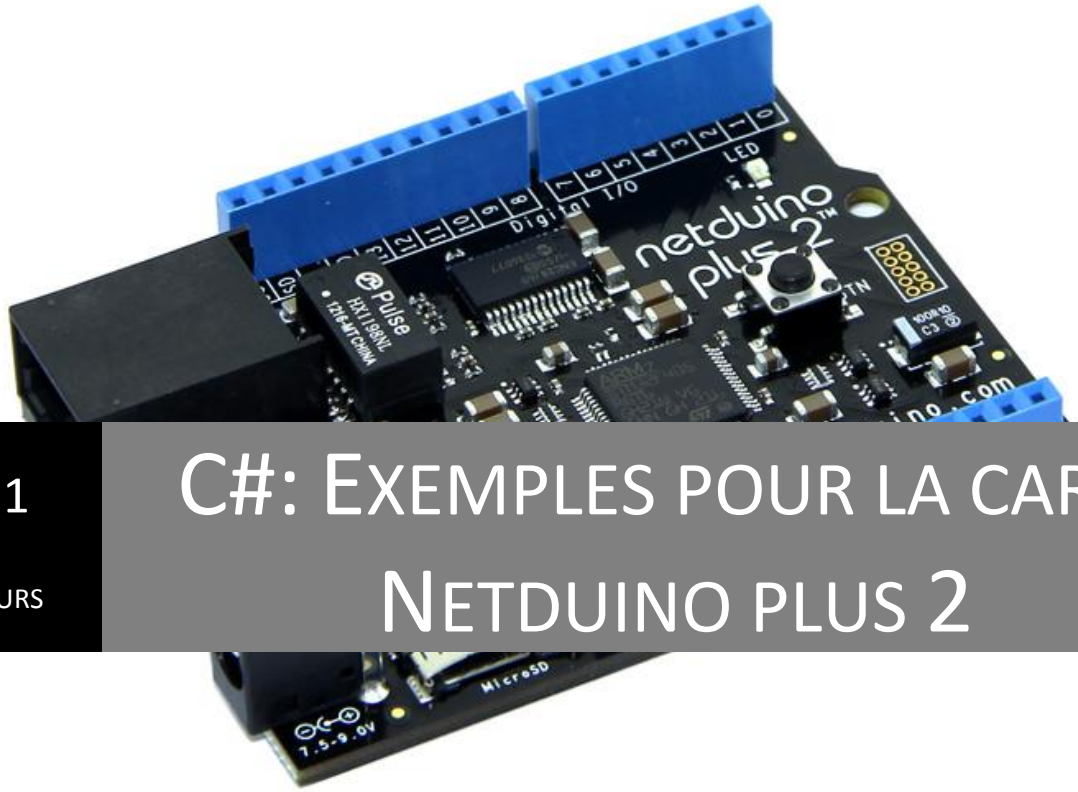
12/04/2015



PARTIE 1

CAPTEURS

ACTIONNEURS

C#: EXEMPLES POUR LA CARTE NETDUINO PLUS 2

philippemariano@gmail.com

Table des matières

PRESENTATION ..... **II**

REPERTOIRES DES PROJETS ASSOCIES AUX EXEMPLES ..... **III**

ORGANISATION D'UN EXEMPLE ..... **IV**

LE MATERIEL **1**

Identification des entrees / sorties de la carte Netduino plus 2 2
Le module Sensor Shield V2 Tinkerkit 3

LES LOGICIELS **4**

1. ENTREES, SORTIES NUMERIQUES **5**

1.1. Faire clignoter la « LED » de la carte Netduino ! 5
1.2. Commander une LED avec un bouton-poussoir ! 6
1.3. Commander un moteur pas a pas ITC-VNC-1 avec une carte EasyDriverStepperMotor V4.4 7
1.4. Interruption : commande d'une LED avec un bouton-poussoir ! 8
1.5. PWM: Faire varier la luminosite d'une LED 9
1.6. PWM : Commande d'un motoreducteur (GHM-16) equipe d'un codeur 10
1.7. PWM : Commande d'un servomoteur de modelisme 12

2. ENTREES ANALOGIQUES **13**

2.1. Regler la frequence de clignotement d'une LED avec un potentiometre ! 13

3. COMMUNICATION SERIE **14**

3.1. UART : Transmettre une valeur numerique via une liaison RS232 14
3.2. UART : Utiliser un afficheur LCD a commandes series (Module COMFILE ELCD-162) 15
3.3. UART : Transmettre des donnees avec des modules XBEE (Emission/Reception) 16
3.4. I2C : Chenillard sur huit LED reliees a un port d'E/S PCF8574 19
3.5. I2C : Commander un afficheur LCD (PCF2119) 20
3.6. I2C : Mesurer une distance avec un telemetre a ultrasons SRF08 21
3.7. I2C : Recopier l'etat de BP (PCF8574) sur des LED (PCF8574) 24
3.8. I2C : Lire la direction donnee par une boussole HMC6352 25
3.9. I2C : Mesurer la temperature ambiante avec un capteur TMP102 27
3.10. I2C : Commander deux motoreducteurs, equipes d'encodeurs, avec une carte Devantech MD25 28
3.11. I2C : Mesurer la luminosite ambiante avec un capteur TSL2561 30
3.12. Un fil : Mesurer la temperature ambiante avec un capteur DS18B20 (OneWire) 32

ANNEXES **33**

A1 - API Reference for .NET Micro Framework 33
A2 – Les types reconnus par Microsoft Visual Studio et le .NET Microframework 34
A3 – Shield Tinkerkit 35

BIBLIOGRAPHIE **36**

WEBOGRAPHIE **37**

DISTRIBUTEURS **38**

INDEX **39**

Présentation

Ce document est le **premier tome** d'un recueil de programmes écrits en **C#**. Ils illustrent la mise en œuvre de **capteurs** et de **prés-actionneurs** avec la carte **Netduino plus 2** développée par la société **Secret Labs**.

Un deuxième tome est consacré à la **communication sur un réseau local** et aux **objets connectés**.

Tous les programmes ont été compilés avec l'IDE **Visual Studio Express 2012**. Ils ciblent une carte **Netduino plus 2** et s'appuient sur la version **4.3 du microframework .NET**. Il est possible de réutiliser les fichiers sources avec d'autres cibles du même fabricant.

Les caractéristiques et la connectique de la carte Netduino plus 2 sont présentées dans la suite de ce document.

Un **shield Tinkerkit** est également décrit en annexe. Il permet de relier facilement différents capteurs et pré-actionneurs.

Les sources des **projets** Visual Studio sont répertoriées en page iii.

Elles sont disponibles au téléchargement.

Les projets comportant des classes maintenues sur **Github** sont identifiés par le symbole ☒. Il s'agit d'un lien hypertexte permettant d'accéder à la page web décrivant la classe.

Tous les exemples de ce document sont organisés selon le schéma de la page iv.

