOCIAL CODING

Page Web de la classe ELCD162

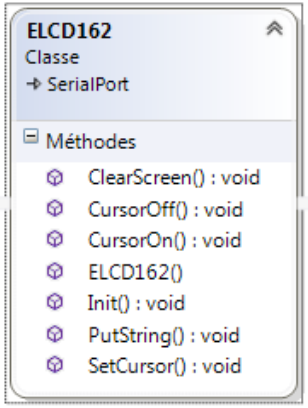
Suite du code

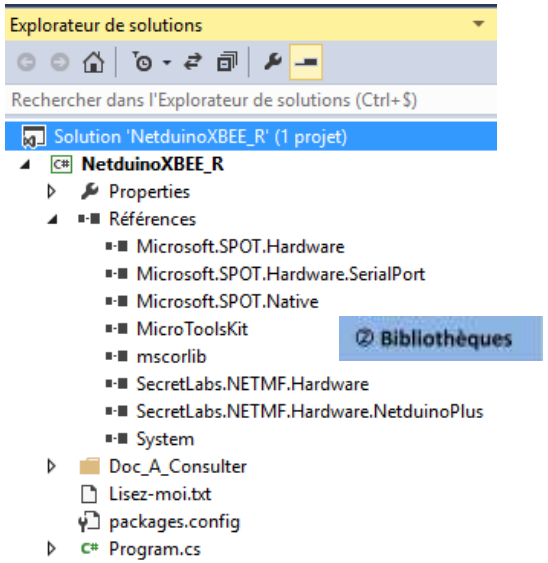
UART_3b : Transmettre des données avec des modules XBEE (Emission/Réception) suite

```
        // Affichage de la valeur sur le LCD
                    if (Aff)
                    {
                        char[] chars = Encoding.UTF8.GetChars(inBuffer);
                        int i = MessageSize(inBuffer);
                        string str = new string(chars, 0, i);
                        display.ClearScreen(); display.PutString(str);
                        Clear_Buffer(inBuffer);
                    }
                }
            }

            private static int MessageSize(byte[] inBuffer)
            {
                int i;
                for (i = 0; i < inBuffer.Length; i++)
                {
                    if ((inBuffer[i] < 0x30) || (inBuffer[i] > 0x39))
                    {
                        break;
                    }
                }
                return i;
            }

            private static void Clear_Buffer(byte[] inBuffer)
            {
                for (int j = 0; j < inBuffer.Length; j++)
                {
                    inBuffer[j] = 0;
                }
            }
        }
    }
```

Page Web de la classe ELCD162

MicroToolsKit

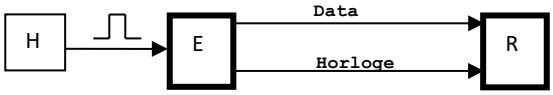




9.2 Communication série synchrone

La synchronisation peut être faite en **transmettant un signal d'horloge** synchronisé avec le signal de donnée. L'horloge est alors unique et commune à l'émetteur et au récepteur.

On dit que la transmission est **SYNCHRONE**.



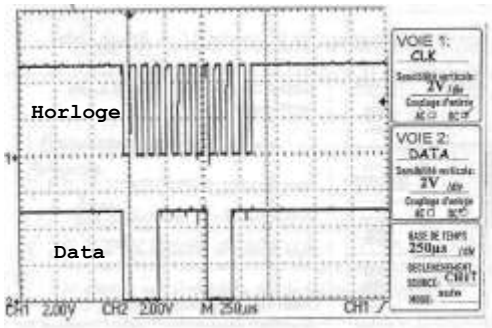


Figure 38 : Trame clavier IBM

9.2.1 Le bus I2C

9.2.1.1 Généralités

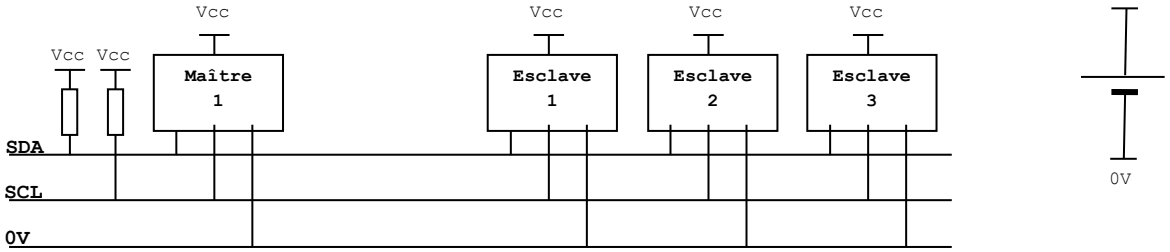
Le bus I²C a été créé au début des années 80 par RTC Philips afin d'apporter une solution simple et peu coûteuse à la **communication entre les circuits intégrés numériques** à l'intérieur des appareils grand public (**téléviseurs, magnétoscopes**, etc.). Le principal avantage du bus I²C est de **limiter le nombre de liaisons** entre les circuits intégrés.

Conventions d'écriture :	**« 0 » : 0 logique**	**« 1 » : 1 logique**

- **Structure et organisation du bus I2C**

  Le bus I²C est un bus de type **série synchrone** ne nécessitant que deux signaux.
  - **SDA** (serial data), le signal de donnée.
  - **SCL** (serial clock), le signal d'horloge.

  Ce bus permet la communication entre un **circuit maître et un circuit esclave**. Le montage peut comporter plusieurs esclaves. **Le maître est le circuit qui émet le signal d'horloge**. Les données peuvent circuler dans les deux sens sur le fil des données, de sorte que chaque circuit, qu'il soit maître ou esclave peut servir **d'émetteur ou de récepteur** (de données).
  Les différents circuits sont placés en **parallèle** sur les lignes SDA et SCL comme sur le schéma suivant :

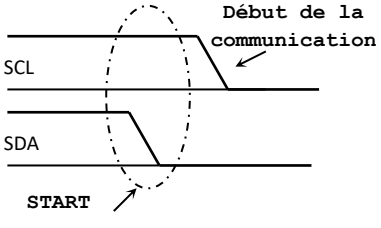


Au repos, c'est à dire lorsqu'aucun circuit n'émet, les signaux SDA et SCL sont au niveau logique haut. Pour éviter les conflits, un maître qui veut émettre doit attendre que le bus soit au repos.

- **Le protocole I2C**

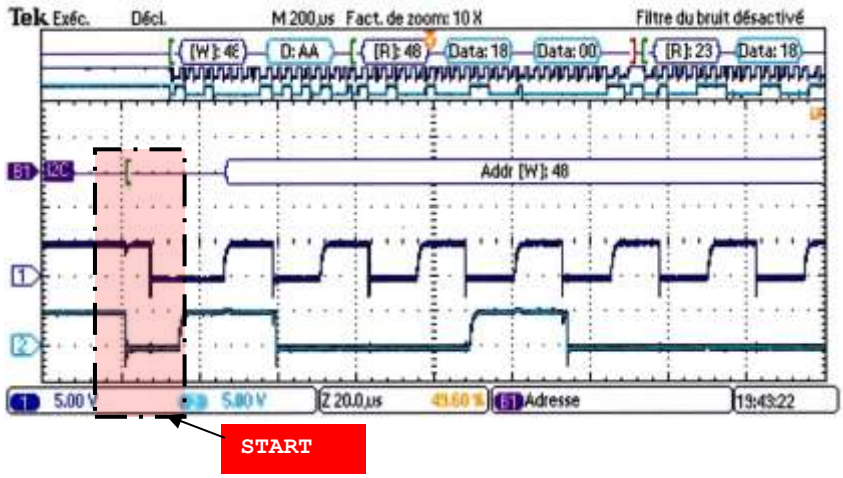
  - ✓ **Prise de contrôle du bus par un maître et début de la communication**

    Le bus étant initialement libre, SDA et SCL **sont au niveau haut (« 1 » logique)**. Un maître prend le contrôle du bus en effectuant un **START** : c'est-à-dire qu'il met SDA à « 0 », SCL restant à « 1 ». Au cours de la communication, l'horloge SCL est envoyée par le maître et SDA ne peut changer d'état que lorsque SCL est à « 0 ».



Exemple de mesure : **START** (Oscilloscope TEKTRONIX MSO2014)

1 : SCL
2 : SDA

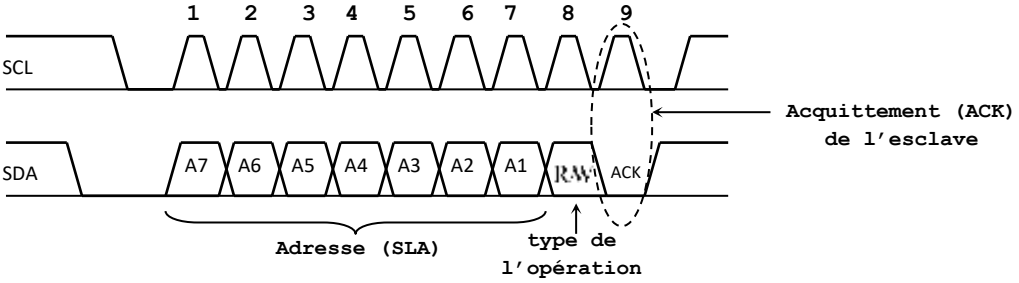


➢ **Le maître choisit l'esclave avec lequel il souhaite communiquer et le type d'opération**

Pour sélectionner un esclave lorsqu'il a pris le contrôle du bus, le maître envoie **un premier octet (octet d'adressage (adress byte) noté [2*SLA+R/$\overline{W}$] dans ce document). Cet octet est constitué** :

- de l'**adresse** sur **sept bits (SLA**),
- d'un bit de **lecture**(**R**ead)**/écriture**(**W**rite) en abrégé (**R/$\overline{W}$**) qui indique si le maître souhaite faire une opération de lecture ou d'écriture sur l'esclave.

Simultanément, le maître génère le signal d'horloge SCL comportant **neuf périodes**.



Tous les esclaves observent l'adresse envoyée par le maître. Pendant la **neuvième période** de SCL, l'esclave qui reconnaît son adresse répond au maître en maintenant SDA au niveau logique bas : c'est le bit d'acquittement appelé **ACK** (acknoledge). L'esclave est prêt à communiquer avec le maître. Les autres esclaves restent en attente.

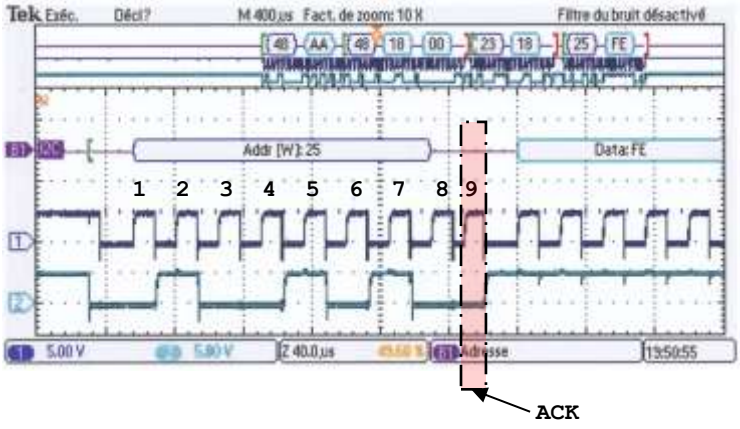
Si le bit **R/$\overline{W}$** envoyé par le maître est à :

- **« 1 » (lecture), l'esclave prend la parole.**
- **« 0 » (écriture), l'esclave se met à l'écoute du maître.**

Dans les exemples ci-dessous, l'esclave est « contacté » avec une adresse SLA = $25_{(16)}$. Dans le premier chronogramme, il acquitte (**ACK**) cette adresse car il a été correctement configuré.

Exemple de mesure : **Acquittement (ACK)** (Oscilloscope TEKTRONIX MSO2014)

1 : SCL
2 : SDA

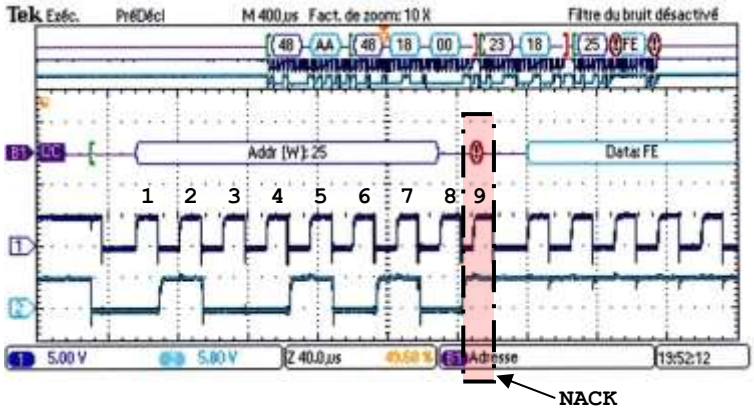


ACK = Acknowledge

Dans l'exemple ci-dessous, , l'esclave est configuré avec une adresse différente de $25_{(16)}$ aussi : il n'acquitte pas (**NACK**) !

Exemple de mesure : **Pas d'acquittement (NACK)** (Oscilloscope TEKTRONIX MSO2014)

1 : SCL
2 : SDA



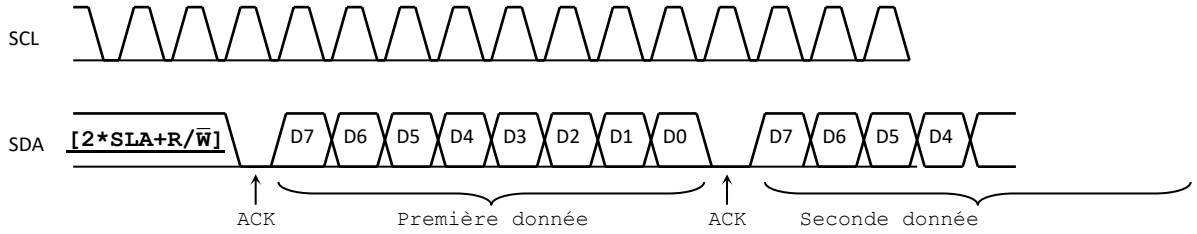
NACK = No Acknowledge

➢ **Envoi d'une ou plusieurs données**

A la suite de l'octet d'adressage, chaque donnée est codée sur un octet envoyé bit par bit sur la ligne SDA. Au cours de la communication, plusieurs données peuvent être transmises.

- s'il s'agit d'une opération de **lecture**, les données sont envoyées au maître par l'esclave. **L'esclave est émetteur, le maître est récepteur**.
- s'il s'agit d'une opération **d'écriture**, les données sont envoyées à l'esclave par le maître. **Le maître est émetteur, l'esclave est récepteur**.

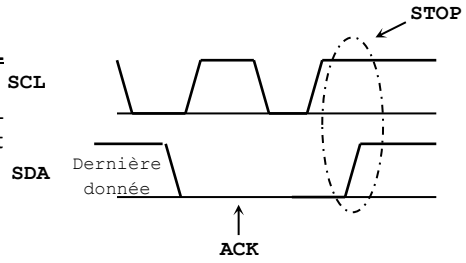
L'émetteur envoie d'abord les huit bits du premier octet puis le récepteur acquitte en **maintenant la ligne SDA à « 0 » (bit ACK)**. L'émetteur envoie ensuite les huit bits du second octet et le récepteur acquitte. Le même cycle se répète jusqu'à ce que toutes les données aient été envoyées. **Pendant la communication, l'horloge SCL est générée par le maître**.