La documentation du microframework .NET 4.3 est consultable à partir du lien suivant :

http://msdn.microsoft.com/en-us/library/jj610646(v=vs.102).aspx

Répertoires des projets associés aux exemples

EXEMPLE	Répertoire du projet Visual Studio 2012 Express	Description		Information complémentaires dans le document « French Beginner Guide ebook1.03»
Num1	BlinkingLed	Ecrire sur une sortie numérique		
Num2	LightSwitch	Lire une entrée numérique et écrire sur une sortie		
Num3	Netduino_EasyStepperMot	Commander un moteur pas à pas avec une carte **EasyStepper Driver Motor V4.4**	☒	Chapitre 9 Entrées/Sorties numériques
Num4	LightSwitchINT	Gérer une Interruption		
Num5	NetduinoPWM	PWM : Faire varier la luminosité d'une Led		
Num6	NetduinoArdumoto	PWM : Faire varier la vitesse d'un moteur à courant continu		Chapitre 12 : Pulse With Modulation
Num7	NetduinoServo	PWM : Régler la position d'un servomoteur de modélisme		
An1	NetduinoPot	Lire une entrée analogique		Chapitre 14 : Entrées analogiques
Com1	NetduinoUART	UART : Transmettre une donnée		
Com2	NetduinoELCD_162	UART : Commander un LCD série **ELCD-162**	☒	Chapitre 17 : Interface série / 17.1 UART
Com3	NetduinoXBeeE	UART : Commander un module XBee (émetteur)		Chapitre 20 : Afficheur / 20.1 LCD
	NetduinoXBeeR	UART : Commander un module XBee (récepteur)		
Com4	NetduinoPCF8574	I2C : Commander un port **PCF8574** en sortie	☒	
Com5	NetduinoI2CLCD	I2C : Afficher sur un LCD BATRON (**PCF2119**)	☒	
Com6	NetduinoSRF08US	I2C : Mesurer une distance avec des ultrasons (module **SRF08)**	☒	
Com7	NetduinoI2CLEDBP	I2C: Commander un port **PCF8574** en entrée et en sortie	☒	
Com8	NetduinoHMC6352	I2C : Mesurer une direction (boussole **HMC6352**)	☒	Chapitre 17 : Interfaces séries /17.3 I2C
Com9	NetduinoTMP102	I2C : Mesurer une température (module **TMP102**)	☒	
Com10	NetduinoMD25	I2C : Commander des motoréducteurs CC équipés d'encodeurs (**MD25**)	☒	
Com11	NetduinoTSL2561	I2C : Mesurer la luminosité ambiante (module **TSL2561**)	☒	
Com12	NetduinoDS18B20	Bus One Wire : Mesurer une température (DS18B20)		Chapitre 17 : Interfaces séries /17.3 I2C

☒ Description de la classe spécifique au projet et page web de l'exemple sur **github**.

Organisation d'un exemple

Titre de l'exemple

Matériel utilisé

Explorateur de solution

Les bibliothèques à placer dans le projet.

Clic droit sur « Référence »

➔ Ajouter une référence

Code de l'exemple

Membres des classes spécifiques au projet

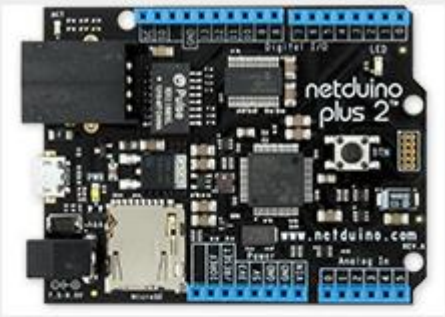
Eventuellement un lien vers Github

où sont maintenues les sources du code

Le matériel

L'écosystème Netduino devrait plutôt être appelé l'écosystème .Net Micro Framework (NETMF). Le Micro Framework .Net est un sous-ensemble du Framework .Net créé par **Microsoft open source**. Netduino est une plateforme open source, développé par la société Secret Labs.

La carte Netduino plus 2 a la même configuration de broches que l'Arduino Uno, et est compatible avec un grand nombre de shield Arduino. La gamme complète des matériels Netduino peut être trouvée sur le site Netduino.

Netduino plus 2	Netduino Go
	

Processor and memory

microcontroller	STMicro STM32F4 (32 bits)
speed	168 MHz (Cortex-M4)
Code storage	384KB
Operating system	NET Micro Framework 4.3 (2014)

Input and output

	Netduino plus 2	Netduino Go
networking	ethernet : 10 Mbit/s	add-on : ethernet gobus module currently in design
arduino shield compatibility	works with most arduino shields (some require .net mf drivers)	add-on : shield base gobus module
digital i/o	22 gpio, 6 pwm, 4 uart, i2c, spi	add-on : gobus i/o modules (gpio, pwm, uart, spi, and more)
analog inputs	6 adc channels (12-bit)	add-on : analog gobus modules
storage	micro sd (up to 2 Gb)	add-on : sd card gobus module currently in design
gobus ports	none	8 gobus ports

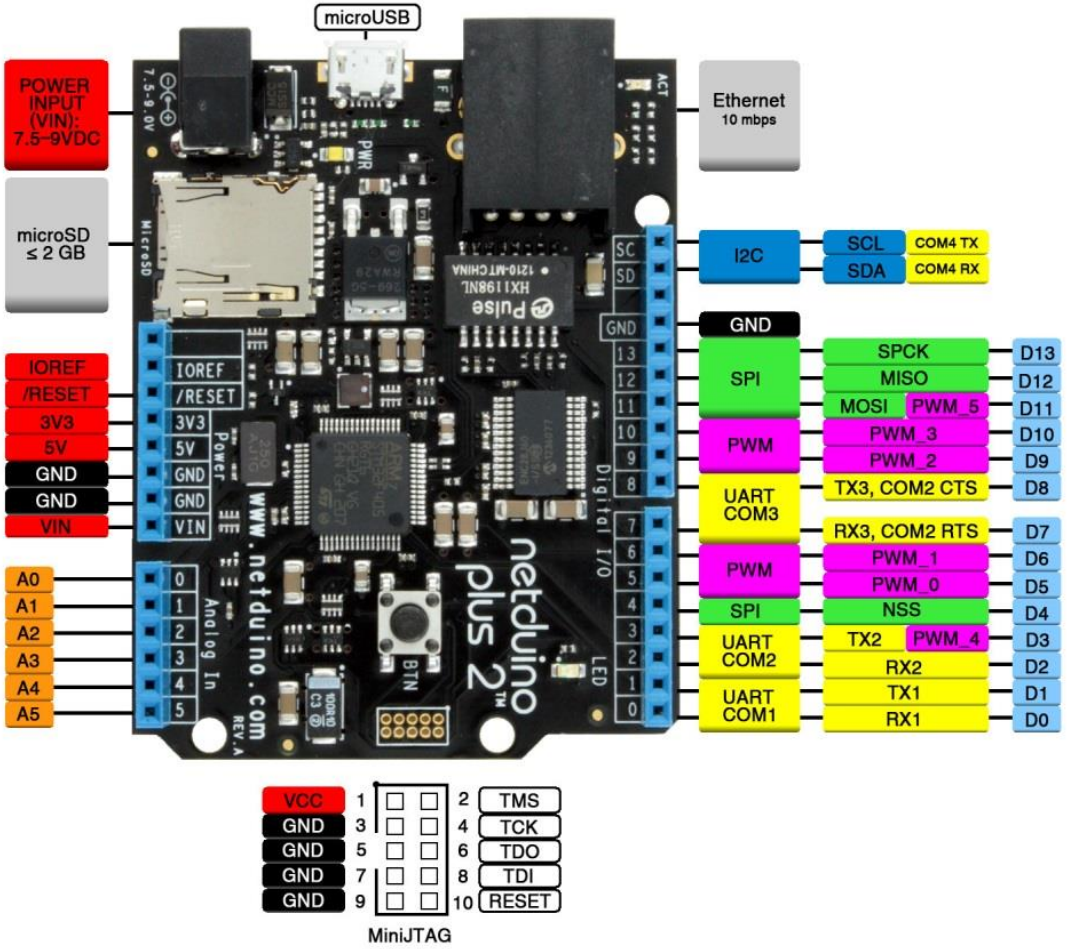
Identification des entrées / sorties de la carte Netduino plus 2

Netduino Plus 2

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

Le module Sensor Shield V2 Tinkerkit

Le module Sensor Shield V2 TinkerKit (fig2) permet de raccorder facilement et sans soudure des capteurs et des pré-actionneurs sur une carte Netduino plus 2.

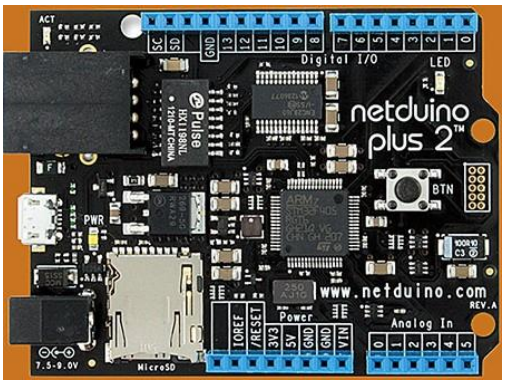


Figure 1 : Netduino plus 2

Figure 2 : Sensor Shield V2Tinkerkit

Correspondance entre les dénominations des connexions de la carte Netduino, celles du shield Tinkerkit et celles du logiciel Visual Studio (VS 2012).

Digital I/O features

VS 2012	Netduino plus 2		Tinkerkit
| | v		Secondary Features	
	SC	SCL/UART 4TX	TWI
	SD	SDA/UART 4 RX	TWI
Pins.GPIO_PIN_D13	13	SPCK	-
Pins.GPIO_PIN_D12	12	MISO	-
Pins.GPIO_PIN_D11	11	PWM / MOSI	O0
Pins.GPIO_PIN_D10	10	PWM	O1
Pins.GPIO_PIN_D9	9	PWM	O2
Pins.GPIO_PIN_D8	8	UART 3 TX / UART 2 CTS	-
Pins.GPIO_PIN_D7	7	UART 3 RX / UART 2 RTS	-
Pins.GPIO_PIN_D6	6	PWM	O3
Pins.GPIO_PIN_D5	5	PWM	O4
Pins.GPIO_PIN_D4	4		-
Pins.GPIO_PIN_D3	3	UART 2 TX / PWM	O5
Pins.GPIO_PIN_D2	2	UART 2 RX	-
Pins.GPIO_PIN_D1	1	UART 1 TX	Serial
Pins.GPIO_PIN_D0	0	UART 1 RX	Serial

Analog Input

VS 2012	Netduino plus 2	Tinkerkit
Pins.GPIO_PIN_A5	5	I5
Pins.GPIO_PIN_A4	4	I4
Pins.GPIO_PIN_A3	3	I3
Pins.GPIO_PIN_A2	2	I2
Pins.GPIO_PIN_A1	1	I1
Pins.GPIO_PIN_A0	0	I0

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

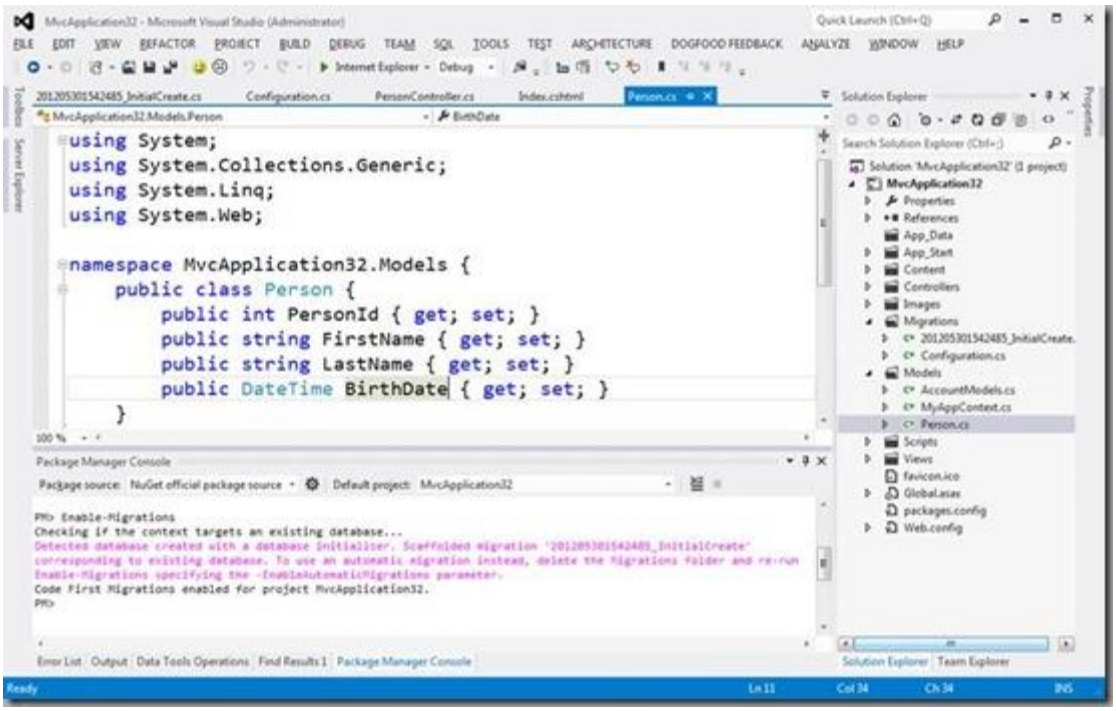
Pour configurer les entrées / sorties de la carte :

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

Les logiciels

La véritable force de l'écosystème NETMF est le framework .NET, et 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) avec l'environnement de développement (IDE) **Microsoft Visual Studio Express (ou professionnel)**. Il est nécessaire d'installer le Micro Framework .NET correspondant à la version de l'IDE et le SDK Secret Labs nécessaire à la carte ciblée.



Fonctionnalités particulièrement appréciables de l'IDE Microsoft Visual studio Express 2012 :

- **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)

Visual Studio Express 2012

ou

Professional 2012 Microsoft Visual Studio

Microsoft .NET Micro Framework

Netduino SDK v4.3.1 (February 2014)

---

1. Entrées, sorties numériques

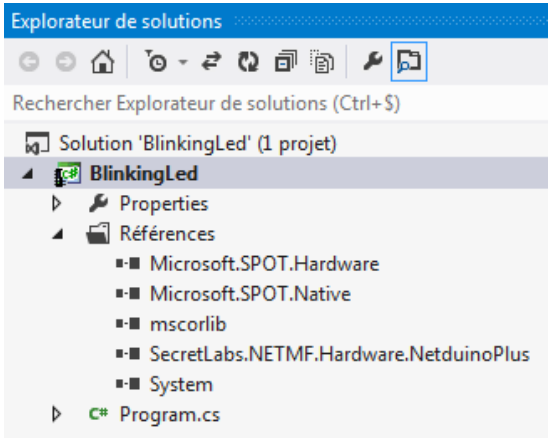
1.1. Faire clignoter la « LED » de la carte Netduino !