➢ **Fin de communication et libération du bus**

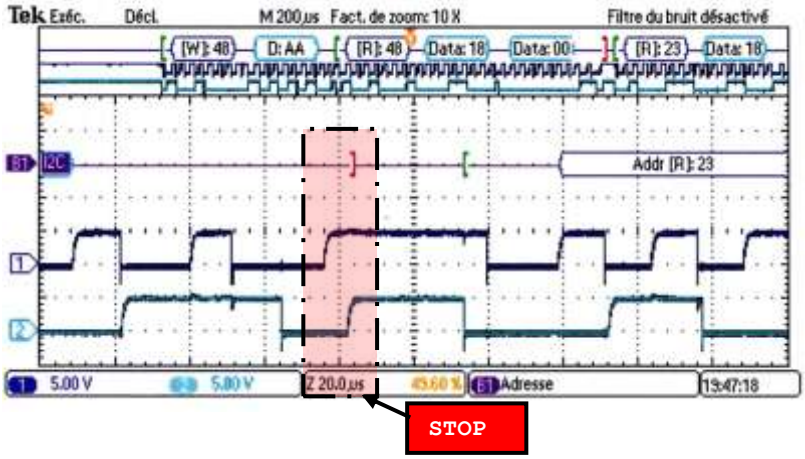
Pendant la communication, les changements d'état sur SDA se produisent lorsque **SCL est à « 0 »**.

Pour mettre fin à la communication, le maître effectue un **STOP** : il met d'abord SCL à « 1 » puis ramène SDA à « 1 ». C'est le changement d'état de SDA alors que SCL est à « 1 » qui met fin à la communication.

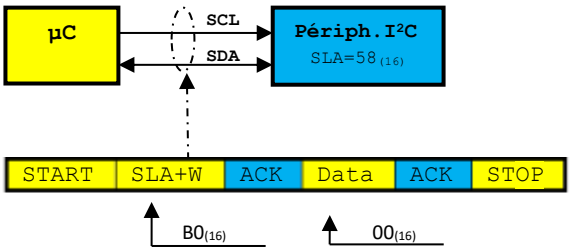


Exemple de mesure : **STOP** (Oscilloscope TEKTRONIX MSO2014)

1 : SCL
2 : SDA



➢ **Exemple de transaction effectuée entre un microcontrôleur et un périphérique I²C**



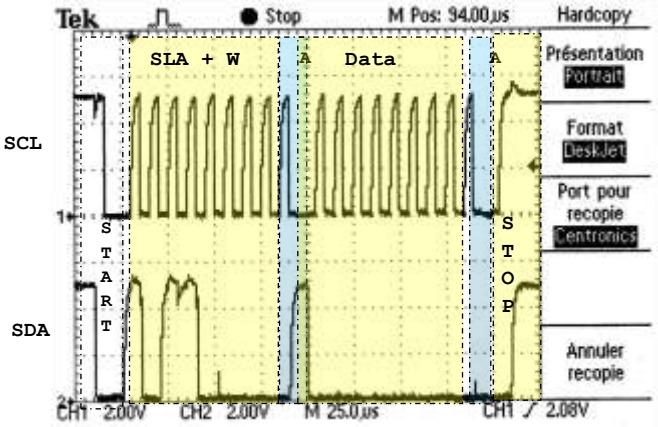


Figure 39 : Trame I2C

➢ **Synchronisation du maître et de l'esclave**

Habituellement, la ligne SCL fonctionne en sortie pour un maître et en entrée pour un esclave car c'est le maître qui génère le signal d'horloge. Mais lorsqu'un **esclave veut obliger un maître à ralentir la communication, l'esclave maintient la ligne SCL à « 0 »** pour empêcher le maître de la ramener à « 1 ». La communication reprend son cours lorsque l'esclave libère la ligne SCL. Cette opération s'appelle la **synchronisation**.

➢ **Caractéristiques du bus I²C (mode standard)**

- Bus **série synchrone bi directionnel 8 bits** (sur deux lignes SDA et SCL)
- Débits : 100kbits/s, 400kbits/s

➢ **Exemples de circuits intégrés pour le bus I²C**

Le maître est souvent un **microcontrôleur possédant une interface I²C**.

Les esclaves sont des circuits intégrant une interface pour le bus I²C et réalisant la même fonction que des circuits intégrés couramment utilisés en électronique :

- CAN ou CNA : PCF8591
- Mémoire vive : PCF8570
- EEPROM : PCF8582
- Horloge calendrier: PCF8583
- Port d'entrées / sorties parallèles (8): PCF8574
- Afficheurs LCD : PCF2119
- ….

Bibliographie

"Le bus I2C: Principes et mise en oeuvre" D. PARET DUNOD

Webographie

http://fr.wikipedia.org/wiki/I2C

9.2.1.2 Le bus I²C des cartes Netduino

Les classes associées aux différents matériels se chargent de gérer le bus I²C.

9.2.1.3 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
I2C_1	NetduinoPCF8574	I²C : Chenillard sur huit Leds reliées à un port d'E/S **PCF8574.**	☒	☒	☒	☒
I2C_2	NetduinoI2CLCD	I²C : Commander un afficheur LCD à circuit **PCF2119.**	☒	☒	☒	☒
I2C_3	NetduinoSRF08US	I²C : Mesurer une distance avec un télémètre à ultrasons **SRF08**	☒	☒	☒	☒
I2C_4	NetduinoI2CLEDBP	I²C : Recopier l'état de boutons poussoirs sur des Leds via des **PCF8574** (carte SSI).	☒	☒	☒	☒
I2C_5	NetduinoHMC6352	I²C : Lire la direction donnée par une boussole **HMC6352.**	☒	☒	☒	☒
I2C_6	NetduinoTMP102	I²C : Mesurer la température ambiante avec un capteur **TMP102.**	☒	☒	☒	☒
I2C_7	NetduinoMD25	I²C : Commander deux motoréducteurs à C.C. équipés d'encodeurs avec une carte **MD25.**	☒	☒	☒	☒
I2C_8	NetduinoTSL2561	I²C : Mesurer la luminosité ambiante avec un capteur **TSL2561.**	☒	☒	☒	☒
I2C 9	NetduinoMLX90614	I²C : Mesurer la température d'un objet avec un capteur **MLX90614.**		☒	☒	
I2C 10	NetduinoMCP3424	I²C : Acquérir des données issues de capteurs analogique avec un CAN **MCP3424.**		☒	☒	

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est inclue dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

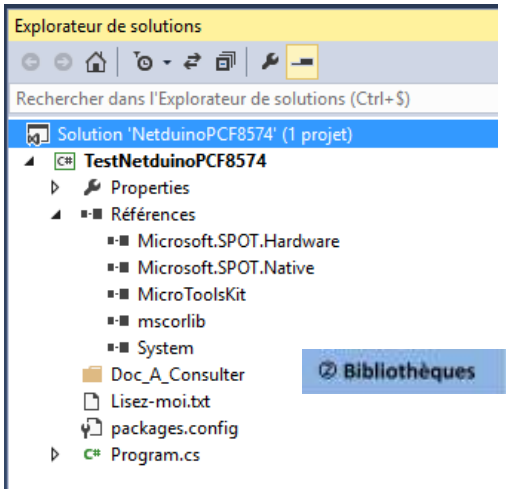
La photo du montage à réaliser est dans le sous répertoire Doc_A_Consulter du répertoire du projet.

Pour plus d'informations, consulter la description de la classe **I²C** sur MSDN

9.2.1.3.1 I2C_1 : Chenillard sur huit LEDs reliées à un port d'entrées, sortie (E/S) PCF8574A

Shield TINKERKIT V2
sur Netduino plus 2

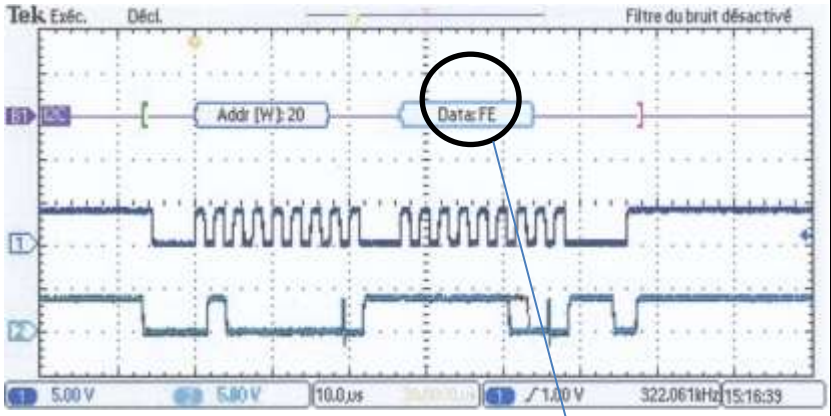
X 8

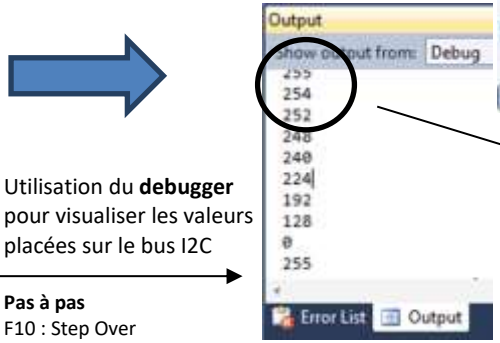


Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.IO;

namespace TestNetduinoPCF8574
{
    public class Program
    {
        public static void Main()
        { // PCF8574(@=0x20 to 0x27)        PCF8574A(@=0x38 to 0x3f)
        // Pour accéder au bus I2C, relier le connecteur TWI de la carte Tinkerkit au Hub I2C
           // et le hub I2C à l'IHM SSI (voir la photo dans Doc_A_Consulter)
           // Paramètres du bus I2C de l'IHM SSI
            byte addLeds_I2C = 0x38; // Adresse (7 bits) du PCF8574A relié aux Leds
            Int16 Freq = 100; // Fréquence d'horloge du bus I2C en kHz

            byte stateLED = 0xFE; // Etat initial des LED. (Un 0 logique => Led éclairée)

            // Création d'un objet Leds
            var Leds = new PCF8574(addLeds_I2C, Freq);

            Leds.Write(stateLED); // Initialisation de l'état des Leds
            Thread.Sleep(500);

            while (true)
             {
                 // Modification de l'état des LED
                 if (stateLED != 0)
                 {
                     stateLED = (byte)(stateLED << 1);
                 }
                 else
                 {
                     stateLED = 0xFF;
                 }

                 //Ecriture sur les Leds
                 Leds.Write(stateLED);
                 Debug.Print(stateLED.ToString());
                 Thread.Sleep(500); // Pour la simulation
             }
        }
    }
}
```


Page Web de la classe PCF8574





Utilisation du **debugger** pour visualiser les valeurs placées sur le bus I2C

Pas à pas
F10 : Step Over
F11 : Step into

Remarque : L'éclairage d'une LED est commandé par un niveau logique 0.

Matériels
- Carte Netduino + Shield Tinkerkit
- IHM SSI (photo page suivante) + Hub I2C

MicroToolsKit

9.2.1.3.2 I2C_2 : Commander un afficheur LCD (PCF2119)

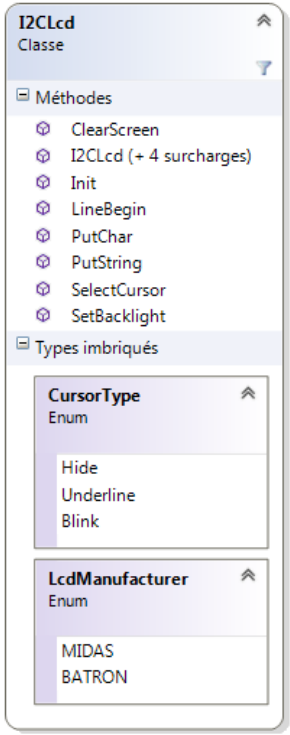
Code C#

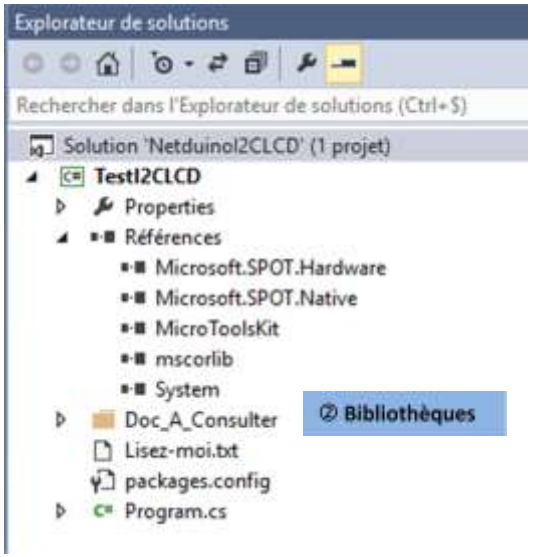
```
using System;
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Displays;

namespace TestNetduinoI2CLCD
{
    public class Program
    {
     public static void Main()
     { // Pour accéder au bus I²C, relier le LCD au hub I2C et le hub I²C au connecteur TWI de la carte Tinkerkit.
       byte InitJauge = 0x5A; // Etat initial d'un caractère personnalisé "jauge"
       UInt16 Freq = 100; // Fréquence d'horloge du bus I2C en kHz

       // Création d'un objet I2CLcd MIDAS MC21605E6W http://www.farnell.com/datasheets/1722538.pdf
       // Documentation de la classe I2CLcd : http://webge.github.io/LCDI2C/
       I2CLcd lcd = new I2CLcd(I2CLcd.LcdManufacturer.MIDAS, Freq);

       // Initialisation du Lcd I2C
       lcd.Init(); lcd.ClearScreen();

       // Message
       lcd.PutString(3, 0, "SSI...");
       lcd.PutChar(11, 0, 0x4E);
       lcd.PutString(2, 1, "Bonjour");
       // Jauges linéaires virtuelles
       for (byte w = InitJauge; w < 0x60; w++)
       lcd.PutChar((byte)(w - 0x51), 1, w);

       while (true)
       {
           // Démo Widgets
           lcd.PutChar(14, 0, 0x11); lcd.PutChar(15, 0, 0x21);
           lcd.PutChar(13, 0, 0x4C);
           Thread.Sleep(200);
           lcd.PutChar(14, 0, 0x21);.PutChar(15, 0, 0x11);
           lcd.PutChar(13, 0, 0x4B);Thread.Sleep(200);