Code C# de l'exemple Num1

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

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

      while (true)
       {
         ledPort.Write(true);    // turn on LED
         Debug.Print("Led éclairée");
         Thread.Sleep(500);      // wait 500 ms
         ledPort.Write(false);   // turn off LED
         Debug.Print("Led éteinte");
         Thread.Sleep(500);      // wait 500 ms
       }
      }
   }
}
```

Espaces de noms

Bibliothèques

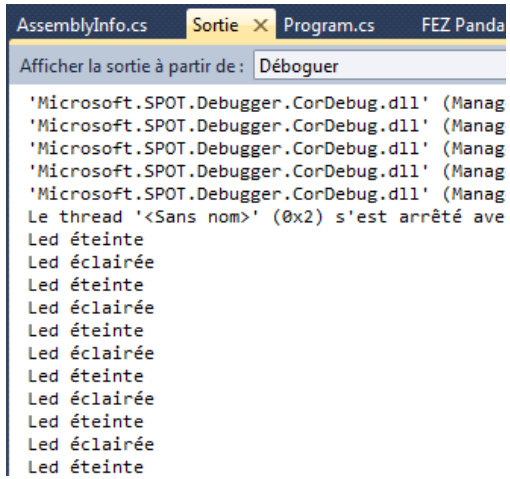


La construction de l'objet LED semble complexe mais l'auto complétion (**IntelliSense)** simplifie le travail !

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

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

Pas à pas
F10 : Step Over
F11 : Step into



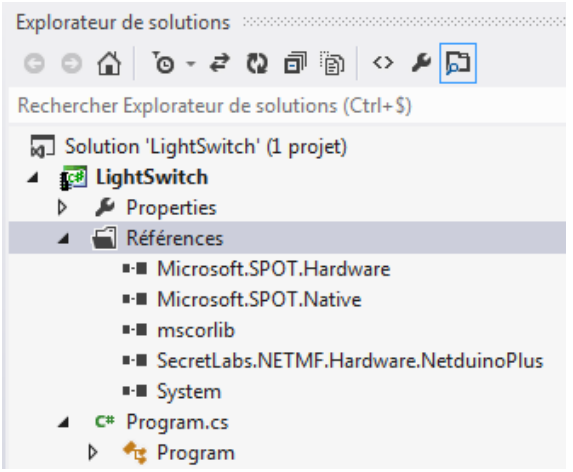
[OutputPort Class]

OutputPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool)

Write(bool)

.net Micro Framework

1.2. Commander une LED avec un bouton-poussoir !



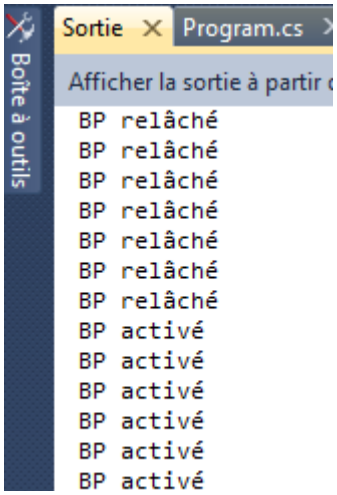
Code C# de l'exemple Num2

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

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

      while (true)
      {
        if (Button.Read())
        {
          led.Write(true);
          Debug.Print("BP activé");
        }
        else
        {
          led.Write(false);
          Debug.Print("BP relâché");
        }
        Thread.Sleep(100);          // 100 milliseconds
      }
    }
  }
}
```

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

Utilisation du **debugger**

Pas à pas
F10 : Step Over
F11 : Step into



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)

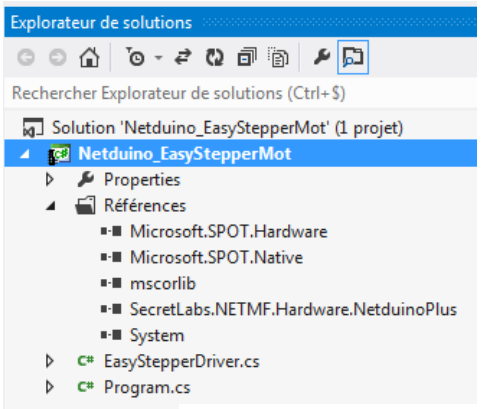
.net Micro Framework

1.3. Commander un moteur pas à pas ITC-VNC-1 avec une carte EasyDriverStepperMotor V4.4

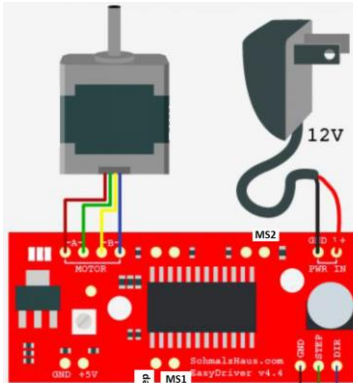
Code C# de l'exemple Num3

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

CI Allegro 3967 MotorDriver: EasyDriver stepper Motor V4.4
Moteur : ITC-CNC-1

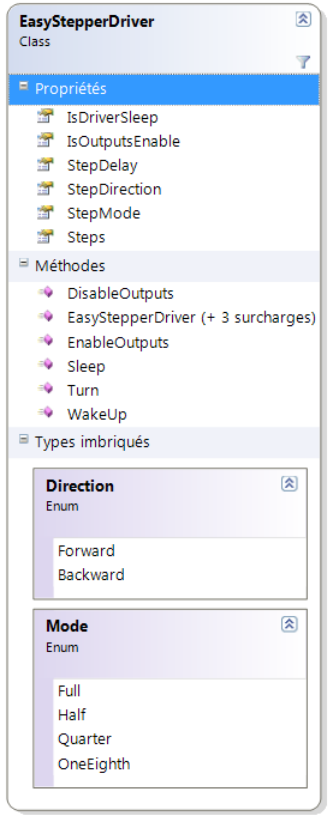



```
using System;
using System.Threading;
using Microsoft.SPOT;
using SecretLabs.NETMF.Hardware.NetduinoPlus;
using ToolBoxes;

namespace TestNetduinoStepper
{
 public class Program
   {
      public static void Main()
      {// Programme de test d'un moteur pas à pas ITC-VNC-1 http://astrojbm.free.fr/bricolages/picastro/Mecanique/ITC_CNC_1_FR.pdf
       // avec une carte EasyDriver V4.4 (N=200 pas - U=12V - C=200g/cm)
         var time = 2000; UInt16 delay = 2; UInt32 nbpas = 200;
         var stepper = new EasyStepperDriver(Pins.GPIO_PIN_D8, Pins.GPIO_PIN_D9, Pins.GPIO_PIN_D10, Pins.GPIO_PIN_D11, Pins.GPIO_PIN_D12,Pins.GPIO_PIN_D13);

          stepper.WakeUp(); // Activation de la logique de commande et des sorties

          while (true)
          {// Exemple d'utilisation de la méthode Turn() et des propriétés Steps, StepMode, StepDirection et StepDelay
             stepper.EnableOutputs(); // Activation des sorties
             Debug.Print("Sleep= " + stepper.IsDriverSleep + " Enable= " + stepper.IsOutputsEnable);
             Debug.Print("Full Forward");                                    // 360° pour le moteur ITC-VNC-1
             stepper.Turn(nbpas, EasyStepperDriver.Direction.Forward, delay,EasyStepperDriver.Mode.Full);
             Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " +
             stepper.StepDirection + " time= " + stepper.StepDelay + "ms" + "\n");
             stepper.DisableOutputs(); Thread.Sleep(time); // Désactivation des sorties (protection bobines)

             Debug.Print("Half Backward"); stepper.EnableOutputs();   // 180° pour le moteur ITC-VNC-1
             stepper.Turn(nbpas, EasyStepperDriver.Direction.Backward, delay, EasyStepperDriver.Mode.Half);
             Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " +
             stepper.StepDirection + " time= " + stepper.StepDelay + "ms" + "\n");
             stepper.DisableOutputs(); Thread.Sleep(time); // Désactivation des sorties

             Debug.Print("Quater Forward"); stepper.EnableOutputs();  // 90° pour le moteur ITC-VNC-1
             stepper.Turn(nbpas, EasyStepperDriver.Direction.Forward, delay, EasyStepperDriver.Mode.Quarter);
             Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " +
             stepper.StepDirection + " time= " + stepper.StepDelay + "ms" + "\n");
             stepper.DisableOutputs(); Thread.Sleep(time); // Désactivation des sorties

             Debug.Print("OneEighth Backward"); stepper.EnableOutputs(); // 45° pour le moteur ITC-VNC-1
             stepper.Turn(nbpas, EasyStepperDriver.Direction.Backward, 1, EasyStepperDriver.Mode.OneEighth);
             Debug.Print("Pas= " + stepper.Steps + " Mode= " + stepper.StepMode + " Dir= " +
             stepper.StepDirection + " time= " + stepper.StepDelay + "ms" + "\n");
             stepper.DisableOutputs();Thread.Sleep(2 * time);// Désactivation des sorties pendant la temporisation
          }
      }
   }
}
```