           // Démo jauge linéaire virtuelle
           lcd.PutChar(0, 0, (byte)InitJauge);
           InitJauge++;
           if (InitJauge > 0x5F) InitJauge = 0x5A;
       }
    }
  }
}
```

Shield TINKERKIT V2
sur Netduino plus 2

Matériels
- Carte Netduino + Shield Tinkerkit
- IHM SSI (LCD I2C 2 x16)
- Hub I2C

MicroToolsKit





IHM SSI

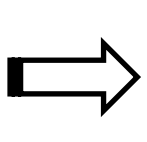
Page Web de la classe I²CLCD

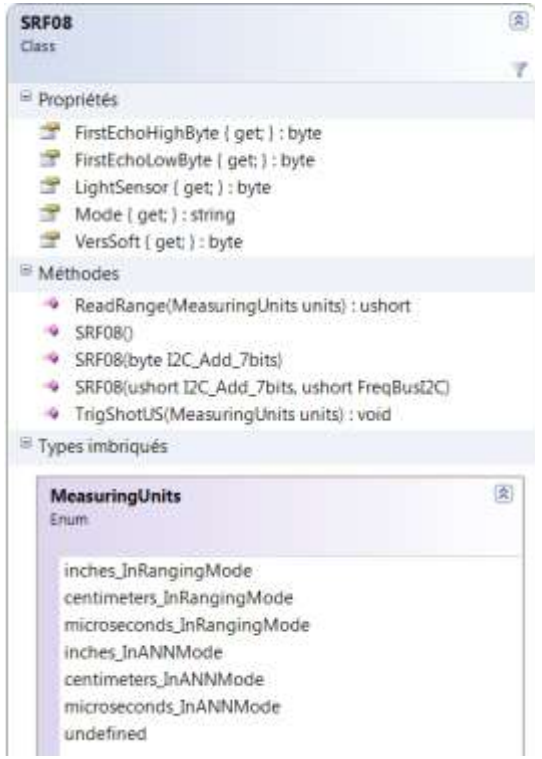
9.2.1.3.3 I2C_3 : Mesurer une distance avec un télémètre à ultrasons SRF08

Description : Mesure de la distance entre le télémètre et un obstacle. Affichage sur un LCD à commandes séries (RS232).

Page Web de la classe SRF08

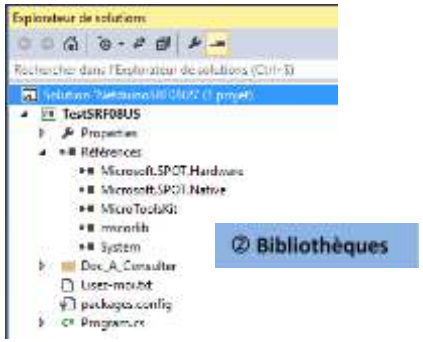
Shield TINKERKIT V2
sur Netduino plus 2





Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Sensors;

namespace TestNetduinoSRF08US
{ // Utiliser une alimentation secteur
    public class Program
    {      // Pour accéder au bus I2C, relier le télémètre SRF08 au connecteur TWI de la
           // carte Tinkerkit. Placer des résistances de rappel (3,3k) entre le +5V et les sorties SCL et SDA
           byte addTelem_I2C = 0x70; // Adresse (7 bits) du télémètre SRF08
           UInt16 Freq = 400; // Fréquence d'horloge du bus I2C en kHz (Rp = 3,3K au +5V)

           // Création d'un objet télémètre SRF08
           SRF08 I2CTelemeter = new SRF08(addTelem_I2C, Freq);

           // Affichage de la version du software du télémètre
           Debug.Print("VerSoft: " + I2CTelemeter.VersSoft);
           // Lecture et affichage de la luminosité
           Debug.Print("Light: " + I2CTelemeter.LightSensor);
           // Affichage du mode de mesure: Ranging ou ANN
           Debug.Print("Mode: " + I2CTelemeter.Mode);
           Debug.Print("_____________________________________");
           Thread.Sleep(2000);

           while (true){
               // Déclenchement, lecture et affichage de la distance en cm
               Debug.Print("Distance: " + I2CTelemeter.ReadRange(SRF08.MeasuringUnits.centimeters_InRangingMode) + "cm");
               // Déclenchement, lecture des registres correspondant au premier écho
               Debug.Print("1st Echo HighByte: " + I2CTelemeter.FirstEchoHighByte + "  " + "1st Echo LowByte: " + I2CTelemeter.FirstEchoLowByte);
               // Déclenchement, lecture et affichage de la distance en inch
               Debug.Print("Distance: " + I2CTelemeter.ReadRange(SRF08.MeasuringUnits.inches_InRangingMode) + "inches");
               // Lecture des registres correspondant au premier écho
               Debug.Print("1st Echo HighByte: " + I2CTelemeter.FirstEchoHighByte + "  " + "1st Echo LowByte: " + I2CTelemeter.FirstEchoLowByte);
               // Déclenchement, lecture et affichage de la distance en microsecondes
               Debug.Print("Distance: " + I2CTelemeter.ReadRange(SRF08.MeasuringUnits.microseconds_InRangingMode) + "µs");
               // Lecture des registres correspondant au premier écho
               Debug.Print("1st Echo HighByte: " + I2CTelemeter.FirstEchoHighByte + "  " + "1st Echo LowByte: " + I2CTelemeter.FirstEchoLowByte);
               // Lecture et affichage de la luminosité
               Debug.Print("Light: " + I2CTelemeter.LightSensor);
               Debug.Print("_____________________________________");
               // Afichage du mode de mesure: Ranging ou ANN
               Debug.Print("Mode: " + I2CTelemeter.Mode);
               Thread.Sleep(1000);
           }
       }
    }
}
```



Utilisation du **debugger**

Pas à pas
F10 : Step Over
F11 : Step into

```
_____________________________________
Distance: 55cm
1st Echo HighByte: 0  1st Echo LowByte: 55
Distance: 21inches
1st Echo HighByte: 0  1st Echo LowByte: 21
Distance: 6824µs
1st Echo HighByte: 12  1st Echo LowByte: 168
Light: 139
_____________________________________
Mode: Ranging
```

Matériels
- Carte Netduino
- **Sensor Shield TINKERKIT V2** + Hub I2C
- Transducteur à ultrasons SRF08

MicroToolsKit

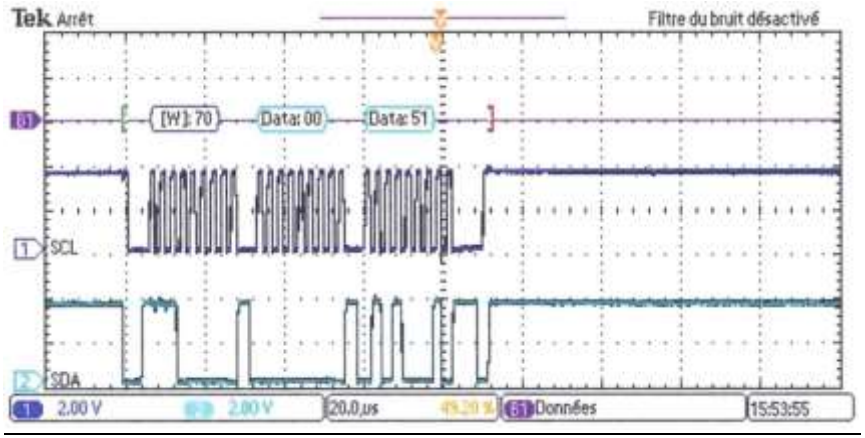
I2C_3 : Mesurer une distance avec un télémètre à ultrasons SRF08 (Suite)

Chronogrammes

Ces chronogrammes ont été analysés à partir des informations données dans la documentation du circuit SRF08. [http://goo.gl/1ktzdG]

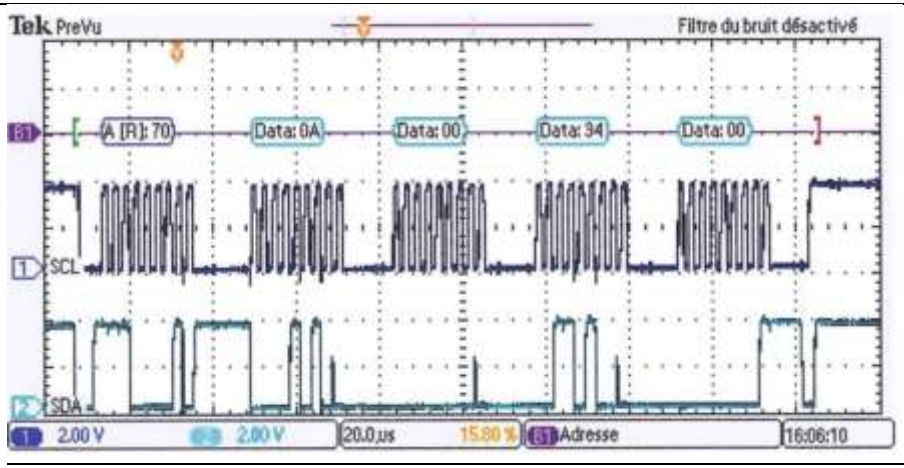
<table>
<tr><th>Méthode</th><th>Transaction</th><th>Chronogrammes</th></tr>
<tr><td rowspan="3">ReadRange(SRF08.UnitType.centimeters)

Remarque : Une méthode => 3 opérations</td><td>(1) Déclenchement du tir US

Start : [
@SLA : 70h
N° registre : 00h
Commande : 51h
Stop:]</td><td> 

Le bit de lecture écriture n’est pas inclus dans la valeur affichée : SLA=70h. (h : valeur en hexadécimal)</td></tr>
<tr><td colspan="2">(2) Attente 75ms</td></tr>
<tr><td>(3)Lecture de la mesure

Start : [
@SLA :70h
Reg0 (Software Rev): 0Ah
Reg1 (Light Sensor): 00h
Reg2 (1st Echo low) : 34h
Reg3(1st Echo High) :00h
Stop :]

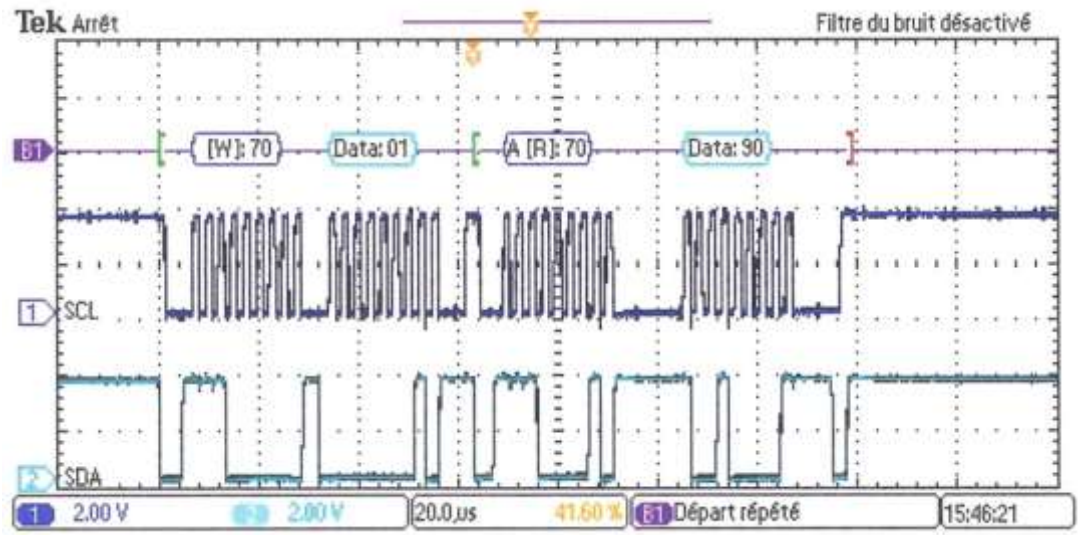
Distance = 0034h = 52cm</td><td> 

Le bit de lecture écriture n’est pas inclus dans la valeur affichée : SLA=70h. (h : valeur en hexadécimal)</td></tr>
</table>

I2C_3 : Mesurer une distance avec un télémètre à ultrasons SRF08 (Suite)

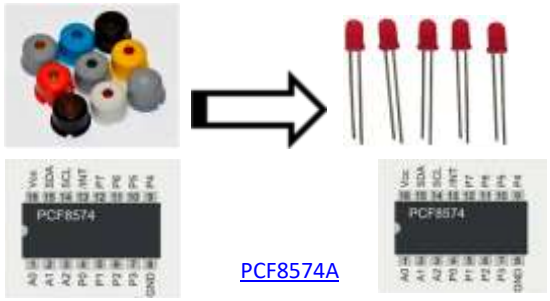
Chronogrammes

Ces chronogrammes ont été analysés à partir des informations données dans la documentation du circuit SRF08. [http://goo.gl/1ktzdG]

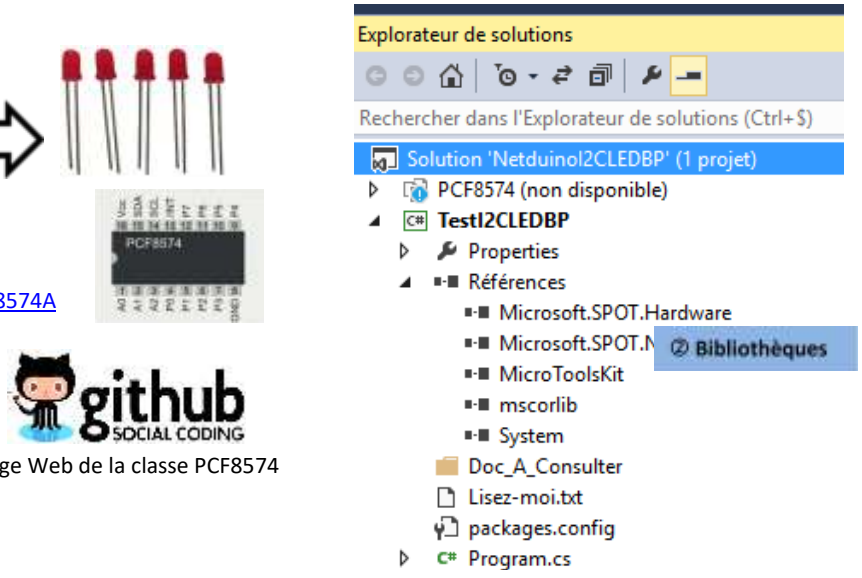
Propriété	Transaction	Chronogrammes
`I2CTelemeter.LightSensor`	**Start: [** SLA: 70h n°registre: 01h **Repeated Start : [** @SLA : 70h Donnée située en n° registre : 90h **Stop** **Luminosité = 144** (nombre entre 0 et 255)	 **Le bit de lecture écriture n'est pas inclus dans la valeur affichée : SLA=70h. (**h : valeur en hexadécimal**)**

9.2.1.3.4 I2C_4 : Recopier l'état de boutons poussoirs (PCF8574A) sur des LED (PCF8574A)

Shield TINKERKIT V2
sur Netduino plus 2



PCF8574A



Page Web de la classe PCF8574

Code C#

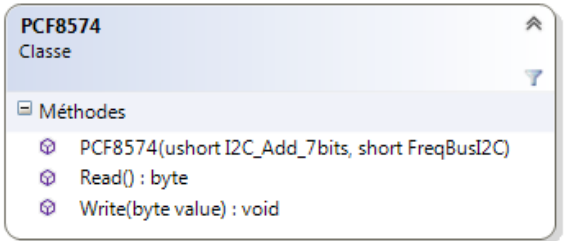
```
using System;

using Microtoolskit.Hardware.IO;

namespace TestNetduinoI2CLEDBP
{
    public class Program
    {
        public static void Main()
        { // Pour accéder au bus I²C, relier les PCF8574 au connecteur TWI de la
          // carte Tinkerkit. Placer des résistances de rappel (3,3k) entre
          // le +5V et les sorties SCL et SDA

            // Paramètres du bus I2C
            byte addLeds_I2C = 0x38; // Adresse (7 bits) du PCF8574A relié aux LEDs de la carte SSI
            byte addBPs_I2C = 0x3F;  // Adresse (7 bits) du PCF8574A relié aux BPs de la carte SSI
            Int16 FreqLed = 100; // Fréquence d'horloge du bus I2C en kHz
            Int16 FreqBP = 200; // Elle peut être différente pour chaque composant

            // Création d'un objet "Leds"
            PCF8574 Leds = new PCF8574(addLeds_I2C, FreqLed);

            // Création d'un objet "BPs"
            PCF8574 BPs = new PCF8574(addBPs_I2C, FreqBP);


            byte stateLED = 0xFF; // Etat initial des LED
            Leds.Write(stateLED);

            while (true)
             {
                 // Lecture des BPs
                 stateLED = BPs.Read();
                 stateLED = (byte)~stateLED;
                 // Ecriture sur les Leds
                 Leds.Write(stateLED);
             }
        }
    }
}
```