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

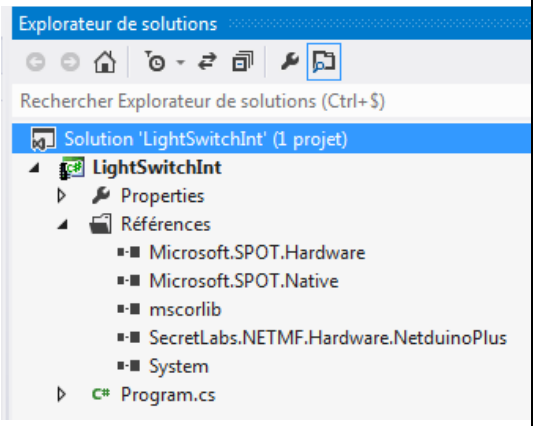
DIR	Direction
H	Forward
L	Backward



Connexions à la carte Netduino : **DIR -> Di8** **Step -> Di9** ***MS_1 -> Di11** ***MS_2 -> Di12** ***/Sleep -> Di10** ***/Enable ->Di13**

*** Option**

1.4. Interruption : commande d'une LED avec un bouton-poussoir !



Code C# de l'exemple Num4

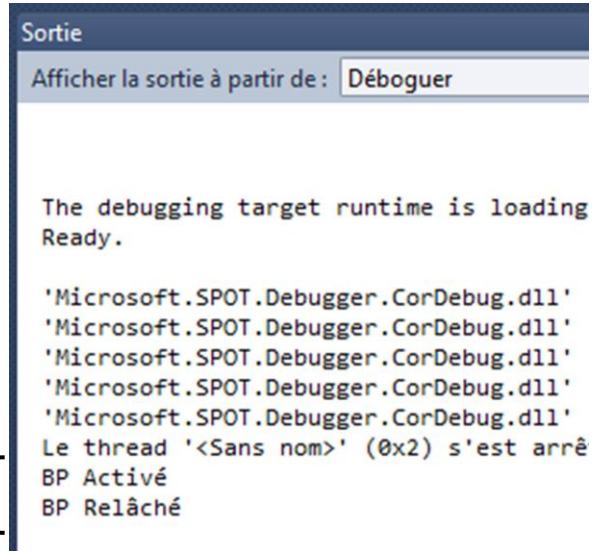
```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

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

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

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

   static void IntButton_OnInterrupt(uint port, uint state, DateTime time)
    {
       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.

Utilisation du **debugger**

Pas à pas
F10 : Step Over
F11 : Step into

Une action est affichée seulement lors d'une interruption.



InterruptPort Class

- ClearInterrupt()
- DisableInterrupt()
- EnableInterrupt()
- InterruptPort(Microsoft.SPOT.Hardware.Cpu.Pin, bool, Microsoft.SPOT.Hardware.Port.ResistorMode, Microsoft.SPOT.Hardware.Port.InterruptMode)
- Interrupt

1.5. PWM : Faire varier la luminosité d'une LED

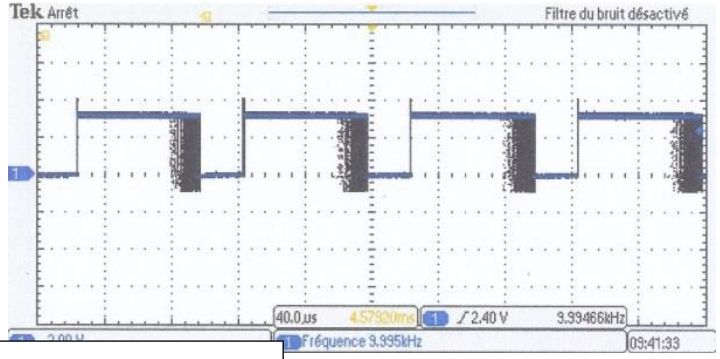
Remarque : La luminosité de la LED croit périodiquement (10% < α < 90%)

Code C# de l'exemple Num5

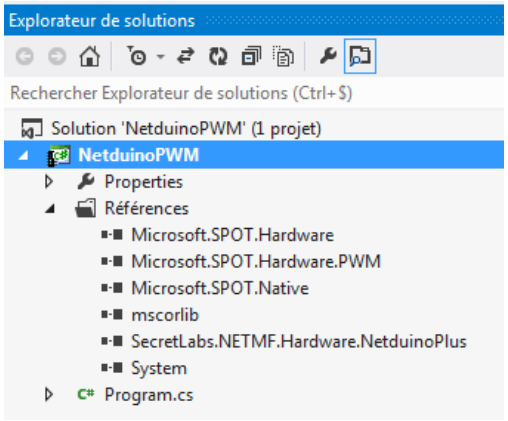

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

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

            pwm.Start();

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

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

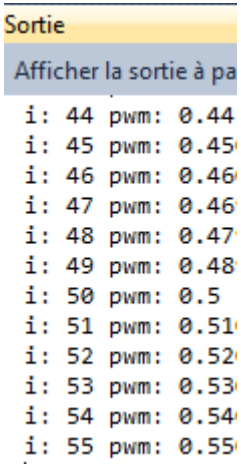


Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S.

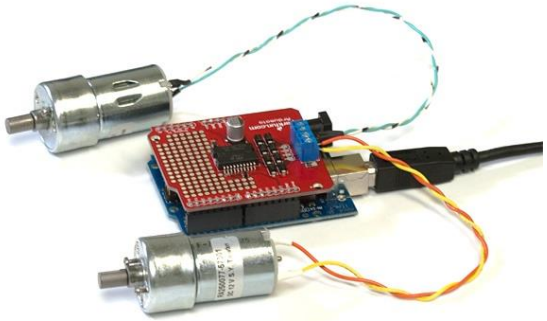


Utilisation du **debugger** pour visualiser le réglage du rapport cyclique.

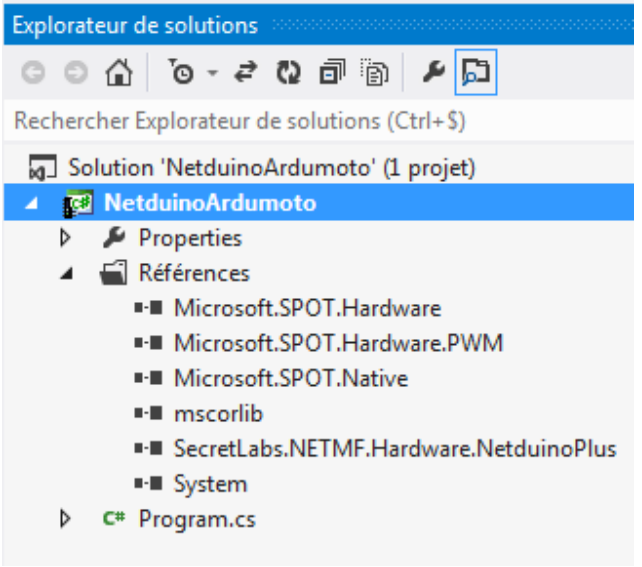
Pas à pas
F10 : Step Over
F11 : Step into



1.6. PWM : Commande d'un motoréducteur (GHM-16) équipé d'un codeur



NetDuino plus2 + Ardumoto



Code C# de l'exemple Num6

```
using System;
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 en Hz

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

         PWMB.Start();

         while (true)
         {// Rampe d'accélération

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

- PWM_ONBOARD_LED
- PWM_PIN_D10
- PWM_PIN_D11
- PWM_PIN_D3
- PWM_PIN_D5
- PWM_PIN_D6
- PWM_PIN_D9
- PWM_NONE

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

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

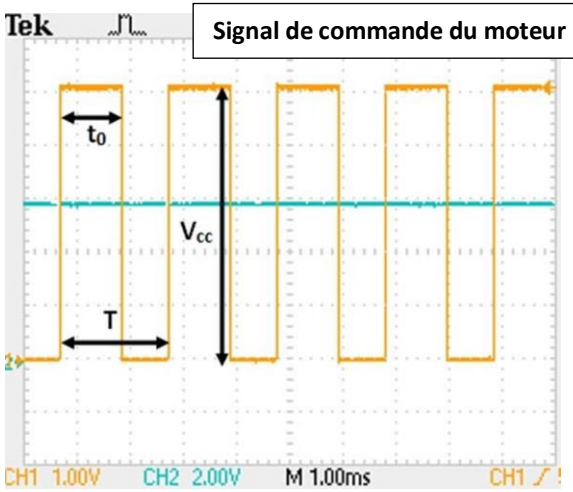
Connectique

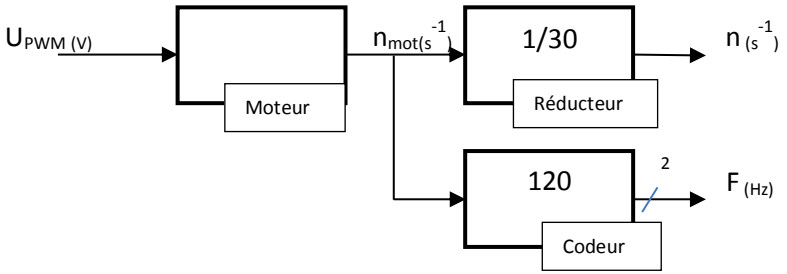
Netduino	Ardumoto	Moteur
Di3	PWMA	A
Di12	DIRA	
Di11	PWMB	B
Di13	DIRB	

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

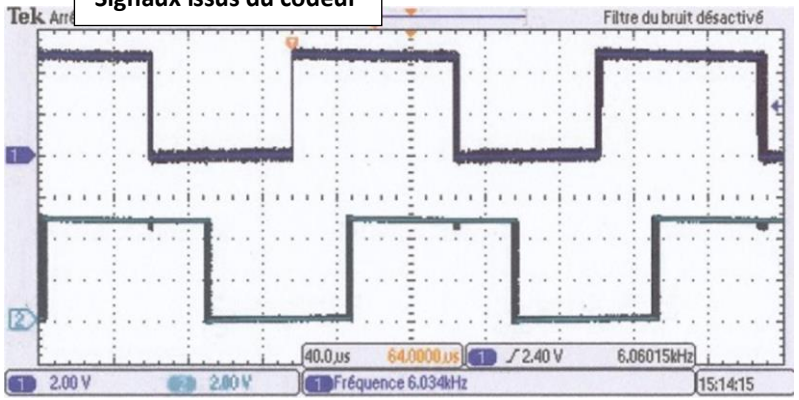
Motoréducteur Lynxmotion GHM-16	
• Voltage: 6-12vdc • RPM: 200 • Couple: 0.78Kg-cm (10.83o • Réduction: 30:1 • Poids : 0.34 livres (154g) • Diamètre extérieur : 37mm • Diamètre (essieu): 6mm	
Encodeur en quadrature lynxmotion **E4P-120-079-HT**	
• Alimentation : 5V Cycles par révolution: 120 • Comptes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

Signal de commande du moteur U_{PWM}

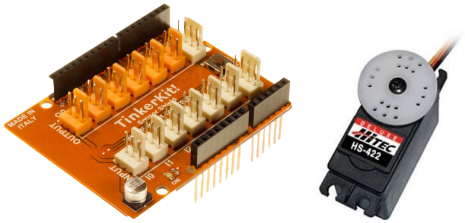




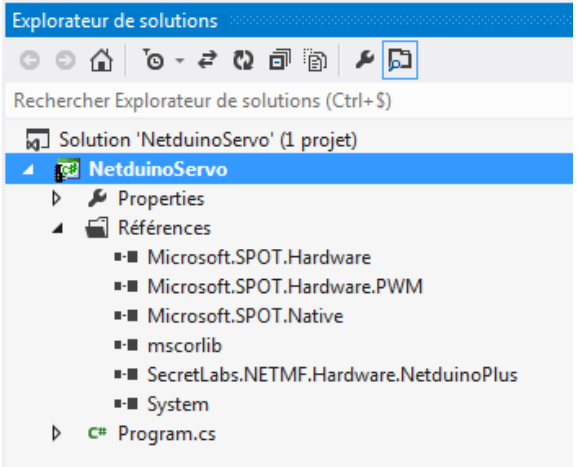
Signaux issus du codeur



1.7. PWM : Commande d'un servomoteur de modélisme



[Shield Tinkerkit V2](#) sur Netduino plus 2
+ [Servomoteur](#)



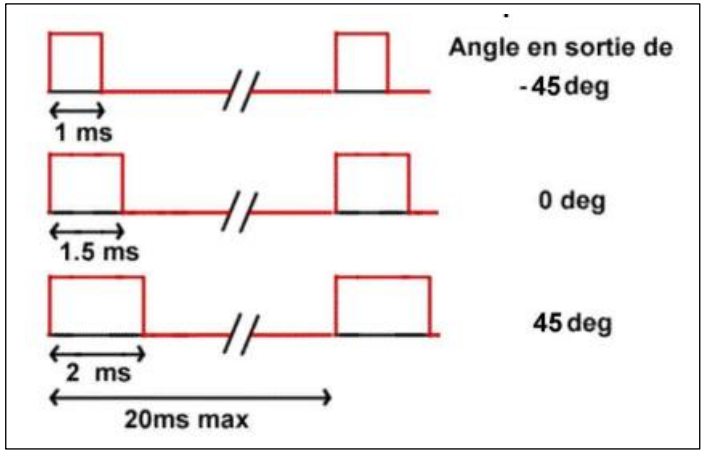
Code C# de l'exemple Num7

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

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

            // Configuration de l'E/S numérique 3 de la NetDuino en 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.