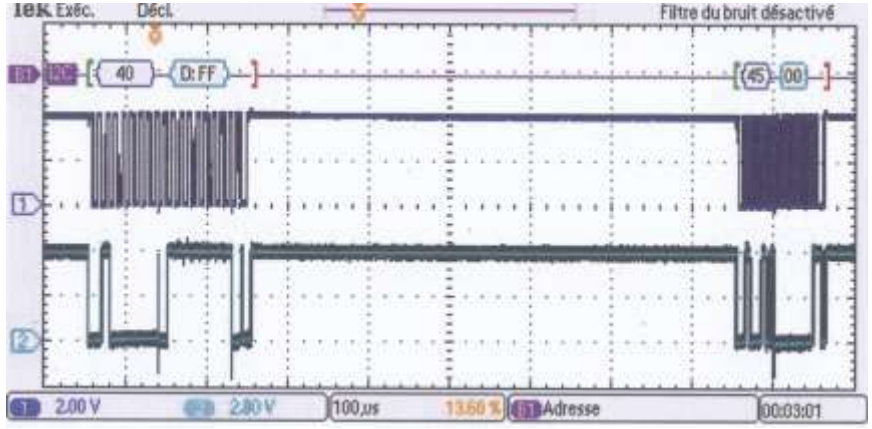
Carte SSI	
BP	**Valeur**
+	1
-	8
↑	2
↓	4
OK	32
ECHAP	128
SET	16
ENTR	64
aucune	0

Matériels
- Carte Netduino
- Sensor **Shield TINKERKIT V2**
- Carte IHM SSI

Carte IHM

MicroToolsKit





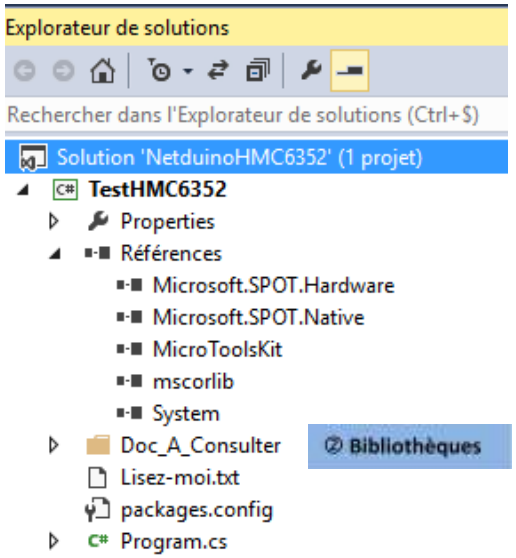
Modification de la fréquence du BUS I2C !

9.2.1.3.5 I2C_5 : Lire la direction donnée par une boussole HMC6352

Code C#

Shield TINKERKIT V2
sur Netduino plus 2

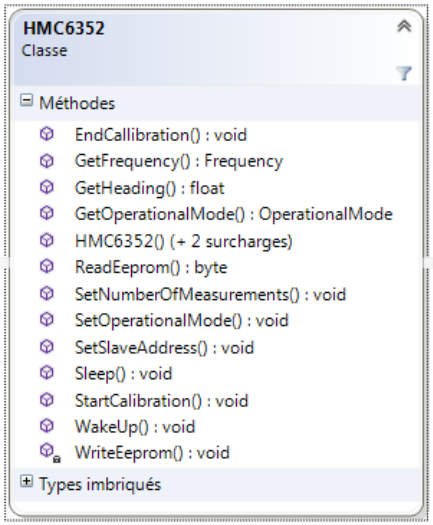
Figure 40 : Boussole HMC6352




```
using System;
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Sensors;

namespace TestNetduinoHMC6352
{
    public class Program
    {    // Pour accéder au bus I2C, relier la boussole HMC6352 au connecteur TWI de la
         // carte Tinkerkit. Placer des résistances de rappel (3,3k) entre le +5V et les sorties SCL et SDA

        public static void Main()
        {
            // Paramètres du bus I2C
            byte addCompass_I2C = 0x21; // Adresse (7 bits) du circuit HMC6352
            UInt16 Freq = 100;

            // Création d'un objet boussole HMC6352
            HMC6352 compass = new HMC6352 (addCompass_I2C, Freq);

            // Affichage de la version du software et de l'adresse de la boussole
            float lastHeading = (int)compass.GetHeading();
            byte ver = compass.ReadEeprom(HMC6352.EEPROMAddress.SoftwareVersion);
            Debug.Print("Software Version = " + ver.ToString());
            byte i2cAddr = compass.ReadEeprom(HMC6352.EEPROMAddress.SlaveAddress);
            Debug.Print("Slave Address = " + i2cAddr.ToString());

            HMC6352.OperationalMode mode = compass.GetOperationalMode();
            Debug.Print("Operational Mode = " + mode.ToString());

            HMC6352.Frequency frequency = compass.GetFrequency();
            Debug.Print("Frequency = " + frequency.ToString());

            Thread.Sleep(2000);
```

Suite du code

Matériels
- Carte Netduino
- Sensor **Shield TINKERKIT V2** + Hub I2C
- Boussole HMC6352



Page Web de la classe HMC6352

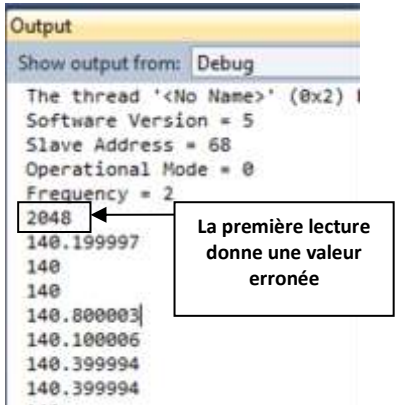
MicroToolsKit

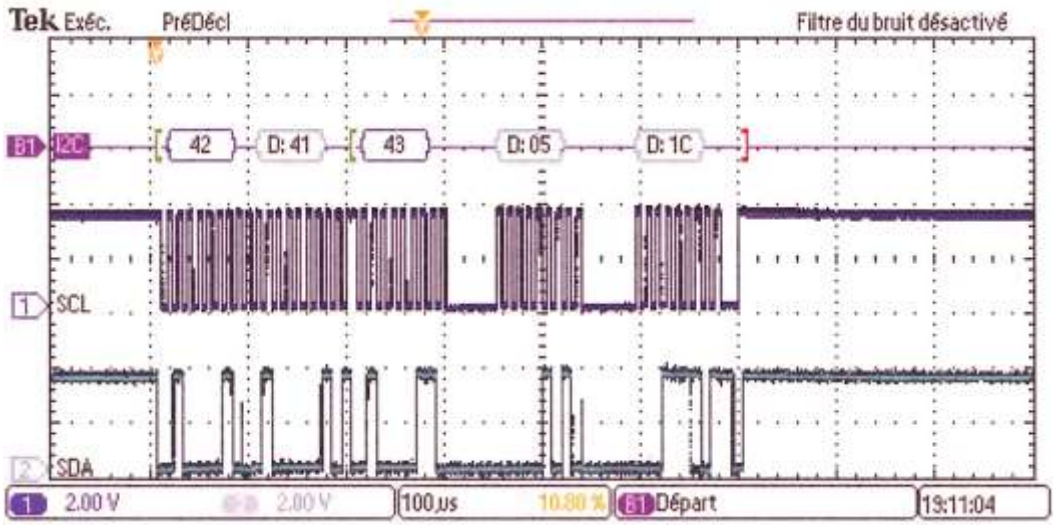
I2C_5 : Lire la direction donnée par une boussole HMC6352 (suite)

```
            while (true)
            {
                Thread.Sleep(500);

                float heading = compass.GetHeading();

                if (heading != lastHeading)
                {
                    lastHeading = heading;
                    Debug.Print(heading.ToString());
                }
            }
        }
    }
}
```

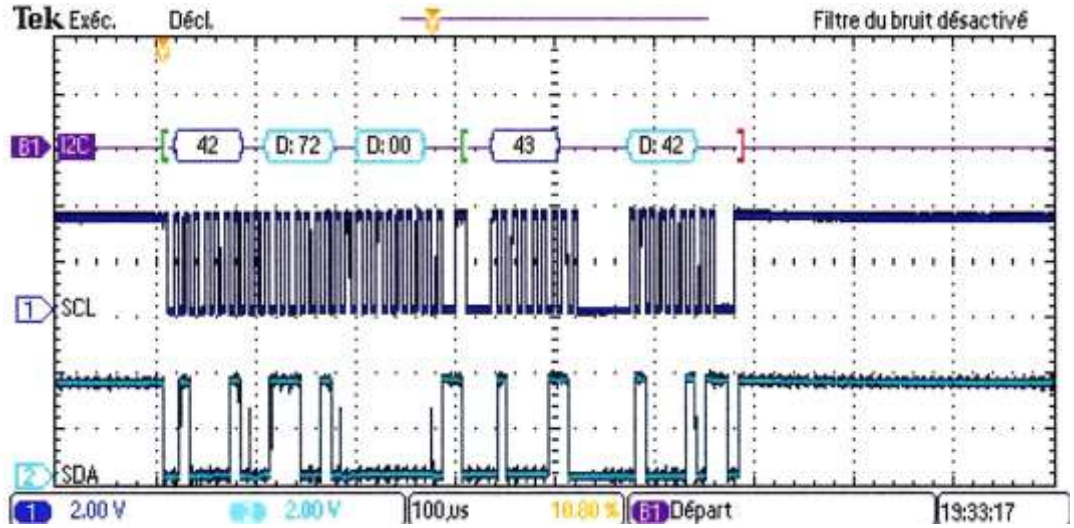
Utilisation du **debugger** pour visualiser la communication avec la boussole.

Pas à pas
F10 : Step Over
F11 : Step into





Le bit de lecture/écriture est inclus dans la valeur affiché =>Adressage : **[42h = 2*SLA+$\overline{W}$] [43h = 2*SLA+R]**
D'où **SLA = 21h**. Le µC demande à lire l'angle mesuré par la boussole (commande **D : 41h = 'A')**
La boussole répond **$051C_{(16)}$ = $1308_{(10)}$ = 130,8°**



Le µC demande l'adresse de la boussole (commande **D :72h = 'r' 00h = 0)**
La boussole répond **42h : 2SLA+ $\overline{W}$. Son adresse SLA est donc 21h (correspond à la valeur ci-dessus !)**

9.2.1.3.6 I2C_6 : Mesurer la température ambiante avec un capteur TMP102

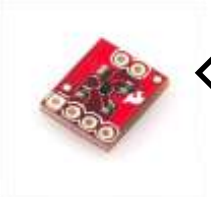
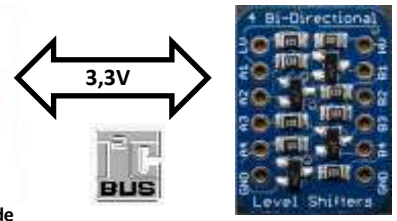


Figure 41 : Capteur de température TMP102



Convertisseur 5V <-> 3.3V (Adafruit BSS138)

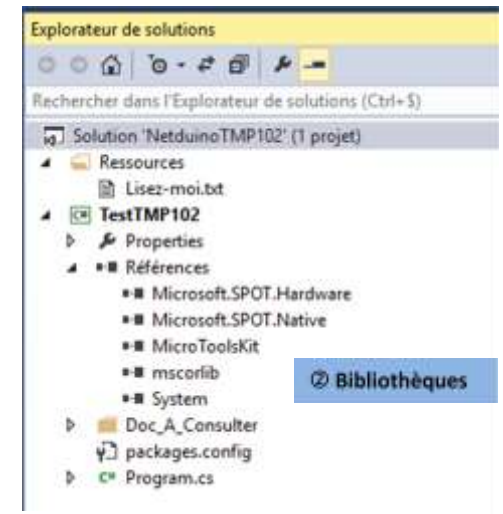
Shield TINKERKIT V2 sur Netduino plus 2

Code C#

```
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Sensors;

namespace TestNetduinoTMP102
{
    public class Program
    {
        public static void Main()
        {
            TMP102 CptTemp = new TMP102();
            CptTemp.Init();

            while (true)
            {
                float temperature = CptTemp.ReadAsCelcius();
                Debug.Print("Temperature:" + temperature.ToString("F1"));

                Thread.Sleep(1000);
            }
        }
    }
}
```

Alimentation: 1,4 à 3,6 Vcc
Consommation: 10 µA maxi (1 µA en veille)
Plage de mesure: -40°C à +125°C
Précision: 0,5 °C (de -25°C à +85°C)
Résolution: 0,0625°C
Interface série 2 fils
Dimensions: 13 x 10 mm

Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2 + Hub I2C
- Capteur de température TMP102



Page Web de la classe TMP102 + Code de l'exemple

MicroToolsKit

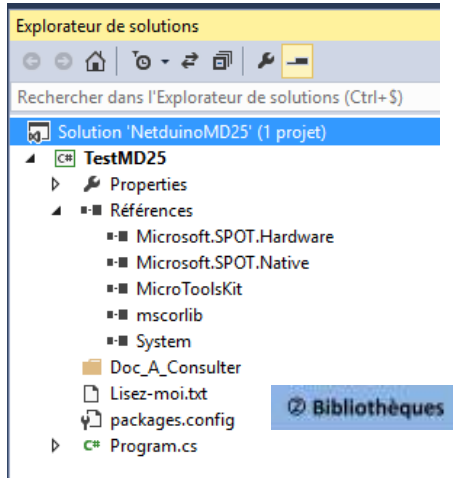
9.2.1.3.7 I2C_7 : Commander deux motoréducteurs, équipés d'encodeurs, avec une carte Devantech MD25

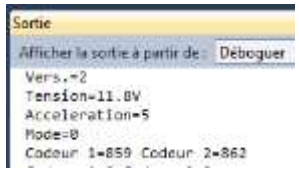
Code C#

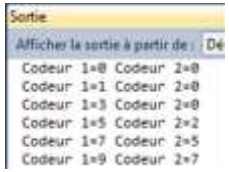
```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microtoolskit.Hardware.MotorDrivers;

namespace TestNetduinoMD25
{
    public class Program
    {
        public static void Main()
        {
            // Programme de test de la carte MD25
            // Création d'un objet MotorControl (carte MD25)
            // avec l'adresse 0x58 et la fréquence de bus F = 100kHz
            MD2x CarteMD25 = new MD2x();
            // Pour info : Lecture des registres de la carte MD2x et affichage de la version du logiciel
            Debug.Print("Vers.=" + CarteMD25.SoftRev.ToString());
            Debug.Print("Tension=" + ((Single)CarteMD25.Battery / 10).ToString("N1") + "V");
            Debug.Print("Acceleration=" + CarteMD25.AccelerationRate.ToString());
            Debug.Print("Mode=" + CarteMD25.Mode.ToString());

            while (true)
            {
                // Essai : Rotation des moteurs jusqu'à ce que la distance recherchée soit atteinte
                // ------------------------------------------------------------------------------
                CarteMD25.RazEncoders(); // Remise à zéro des codeurs
                CarteMD25.SetSpeedTurn(140, 140); // Réglage de la vitesse des moteurs

                while (CarteMD25.Encoder1 < 2000)
                {
                    Debug.Print("Codeur 1=" + CarteMD25.Encoder1.ToString() + " " +
                    "Codeur 2=" + CarteMD25.Encoder2.ToString());
                }
                CarteMD25.StopMotor(); // Arrêt des moteurs
                Thread.Sleep(5000);
            }
        }
    }
}
```

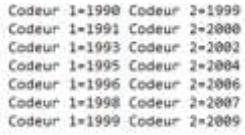
Shield TINKERKIT V2
sur Netduino plus 2

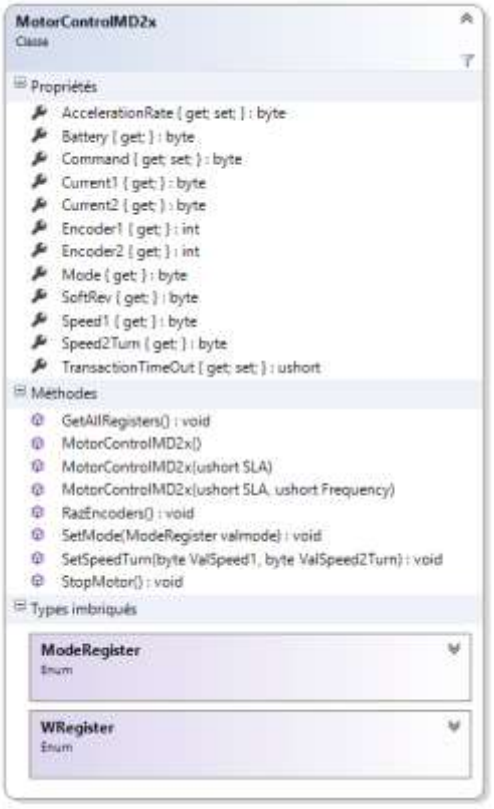
Figure 42 : Ensemble de pilotage pour moteurs "DCM2"











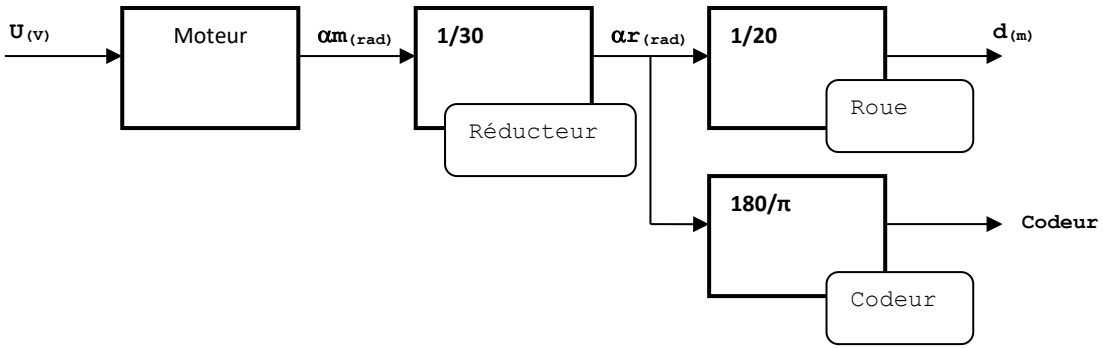
Matériels

- Carte Netduino
- Sensor **Shield TINKERKIT V2** + Hub I2C
- Ensemble de pilotage pour moteurs "DCM2"

Page Web de la classe MD25

MicroToolsKit

I2C_7 : Commander deux motoréducteurs, équipés d'encodeurs, avec une carte Devantech MD25 (suite)



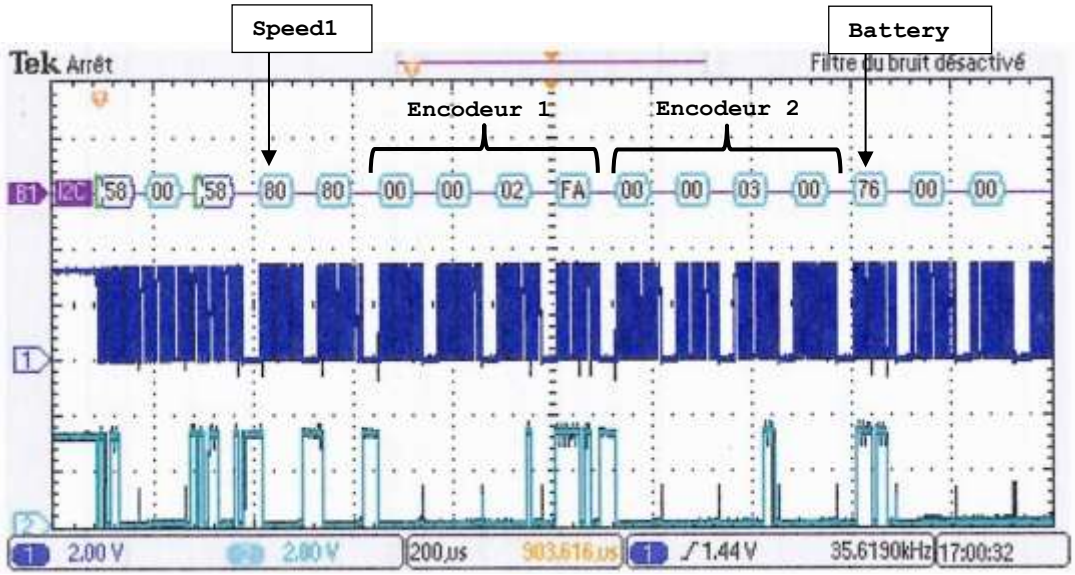
Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	R/W	Used for reset of encoder counts and module address changes

Figure 43 : Registres de la carte MD25

Lecture des registres de la carte MD25(partielle)

Ces chronogrammes ont été analysés à partir des informations données dans la documentation de la carte MD25. (Tableau ci-dessus)

Start: [
SLA: 58h
N du premier registre à lire: 00h
Repeated Start : [
@SLA : 58h
Speed1 : 80h
Speed2 : 80h
Enc1a : 00h
Enc1b : 00h
Enc1c : 02h
Enc1d : FAh
Etc..
Battery:76h
Etc..
Stop



Analyse de la trame avec la documentation de la carte MD25

Vitesse moteur 1 = Vitesse moteur 2 = 80h
D'après la documentation, en mode 0, cette valeur place les moteurs à l'arrêt.

Encodeur 1 = 02FAh
L'encodeur 1 a compté 762 impulsions codeur

Encodeur 2 = 30076h
L'encodeur 2 a compté 196726 impulsions codeur

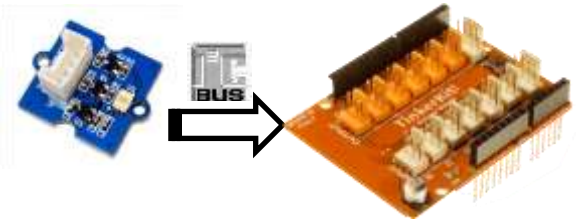
Batterie = 76h
La tension de la batterie est égal à 11,8V

Le bit de lecture écriture n'est pas inclus dans la valeur de l'octet d'adressage : SLA=58h. (h : valeur en hexadécimal**)**

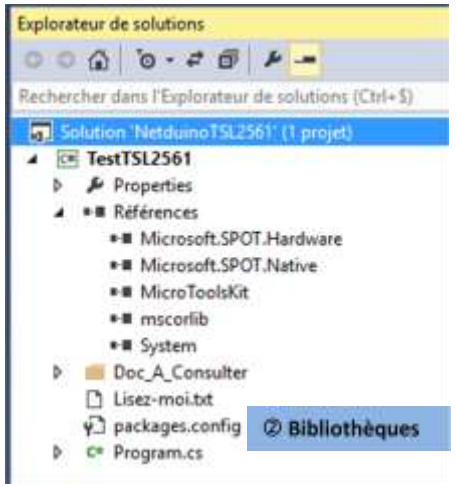
9.2.1.3.8 I2C_8 : Mesurer la luminosité ambiante avec un capteur TSL2561

Code C#

```
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Sensors;

namespace TestNetduinoTSL2561
{
    public class Program
    {
   // http://webge.github.io/TSL2561/
        public static void Main()
        {
         // Création d'un objet TSL2561
            // avec l'adresse SLA=0x29 et la fréquence de bus F = 100kHz
            TSL2561 I2CLightSensor = new TSL2561();
            I2CLightSensor.Init(TSL2561.Gain.x1, TSL2561.IntegrationTime._13MS);

            // Réglage des seuils
            I2CLightSensor.ThreshLowHigh = 0x20; I2CLightSensor.ThreshLowLow = 0x10;
            I2CLightSensor.ThreshHighHigh = 0x40; I2CLightSensor.ThreshHighLow = 0x20;

            while (true)
            { // Affichage du contenu des registres
                Debug.Print("Lecture des registres");
                Debug.Print("----------------------");
                Debug.Print("00h Control              : " + I2CLightSensor.Control);
                Debug.Print("01h Timing               : " + I2CLightSensor.Timing);
                Debug.Print("02h ThreshLowLow         : " + I2CLightSensor.ThreshLowLow);
                Debug.Print("03h ThreshLowhigh        : " + I2CLightSensor.ThreshLowHigh);
                Debug.Print("04h ThreshhighLow        : " + I2CLightSensor.ThreshHighLow);
                Debug.Print("05h Threshhighhigh       : " + I2CLightSensor.ThreshHighHigh);
                Debug.Print("06h Interrupt            : " + I2CLightSensor.Interrupt);
                Debug.Print("0Ah Part number / Rev Id : " + I2CLightSensor.Id);
                Debug.Print("--------------------------");
                Debug.Print("Valeurs des canaux 0 et 1");
                Debug.Print("--------------------------");
                Debug.Print("Canal 0: " + I2CLightSensor.Channel0); Debug.Print("Canal 1: " + I2CLightSensor.Channel1);
                if (I2CLightSensor.CalculateLux()!=0; Debug.Print("Luminosité: " + I2CLightSensor.CalculateLux()+"lux");
                } Else { Debug.Print("Sensor overload");}
                Thread.Sleep(1000);
            }
        }
    }
}
```



Shield TINKERKIT V2
sur Netduino plus 2



nuget
MicroToolsKit

```
Lecture des registres
----------------------
00h Control              : 3
01h Timing               : 0
02h ThreshLowLow         : 16
03h ThreshLowhigh        : 32
04h ThreshhighLow        : 32
05h Threshhighhigh       : 64
06h Interrupt            : 0
0Ah Part number / Rev Id : 17
--------------------------
Valeurs des cannaux 0 et 1
--------------------------
Canal 0: 185
Canal 1: 90
Luminosité: 729lux
```

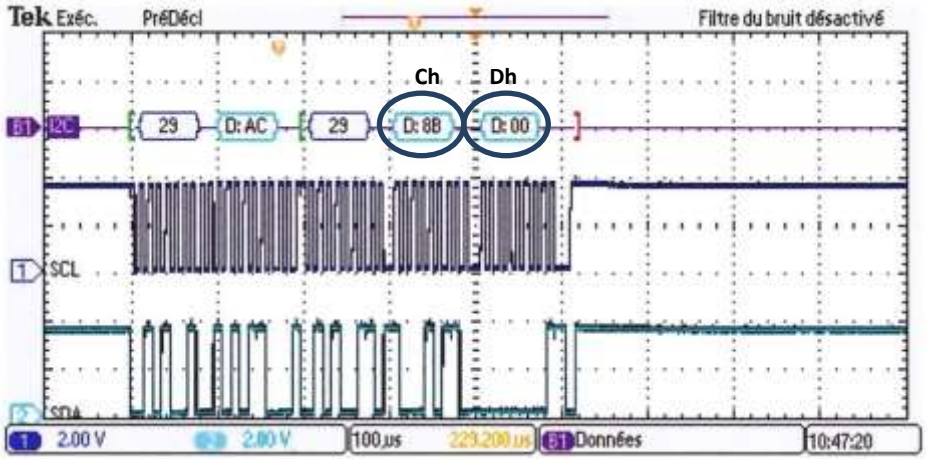
Matériels
- Carte Netduino
- Sensor **Shield TINKERKIT V2** + Hub I2C
- Capteur de luminosité TSL2561 **(Grove SEN-10171P)**

github SOCIAL CODING
Page Web de la classe TSL2561

I2C_8 : Mesurer la luminosité ambiante avec un capteur TSL2561 (suite)

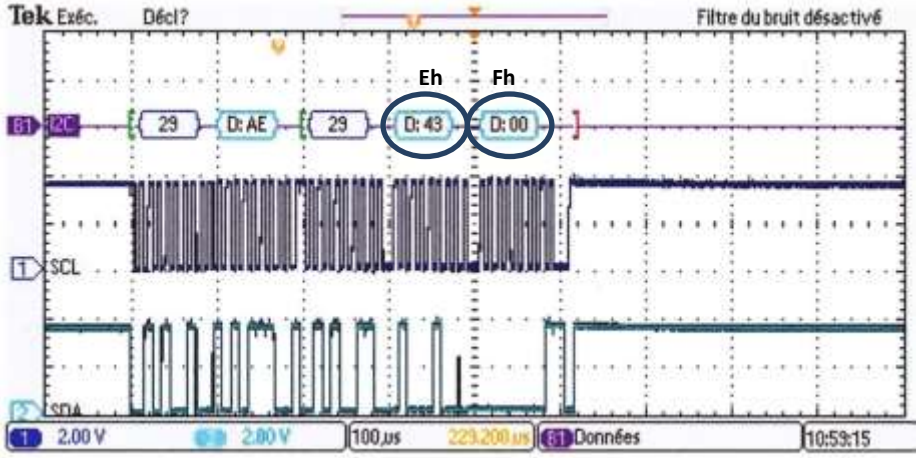
Lecture du canal 0 en mode mot : registres Ch(Low byte) **et Dh(**High byte)

Canal 0 = 008Bh



Lecture du canal 1 en mode mot : registres Eh(Low byte) **et Fh(**High byte)

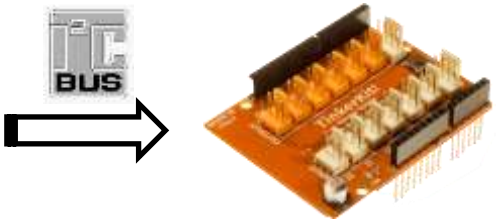
Canal 1 = 0043h



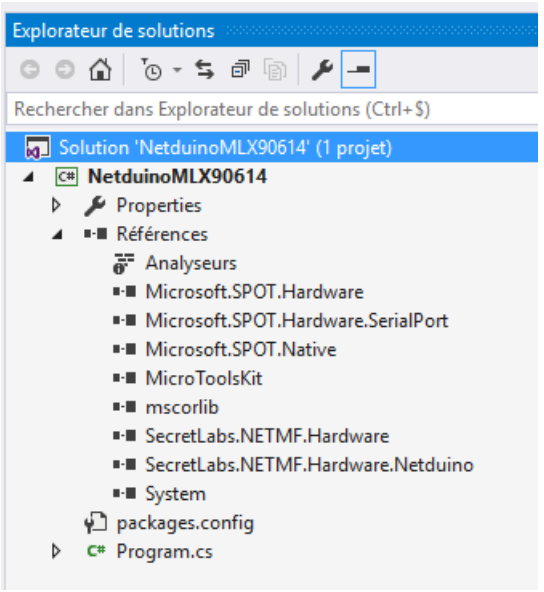
La méthode **calculateLux()**
renvoie 420 lux pour ces valeurs de canaux.