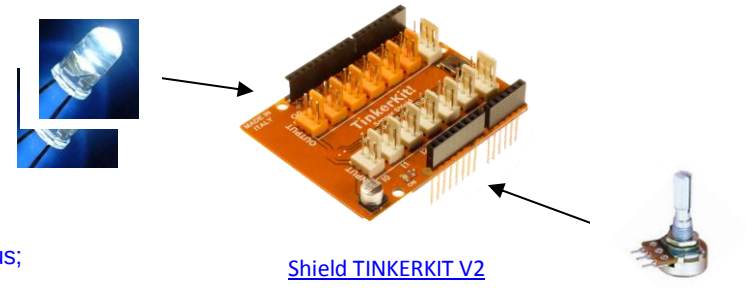
[PWM Class](#)

- 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

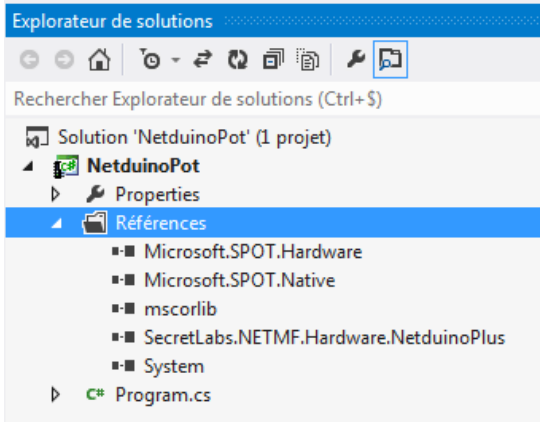


2. Entrées analogiques

2.1. Régler la fréquence de clignotement d'une LED avec un potentiomètre !



Shield TINKERKIT V2
sur Netduino plus 2

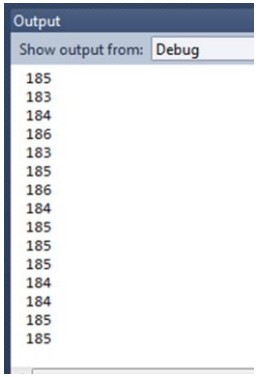


Code C# de l'exemple An1

```
using System.Threading;

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

using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TestNetduinoPot
{
    public class Program
    {
        public static void Main()
        {
            OutputPort 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)
            AnalogInput 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);
            }

        }
    }
}
```

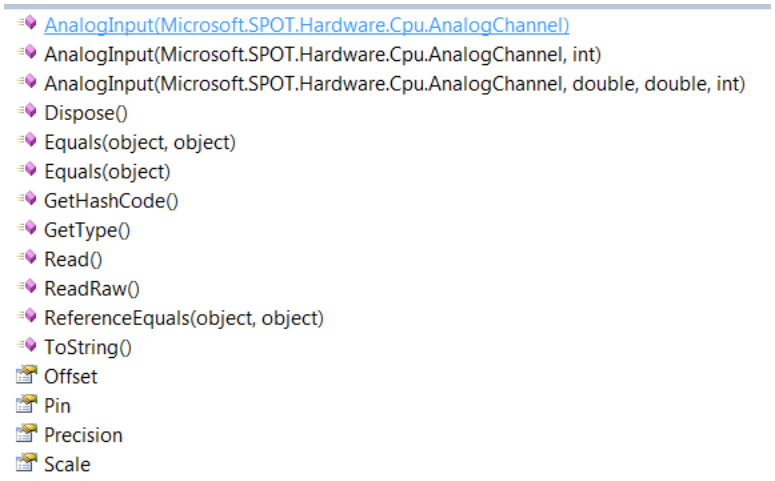
> Utiliser **impérativement** la classe NETMF pour la déclaration de l'entrée analogique et la classe SecretLabs pour la déclaration de la sortie numérique.

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



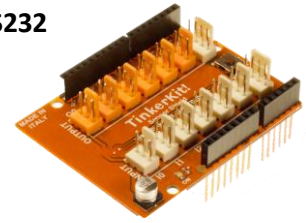
Pas à pas
F10 : Step Over
F11 : Step into

AnalogInput Class



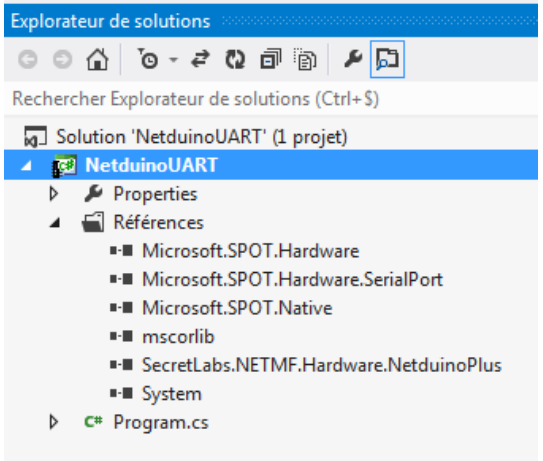
3. Communication série

3.1. UART : Transmettre une valeur numérique via une liaison RS232



Shield TINKERKIT V2

Convertisseur TTL <-> RS232



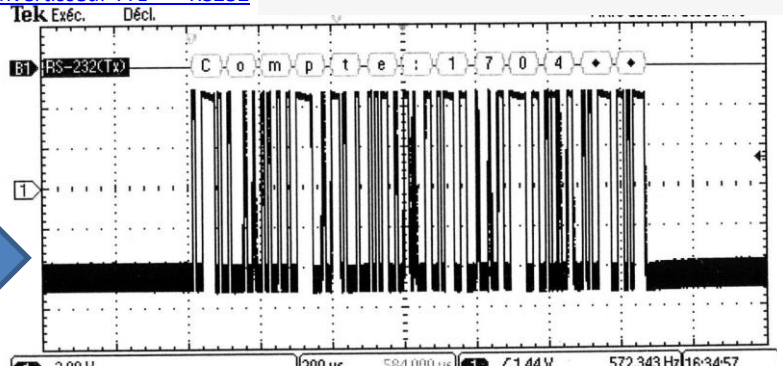
Code C# de l'exemple Com1

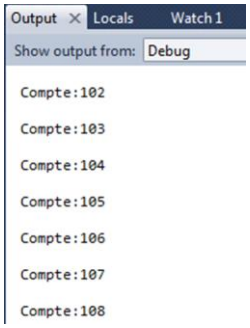
```
using System.Text;
using System.Threading;
using System.IO.Ports;
using Microsoft.SPOT;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TestNetduinoUART
{
    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)

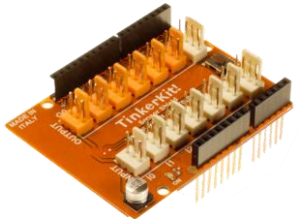
- Close()
- DiscardInBuffer()
- DiscardOutBuffer()
- Flush()
- Open()
- Read(byte[], int, int)
- Seek(long, System.IO.SeekOrigin)
- SerialPort(string, int, System.IO.Ports.Parity, int, System.IO.Ports.StopBits)
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)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

3.2. UART : Utiliser un afficheur LCD à commandes séries (Module COMFILE ELCD-162)

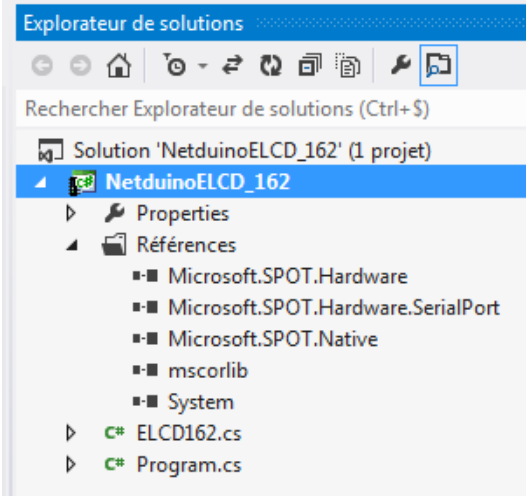
Code C# de l'exemple Com2

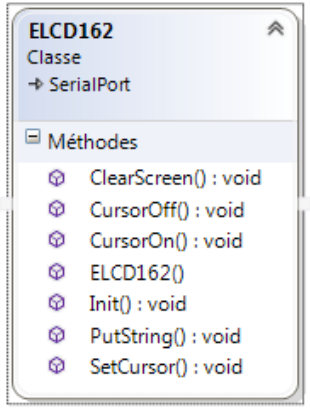
```
using System.Threading;
using ToolBoxes;