9.2.1.3.9 I2C_9 : Mesurer la température d'un objet avec un capteur MLX90614

MLX90614



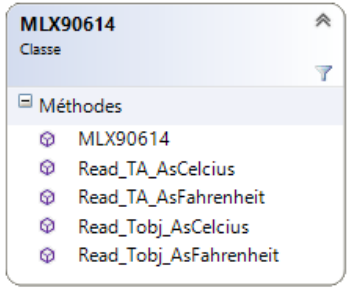
Shield TINKERKIT V2
sur Netduino plus 2



MicroToolsKit

Code C#

```
using System.Threading;
using Microsoft.SPOT;

using Microtoolskit.Hardware.Sensors;

namespace NetduinoMLX90614
{
    public class Program
    {
        public static void Main()
        {
            MLX90614 tempIR = new MLX90614();

            while (true)
            {
                Debug.Print("Température Air = " + tempIR.Read_TA_AsCelcius().ToString("F1") + " °C");
                Debug.Print("Température IR = " + tempIR.Read_Tobj_AsCelcius().ToString("F1") + " °C");
                Debug.Print("----------------------");
                Thread.Sleep(500);
            }
        }
    }
}
```



Page Web de la classe MLX90614

Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2 + Hub I2C
- Capteur de température MLX90614

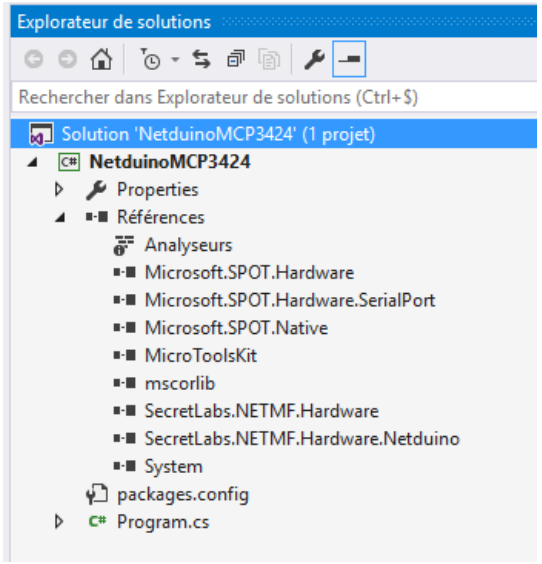
9.2.1.3.10 I2C_10 : Acquérir des données issues de capteurs analogique avec un MCP3424

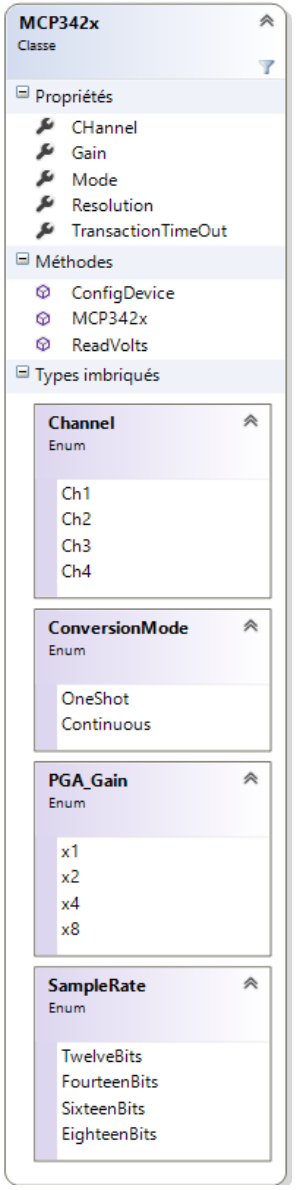
Code C#

```
using Microsoft.SPOT;
using Microtoolskit.Hardware.IO;
using System.Threading;

namespace NetduinoMCP3424
{
    public class Program
    {
        public static void Main()
        {
            MCP342x can = new MCP342x();
            can.CHannel = MCP342x.Channel.Ch2;
            double canal = System.Math.Pow(2, (double)can.CHannel);

            can.Resolution = MCP342x.SampleRate.FourteenBits;
            int resolution = 12 + 2 * (byte)can.Resolution;

            can.Gain = MCP342x.PGA_Gain.x4;
            double gain = System.Math.Pow(2, (byte)can.Gain);

            while (true)
            {
                Debug.Print("Continuous on channel " + canal + " " + "Gain: " + gain + " " + "Resolution: " + resolution);
                Debug.Print("Tension = " + can.ReadVolts().ToString("F2"));
                Thread.Sleep(1000);
            }
        }
    }
}
```

Page Web de la classe MCP342x

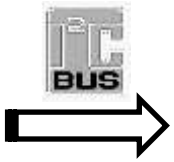
MicroToolsKit





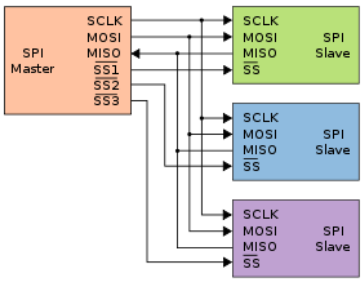
Matériels
- Carte Netduino
- Sensor Shield TINKERKIT V2 + Hub I2C
- Convertisseur analogique numérique MCP342x

MCP3424



Shield TINKERKIT V2
sur Netduino plus 2

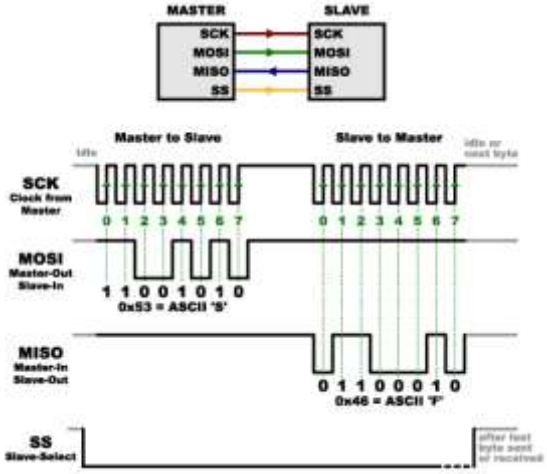
9.2.2 SPI



9.2.2.1 Généralités

*"Une liaison **SPI** (pour **Serial Peripheral Interface**) est un bus de données série synchrone baptisé ainsi par Motorola, qui opère en mode Full-duplex.*

Les circuits communiquent selon un schéma maître-esclaves, où le maître s'occupe totalement de la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée chip select. » Wikipédia



Le bus SPI utilise quatre signaux logiques :
SCLK — Serial Clock, Horloge (généré par le maître)
MOSI — Master Output, Slave Input (généré par le maître)
MISO — Master Input, Slave Output (généré par l'esclave)
SS — Slave Select, Actif à l'état bas (généré par le maître)

Une transmission SPI typique est une communication simultanée entre un maître et un esclave :
Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer par l'utilisation du signal SS
L'esclave répond aux requêtes du maître
À chaque coup d'horloge le maître et l'esclave s'échangent un bit. Après huit coups d'horloges le maître a transmis un octet à l'esclave et vice versa. La vitesse de l'horloge est réglée selon des caractéristiques propres aux périphériques.

Avantages
Communication Full duplex. Débit assez important par rapport à I2C
Flexibilité du nombre de bits à transmettre ainsi que du protocole en lui-même
Simplicité de l'interface matérielle
Aucun arbitre nécessaire car aucune collision possible
Les esclaves utilisent l'horloge du maître et n'ont donc pas besoin d'oscillateur propre
Partage d'un bus commun pour l'horloge, MISO et MOSI entre les périphériques

Inconvénients
Monopolise plus de broches d'un boîtier que l'I2C ou une UART qui en utilisent seulement deux.
Aucun adressage possible, il faut une ligne de sélection par esclave en mode non chaîné.
Le protocole n'a pas d'acquittement. Le maître peut parler dans le vide sans le savoir.
Ne s'utilise que sur de courtes distances contrairement aux liaisons RS-232, RS-485 ou bus CAN. Néanmoins, il est possible d'utiliser des tampons de bus, comme des adaptateurs RS-232 (exemple : MAX208), RS485 ou LVDS pour relier un maître et un esclave avec un câble de quelques mètres. La mise en œuvre de ces circuits reste délicate car il devient nécessaire de procéder à des adaptations d'impédance pour éviter les oscillations parasites.

9.2.2.2 Le bus SPI des cartes Netduino

Développement dans la prochaine révision du document

9.2.2.3 Précautions d'utilisation

Développement dans la prochaine révision du document

9.2.2.4 Objet logiciel « AnalogInput»

Développement dans la prochaine révision du document

9.2.2.5 Tableau récapitulatif des exemples de code

Développement dans la prochaine révision du document

9.2.2.5.1 SPI_1 : Commande d'un convertisseur analogique numérique 16 bits (ADC124S101)

Développement dans la prochaine révision du document

9.2.2.5.2 SPI_2 : Afficheur graphique tactile FTDI FT800

Développement et exemple dans la prochaine révision du document

Code C#

Explications détaillées de l'exemple

Développement et exemple dans la prochaine révision du document

9.2.3 Protocoles 1 fil (One Wire)

9.2.3.1 Généralités

Développement dans la prochaine révision du document

9.2.3.2 Le bus One Wire des cartes Netduino

Développement dans la prochaine révision du document

9.2.3.3 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
One_Wire_1	NetduinoDS18B20	One_Wire : Mesurer la température ambiante avec un capteur **DS18B20**				

Signalétique

Lien hypertexte vers l'exemple décrit dans le document.

Nom du répertoire contenant le projet Visual Studio.

Si la colonne est cochée, un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au CI ou au module (maintenue sur **Github)**.

La classe spécifique au CI ou au module est inclue dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous répertoire Doc_A_Consulter du répertoire du projet.

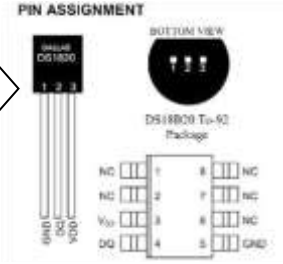
Pour plus d'informations, consulter la description de la classe **One wire** sur MSDN

9.2.3.3.1 One Wire_1 : Mesurer la température ambiante avec un capteur DS18B20

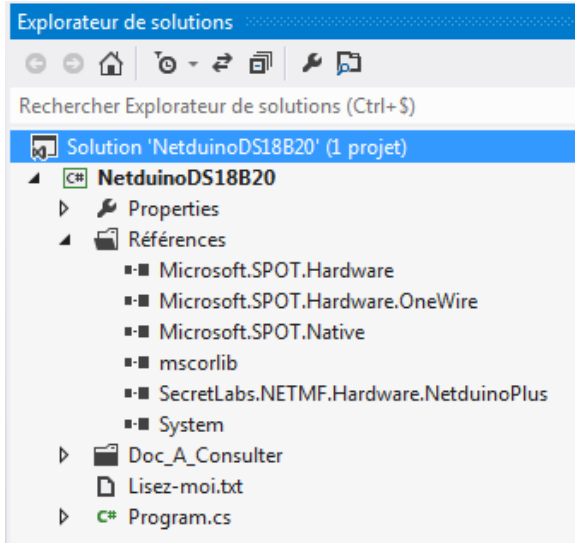
Code C#

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;
namespace TestNetduinoDS18B20
{
public class Program
    {
        public static void Main()
        {
            // DS18B20 connecté sur O5 du shield Tinkerkit
            var Pin = new OutputPort(Pins.GPIO_PIN_D3,false);

            var DS18B20 = new OneWire(Pin);
            UInt16 temperature;

            // read every second
            while (true)
            {
                if (DS18B20.TouchReset()!=0) // Détection du circuit
                {
                    DS18B20.WriteByte(0xCC);     // Skip ROM, si un seul circuit
                    DS18B20.WriteByte(0x44);     // Début de conversion

                    while (DS18B20.ReadByte() == 0) ;   // Attente fin de conversion

                    DS18B20.TouchReset();
                    DS18B20.WriteByte(0xCC);     // skip ROM
                    DS18B20.WriteByte(0xBE);     // Read Scratchpad

                    temperature = (byte)DS18B20.ReadByte();           // LSB
                    temperature |= (UInt16)(DS18B20.ReadByte() << 8); // MSB

                    Debug.Print("Temperature: " + temperature / 16);
                    Thread.Sleep(1000);
                }
                else
                {
                    Debug.Print("Le circuit n'a pas été détecté.");
                }

                Thread.Sleep(1000);
            }
        }
    }
}
```

Shield TINKERKIT V2



One Wire



Classe OneWire

```
public OneWire(Cpu.Pin pin);

public static byte CalculateCRC(byte[] buffer, int offset, int count);
public static ushort CalculateCRC16(byte[] buffer, int offset, int count, ushort seed);
public void Dispose();
public void Read(byte[] buffer, int offset, int count);
public byte ReadBit();
public byte ReadByte();
public bool Reset();
public bool Search_GetNextDevice(byte[] romRegistrationNumber);
public bool Search_IsDevicePresent(byte[] romRegistrationNumber);
public void Search_Restart();
public void Write(byte[] buffer, int offset, int count);
public void WriteBit(byte value);
public void WriteByte(byte value);
```

Matériels

- Carte Netduino
- Sensor Shield TINKERKIT V2 + Hub I2C
- Capteur de température DS18B20

10 La gestion du temps

10.1 HTR

Développement dans la prochaine révision du document

10.2 Timer

10.2.1 Généralités

Développement dans la prochaine révision du document

10.2.2 Les timers des cartes Netduino

Développement dans la prochaine révision du document

10.2.3 Tableau récapitulatif des exemples de code

	Visual Studio	Description (CI ou module)				
Timer 1	TimerExample	Principe				
Timer 2	NetduinoAnemometre	Application : Mesurer la vitesse du vent avec un kit Weather Sensor Assembly p/n 80422				

Signalétique

Lien hypertexte vers le code de l'exemple.

Nom du répertoire contenant le projet Visual Studio.

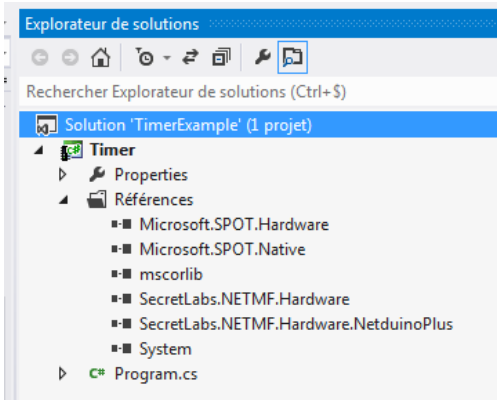
Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.

Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.

La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**

La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

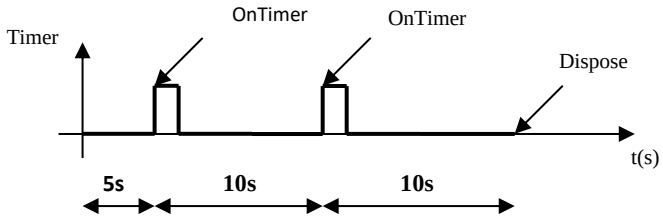
Pour plus d'informations, consulter la description de la classe **Timer** sur MSDN



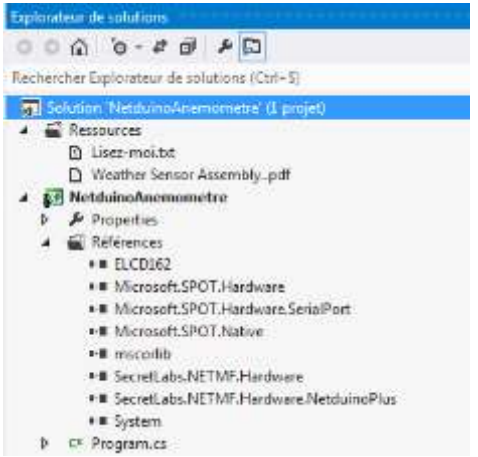
10.2.3.1 Timer 1 - Principe

Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TimerExample
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Kicking off the timer");
            Debug.Print(DateTime.Now.ToString());

            // Appel de la méthode OnTimer après une attente de 5 seconds et répétition toutes les 10 seconds
            Timer timer = new Timer(new TimerCallback(OnTimer), null, 5000, 10000);

            Debug.Print("Waiting for the timer to expire");
            Debug.Print(DateTime.Now.ToString());
            Thread.Sleep(25000);

            Debug.Print("Killing the timer");
            timer.Dispose();
            Thread.Sleep(Timeout.Infinite);
        }



        public static void OnTimer(object state)
        {
            Debug.Print("Timout Triggered");
            Debug.Print(DateTime.Now.ToString());
        }
    }
}
```



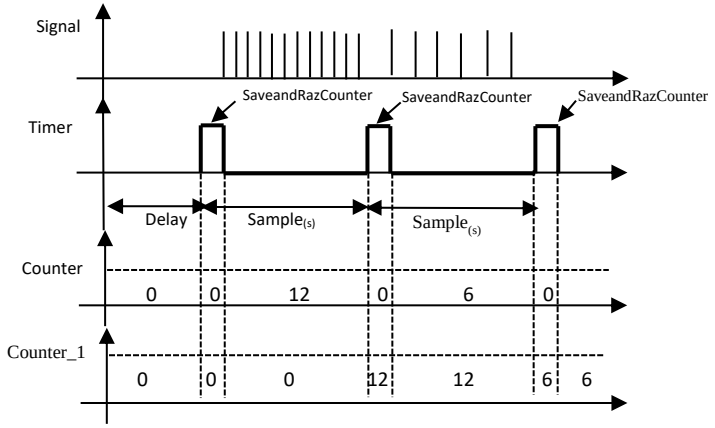
10.2.3.2 Timer 2 – Application : Mesurer la vitesse du vent avec un kit Weather Sensor Assembly p/n 80422



Code C#

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

using ToolBoxes; // Contient la bibliothèque ELCD162


namespace NetduinoAnemometre
{ // Titre : Mesure de la vitesse du vent avec un kit Weather Sensor Assembly p/n 80422 https://goo.gl/Pdbzt9
    public class Program
    {
        // Compteurs: mémorisent le nombre de tours de l'anémomètre pendant un temps t = sample
        static UInt32 Counter = 0, Counter_1 = 0;

        public static void Main()
        {   // variables
            UInt32 RazCount = 0;
            var delay = 0;    // Attente avant échantillonnage de la vitesse du vent
            var sample = 2;   // Période d'échantillonnage en seconde, à régler en fonction
            // de l'erreur acceptée sur la mesure
            // Exemple:  sample = 20*Tsignal pour une erreur de 5%

            // LCD à connecter sur la sortie O5 de la carte Tinkerkit
            // Documentation de la classe ELCD162 : https://github.com/WebGE?tab=repositories
            // Création et initialisation d'un objet afficheur série ELCD-162 (par défaut: COM2, 19200, None, 8, 1)
            var Lcd = new ELCD162(); Lcd.Init(); Lcd.ClearScreen();

            // Acquisition du signal TOR de l'anémomètre sur l'entrée I0 de la carte Thinkerkit
            var signal = new InterruptPort(Pins.GPIO_PIN_A0, true, Port.ResistorMode.Disabled, Port.InterruptMode.InterruptEdgeHigh);

            // Inscription de l'interruption
            signal.OnInterrupt += new NativeEventHandler(measureFreq);

            // Timer utilisé pour la remise à zéro du compteur après chaque échantillonnage de la vitesse du vent
            signal.DisableInterrupt();
            var timer = new Timer(new TimerCallback(SaveandRazCounter), RazCount, delay, (int)(1000 * sample));
            Counter = 0; signal.EnableInterrupt();
```

```
            while (true)
            {// la mesure de la fréquence de rotation de l'anémomètre est désactivée pendant l'affichage
                signal.DisableInterrupt(); // Désactivation de l'interruption pour ne pas perturber l'affichage
                var speed = 2.4 * (Counter_1 / sample);
                Lcd.ClearScreen(); Lcd.PutString("Vvent=" + speed.ToString("F1") + "km/h"); // Affichage de la fréquence
                Counter = 0;
                signal.EnableInterrupt(); // Autorisation de l'interruption
                Thread.Sleep(5000 * sample);
            }
        }

        // ------------------------------------------------------------
        // Gestionnaires d'interruption
        // ------------------------------------------------------------
        // Raz compteur
         static void SaveandRazCounter(object RazCount)
                {
                    Counter_1 = Counter;
                    Counter = (UInt32)RazCount;
                }
         // ------------------------------------------------------------
        // Compteur
        static void measureFreq(uint port, uint state, DateTime time)
        {
            Counter++;
        }
        // ------------------------------------------------------------
    }
}
```



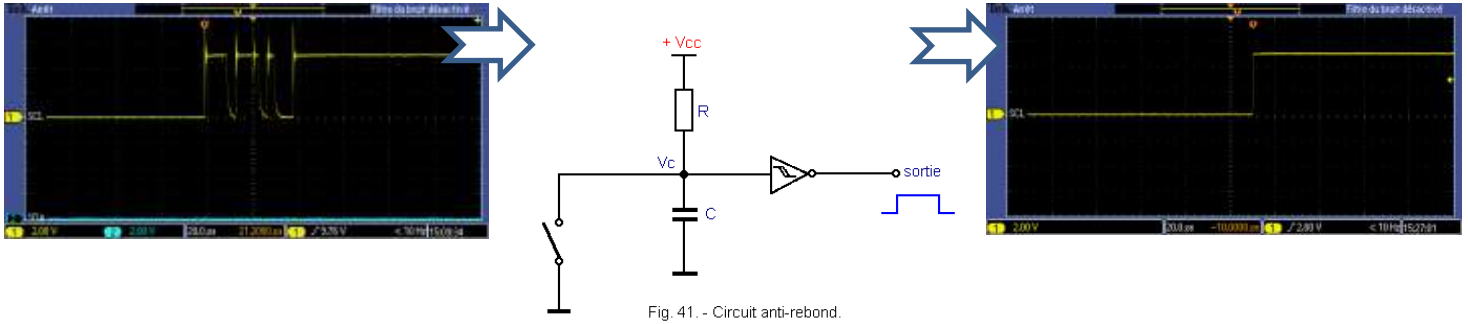


Fig. 41. - Circuit anti-rebond.

11 Le système de fichiers

Développement et exemple dans la prochaine révision du document

11.1 Accès à une carte SD

Expandable memory is always a plus no matter whether it's a Phone, Camera or a Microcontroller. Even advantageous if we know how to use it. In this fifth day Netduino tutorial, we will learn a few writing operations about an SD card. We will learn how to write to a text file, shown as an example of writing a log.
The ***Logger* class** is also capable of creating a text file at any given location then writing some text information to it.

SD Card in Netduino Plus
Circuit Setup and Theory
Well, pop in the SD card to the slot and you are good to go! No messy wires…
When an SD card is inserted into the slot, Netduino Plus will automatically mount it as SD directory. So the root direct is SD. The logger file, by default creates a log at SD\Report\logger.txt location. If you need to write to a custom location then use the *LogCustom* method.
After logged, typically at the end of your program, you **MUST** call the *Close* method otherwise information will not be saved unless you call *Flush* method from the *Logger* class.

Développement et exemple dans la prochaine révision du document

Annexes

A1 – Tableaux récapitulatifs des exemples de code

Signalétique


Lien hypertexte vers le code de l'exemple.


Nom du répertoire contenant le projet Visual Studio.


Un fichier "*Lisez-moi.txt*" décrivant le projet est disponible dans le répertoire Visual Studio.


Lien hypertexte vers la page web décrivant **la classe spécifique** au circuit intégré ou au module (maintenue sur **Github)**.


La classe spécifique au circuit intégré ou au module est incluse dans la bibliothèque **MicroToolsKit** (un NuGet **sur NuGet.org).**


La photo du montage à réaliser est dans le sous-répertoire Doc_A_Consulter du répertoire du projet.

Les entrées, sorties numériques

	Visual Studio	Description (CI ou module)				
E/S_1	BlinkingLed	- Sortie numérique : faire clignoter la LED de la carte Netduino.				
E/S_2a	LightSwitch	- E/S numériques : commander une LED avec un bouton-poussoir.				
E/S_2b	NetduinoClav4Digilent	- E/S numériques : commander une LED avec un clavier **Digilent 4BP**	☒			☒
E/S_3	Netduino_EasyStepperMot	- Sorties numériques : commander un moteur pas à pas avec une carte **EasyStepper Driver Motor V4.4**	☒	☒	☒	

	Visual Studio	Description (CI ou module)				
INT_1	LightSwitchINT	E numérique : commander une LED avec un bouton-poussoir.				
Timer_2	NetduinoAnemometre	Timer : Mesure de la vitesse du vent avec un kit wheather Sensor Assembly p/n80422				☒

	Visual Studio	Description (CI ou module)				
PWM_1	NetduinoPWM	PWM : Faire varier la luminosité d'une LED				
PWM_2	NetduinoArdumoto	PWM : Faire varier la vitesse d'un moteur à CC	☒			☒
PWM_3	NetduinoServo	PWM : Régler la position d'un servomoteur de modélisme	☒			
PWM_4	NetduinoEscBrushless	PWM : Régler la fréquence de rotation d'un moteur brushless				

Les entrées analogiques

	Visual Studio	Description (CI ou module)				
Analog_1	NetduinoPot	Régler la fréquence de clignotement d'une LED avec un potentiomètre.	☒			
Analog_2	MesureAngle	Mesurer une position angulaire avec un potentiomètre		☒	☒	☒
Analog_3	NetduinoThermoGHI	Mesurer la température ambiante avec un module GHI FEZ thermomètre.				
Analog_4	NetduinoGirouette	Mesurer la direction du vent avec un kit wheather Sensor Assembly p/n80422	☒	☒	☒	☒

La communication série – Asynchrone - UART

	Visual Studio	Description (CI ou module)				
UART_1	NetduinoUART	UART : Transmettre une valeur numérique via une liaison **RS232**	☒			☒
UART_2	NetduinoELCD_162	UART : Utiliser un afficheur LCD à commande série **ELCD-162**	☒	☒	☒	☒
UART_3a	NetduinoXBee_E	UART : Transmettre des données avec un module **XBee**	☒			☒
UART_3b	NetduinoXBee_R	UART : Recevoir des données avec un module **XBee**	☒	☒	☒	☒

La communication série – Synchrone – I²C

	Visual Studio	Description (CI ou module)				
I2C_1	NetduinoPCF8574	I²C : Chenillard sur huit LED reliées à un port d'E/S **PCF8574.**	☒	☒	☒	☒
I2C_2	NetduinoI2CLCD	I²C : Commander un afficheur LCD à circuit **PCF2119.**	☒	☒	☒	☒
I2C_3	NetduinoSRF08US	I²C : Mesurer une distance avec un télémètre à ultrasons **SRF08.**	☒	☒	☒	☒
I2C_4	NetduinoI2CLEDBP	I²C : Recopier l'état de boutons poussoirs sur des LED via des **PCF8574 (**carte SSI).	☒	☒	☒	☒
I2C_5	NetduinoHMC6352	I²C : Lire la direction donnée par une boussole **HMC6352.**	☒	☒	☒	☒
I2C_6	NetduinoTMP102	I²C : Mesurer la température ambiante avec un capteur **TMP102.**	☒	☒	☒	☒
I2C_7	NetduinoMD25	I²C : Commander deux motoréducteurs à C.C. équipés d'encodeurs avec une carte **MD25.**	☒	☒	☒	☒
I2C_8	NetduinoTSL2561	I²C : Mesurer la luminosité ambiante avec un capteur **TSL2561**	☒	☒	☒	☒
I2C 9	NetduinoMLX90614	I²C : Mesurer la température d'un objet avec un capteur **MLX90614.**		☒	☒	
I2C 10	NetduinoMCP3424	I²C : Acquérir des données issues de capteurs analogiques avec un CAN **MCP3424.**		☒	☒	

La communication série – Synchrone – SPI

	Visual Studio	Description (CI ou module)				
SPI1		Commande d'un convertisseur analogique numérique 16 bits **ADC124S101** (à venir)				
SPI2		Afficheur graphique tactile **FTDI FT800** (A venir)				

La communication série – Synchrone – One Wire

	Visual Studio	Description (CI ou module)				
One_Wire_1	NetduinoDS18B20	OneWire : Mesurer la température ambiante avec un capteur **DS18B20**				

La gestion du temps - HTR

	Visual Studio	Description (CI ou module)				

La gestion du temps - Timer

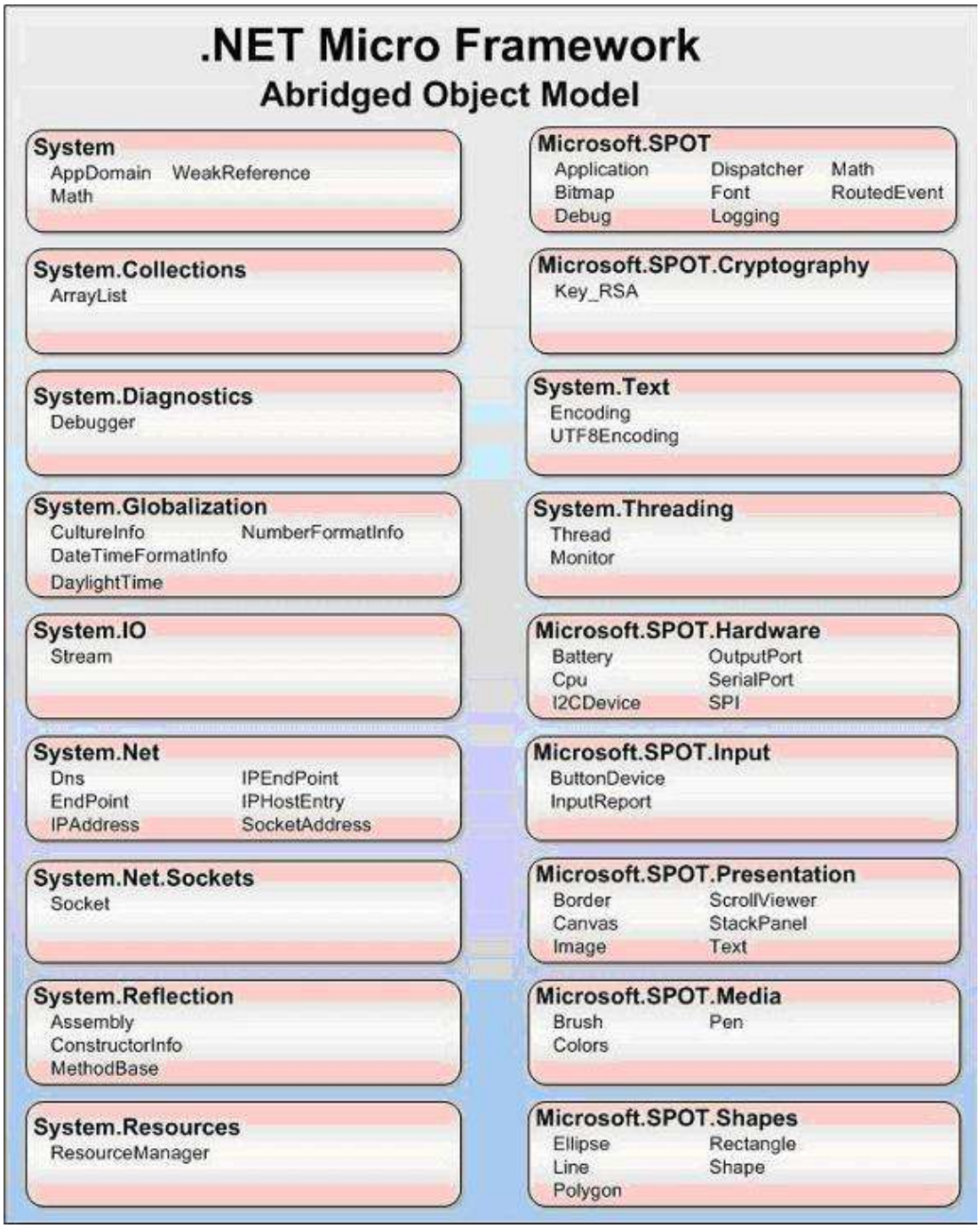
	Visual Studio	Description (CI ou module)				
Timer 1	TimerExample	Timer : Principe				
Timer 2	NetduinoAnemometre	Timer : Application : Mesurer la vitesse du vent avec un kit Weather Sensor Assembly p/n 80422				

Les systèmes de fichier

	Visual Studio	Description (CI ou module)				
SD1	NetduinoSD	Accès à un fichier sur une carte SD. (A venir)				

A2 - API Reference for .NET Micro Framework 4.3

https://goo.gl/dIvfxy



A3 – Les types reconnus par Microsoft Visual Studio et le .NET Microframework

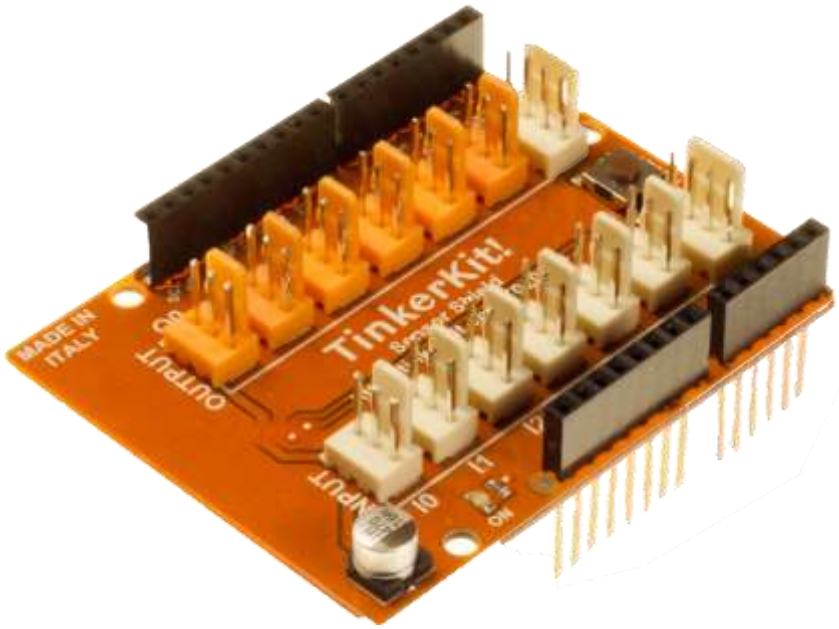
Short Name	.Net Struct/Class	Signed	Width (bytes)	Range	Exemple d'utilisation
bool	Boolean	-	1	Vrai (true) ou faux (false)	bool present = false; Boolean present = false;
byte	Byte	non	1	0 to 255	
sbyte	SByte	oui	1	-128 to 127	
int	Int32	oui	4	2^{31} to $2^{31} - 1$ -2147483648 to 2147483647	int valeur = -164; Int32 valeur = -164;
uint	UInt32	non	4	0 to $2^{32} - 1$ 0 to 4294967295	uint compte = 42; UInt32 compte = 42;
short	Int16	oui	2	-32768 to 32767	
ushort	UInt16	non	2	0 to 65535	
long	Int64	oui	8	2^{63} to 2^{63} - 1	long attente = 421; Int64 attente = 421;
ulong	UInt64	non	8	0 to 2^{64} - 1	
float	Single	oui	4	-3.402823e38 to 3.402823e38	float nombre = 0.45F; Single nombre = 0.45F;
double	Double	oui	8	-1.79769313486232e^{308} to 1.79769313486232e^{308}	double nombre = 0.45; Double nombre = 0.45;
décimal	Decimal	oui	12	±1.0 × 10e−28 to ±7.9 × 10e28 Precise fractional or integral type that can represent decimal numbers with 29 significant digits	
char	Char	-	2	0 à 65535	char lettre = 'A'; Char lettre = 'A';
String	**string**	-	2 par caractère		string couleur = "rouge"; String couleur = "rouge";

Extrait de la documentation Microsoft

A4 - Icônes utilisées dans Visual Studio

Icon	Description	Icon	Description
	Namespace		Method or Function
	Class		Operator
	Interface		Property
	Structure		Field or Variable
	Union		Event
	Enum		Constant
	TypeDef		Enum Item
	Module		Map Item
	Extension Method		External Declaration
	Delegate		Error
	Exception		Template
	Map		Unknown
	Type Forwarding		

A5 – Le Shield Tinkerkit



The **Sensor Shield v.2** allows you to hook up the TinkerKit **SENSORS** and **ACTUATORS** directly to the NetDuino, without the use of the breadboard.

It has 12 standard TinkeKit 3pin connectors. The 6 labeled **I0** through **I5** are **Analog Inputs**. The ones labeled **O0** through **05** are **Outputs** connected to the PWM capable outputs of the Netduino Board (it is possible to change these to Digital Inputs, in which case they will report either **HIGH** or **LOW**, but nothing in between).

- Pin **11** on the Netduino is **O0** on the shield.
- Pin **10** on the Netduino is **O1** on the shield.
- Pin **9** on the Netduino is **O2** on the shield.
- Pin **6** on the Netduino is **O3** on the shield.
- Pin **5** on the Netduino is **O4** on the shield.
- Pin **3** on the Netduino is **O5** on the shield.

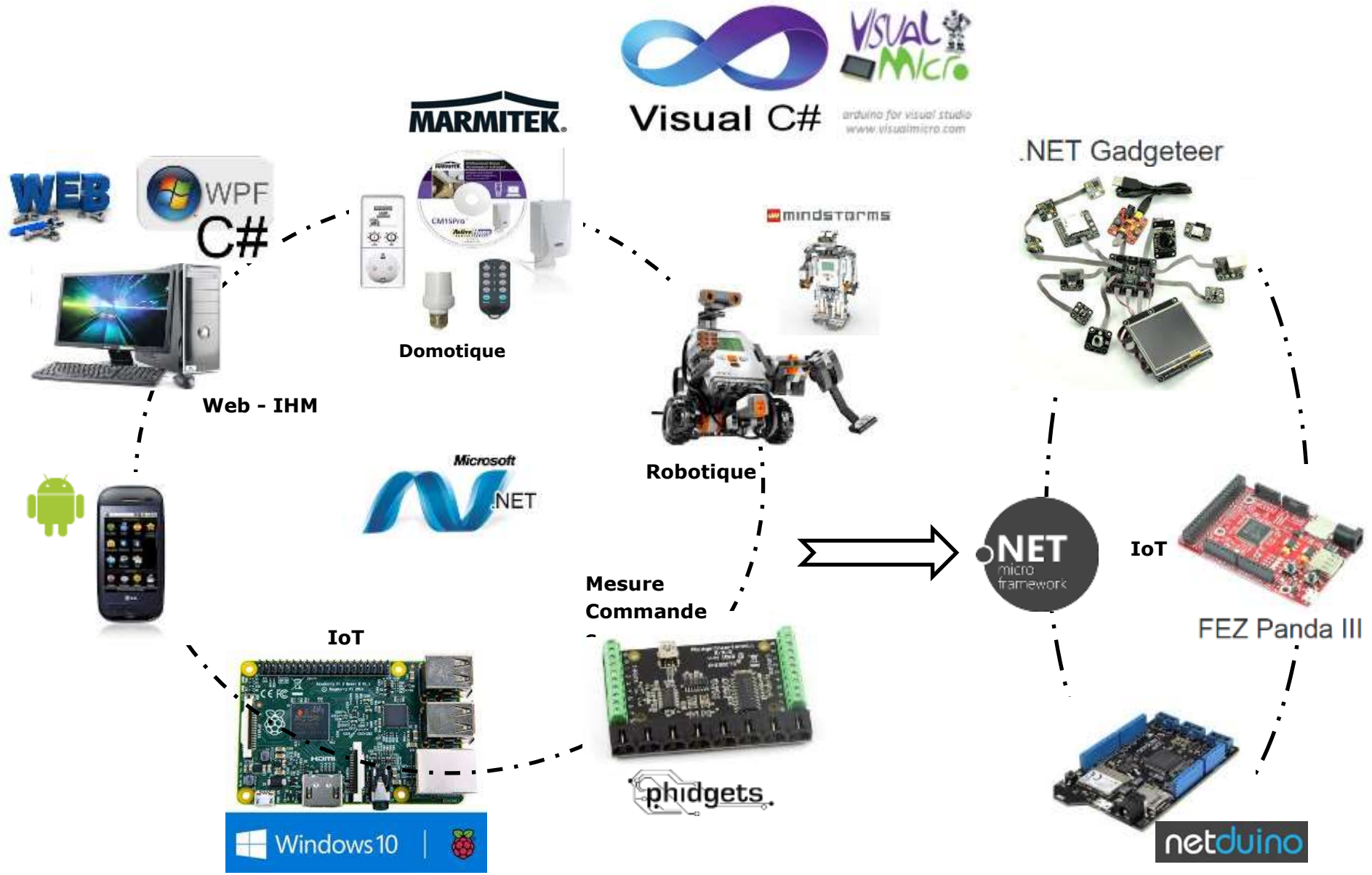
Module description: A green LED signals that the shield is correctly powered, a standard 6mm pushbutton allows you to RESET the board.

The **4pin TWI socket** allows communication to any device supporting the I2C protocol through the Wire library on Netduino. 5V and Ground are provided on the socket.

The **4pin SERIAL socket** allows the board to communicate with other devices that support serial communication. 5V and Ground are provided on the socket for your convenience.
Note: If you are sending or receiving data to and from the computer this serial connector is not available.

Two mounting holes are provided in the same position found on the Netduino board. A third hole allows you to see the led connected to pin 13 of the Netduino.

A6 – Quelques applications de la technologie .NET



Glossaire

Autocompletion (Intellisense)
L'**autocompletion** (**intellisence**) permet de réduire le taux d'erreur lors de l'écriture du code.

Bibliothèque (Assembly)
Les **bibliothèques** (*assembly*) sont des fichiers compilés (.dll) contenant des classes. Lorsqu'on utilise les fonctionnalités d'une classe dans un programme, elle doit être présente dans le répertoire "*Référence*" du projet. La documentation de la bibliothèque de classes du micro framework .Net est accessible à partir du site MSDN. [https://goo.gl/dIvfxy]
Coloration syntaxique,

Debugger in situ
Un **débogueur** (ou *débogueur*, de l'anglais debugger) est un logiciel qui aide un développeur à analyser les bugs d'un programme. Pour cela, il permet d'exécuter le programme pas-à-pas, d'afficher la valeur des variables à tout moment, de mettre en place des points d'arrêt sur des conditions ou sur des lignes du programme ...

Espaces de noms
Les **espaces de noms** résolvent le problème du nommage des classes. Deux classes peuvent avoir le même nom si elles se situent dans des espaces de noms différents. Ils permettent donc d'organiser les nombreuses classes du micro framework .Net. Leur déclaration derrière le mot clé ***using*** évite de réécrire le nom complet lors de leur utilisation.

Exemple : La déclaration `using SecretLabs.NETMF.Hardware.NetduinoPlus;`
permet d'écrire `OutputPort(Pins.ONBOARD_LED, false);`
sinon,il serait nécessaire d'écrire
`SecretLabs.NETMF.Hardware.NetduinoPlus.OutputPort(Pins.ONBOARD_LED, false);`

Multithreading
Un processeur est dit **multithread** s'il est capable d'exécuter efficacement plusieurs threads (processus légers) simultanément.

Template de code
En programmation informatique, les ***templates*** (en Français **modèles**, parfois aussi appelés **patrons**) autorisent l'écriture d'un code sans considération envers le type des données avec lesquelles il sera finalement utilisé.

Bibliographie

PDF téléchargeables

	**Visual Studio 2015**	http://it-ebooks.info/book/1472846854/
	**Expert .NET Micro Framework**	http://it-ebooks.info/book/2053/
	**Getting_Started_with_the_Internet_of_Things**	Make

Livres

	**Cours et Exercices en UML 2**	Hugues Bersini	EYROLLES
	**C#6 et Visual Studio 2015** **Les fondamentaux du langage**	Sébastien Putier	ENI

Numériques (Kindle Edition)

Netduino Measurement Electronics: hardware and software [Kindle Edition]	Amazon
Professional's Guide To .NET Micro Framework Application Development [Kindle Edition]	Amazon
Netduino Home Automation Projects [Kindle Edition]	Amazon

Webographie

Blogs

Comparatif	Arduino, Netduino, Raspberry Pi et Beaglebone Black	http://www.dragonflythingworks.com/
	Client et serveur HTTP, multithreading	http://www.gsiot.info/

Matériels, documentation, forums, projets

netduino	http://netduino.com/
CodePlex Project Hosting for Open Source Software	https://www.codeplex.com
github SOCIAL CODING	https://github.com/
CODE PROJECT For those who code	http://www.codeproject.com/

Ressources logicielles

Gestionnaire de paquets à installer dans Visual studio À partir d'outils -> Extensions et mises à jour.	nuget	
		FTDI Chip
WPF-like library for simple graphic-UI application using Netduino (Plus) 2 and the FTDI FT800 Eve board.	http://cetmicrowpf.codeplex.com/	 **VM800B43A-PL**
Netduino plus 2 GPS	http://n2plusgps.codeplex.com/	 Adafruit Ultimate GPS Logger Shield
Shared library for the Netduino hardware and components (MCP23017, DS1307).	http://ndl.codeplex.com/	
µPLibrary is an helper library composed of managed drivers for common hardware that you can interface to your .Net Micro Framework board and some other useful components (PIR, TMP102, DS1307, SHT1x, anémomètre, etc.).	http://uplibrary.codeplex.com/	
The 'netduino Helpers' is a C# library providing hardware drivers and utility classes	http://netduinohelpers.codeplex.com/	

Exemples

Project Netduino data logger for race motorcycle based on Netduino + programmed using C# and .NetMF.	http://datalogger.codeplex.com/

Technologie .NET

Télécharger		http://goo.gl/bzSSo1
Guide de référence du programmeur VS2015		https://goo.gl/D0aPvE
Débuter en C# (Vidéos en Français) 		http://goo.gl/59AKhM
Débuter en C# (Cours en ligne)		http://goo.gl/jvSh6K
Plus loin avec le Coach C# (Cours en ligne)		http://goo.gl/YTY98z
NETMF v4.3 (Ressources, Téléchargement)		http://www.netmf.com/
Le Blog NETMF(Actualités)		http://blogs.msdn.com/b/netmfteam/

Sujets avancés

Timer	https://goo.gl/gaaJQY	
Multithreading	http://goo.gl/1hUFcJ	

Distributeur

http://www.mouser.fr/

Index

Table des illustrations