namespace TestNetduinoELCD_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();
        }
      }
   }
}
```

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



Shield TINKERKIT V2





Page Web de la classe SerialELCD162 +
Code de l'exemple

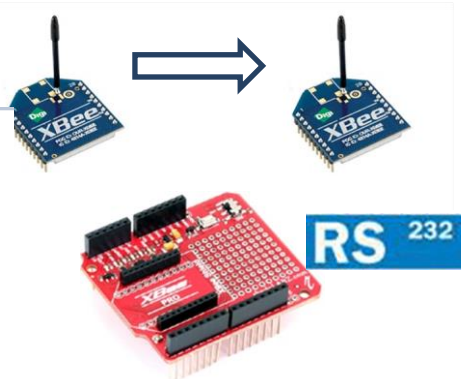
3.3. UART : Transmettre des données avec des modules XBEE (Emission/Réception)

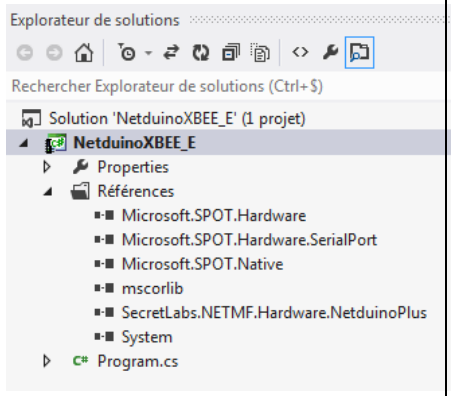
Code C# de l'exemple Com3

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


using SecretLabs.NETMF.Hardware.NetduinoPlu


namespace TestNetduinoXBEE_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
            UInt16 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

                }
            }
        }
    }
}
```

SerialPort Class (partielle)

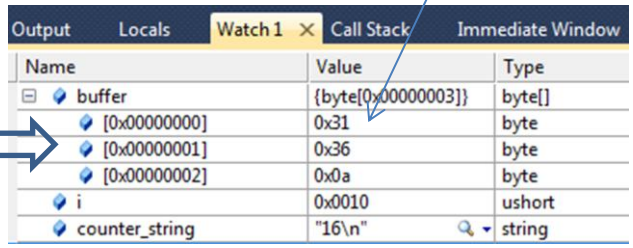
- Close()
- DiscardInBuffer()
- DiscardOutBuffer()
- Flush()
- Open()
- Read(byte[], int, int)
- Seek(long, System.IO.SeekOrigin)
- SerialPort(string, int, System.IO.Ports.Parity, int, System.IO.Ports.StopBits)
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)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

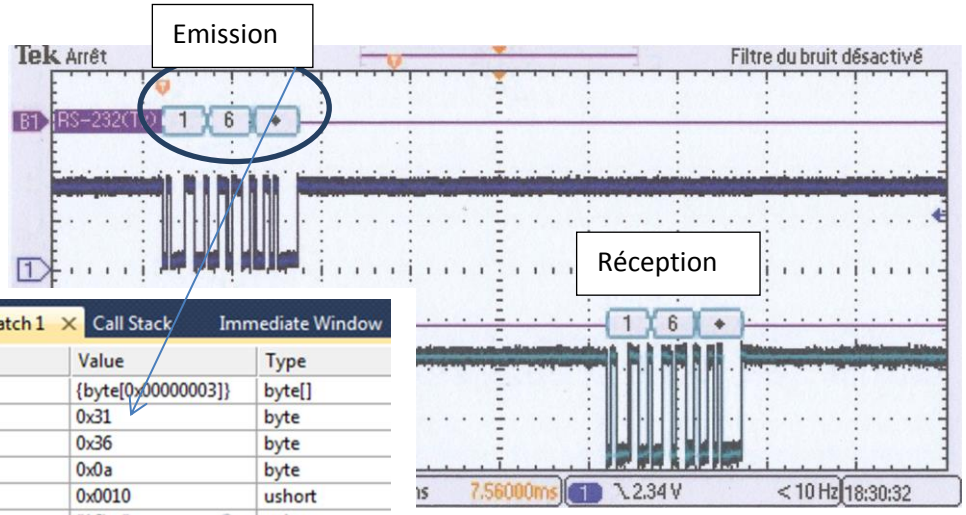




Utiliser **impérativement** les classes SecretLabs pour la déclaration des E/S.

Transmission de "16 <CR>"

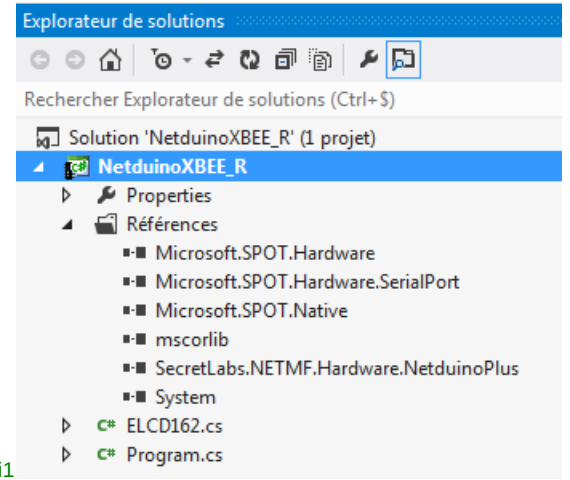




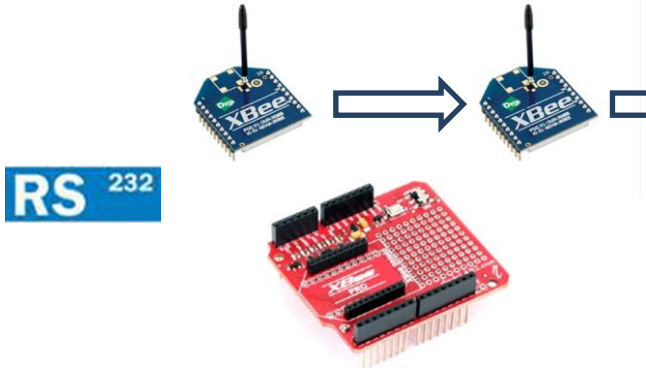
Transmettre des données avec des modules XBEE (Emission**/Réception)**

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.IO.Ports.StopBits)
- SerialPort(string, int, System.IO.Ports.Parity, int)
- SerialPort(string, int, System.IO.Ports.Parity)
- SerialPort(string, int)
- SerialPort(string)




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

using SecretLabs.NETMF.Hardware.NetduinoPlus;

using ToolBoxes;

namespace TestNetduinoXBEE_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).aspx
         // 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 Di0 et XBEE_TX sur Di1
           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);       }
```



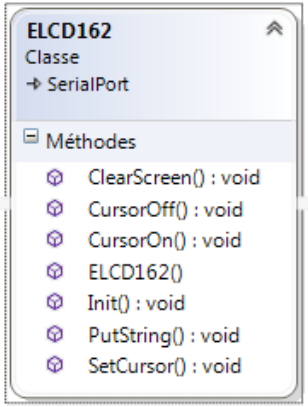
Afficheur 2x16 équipé d'un
module ELCD162 COMFILE Série (19200b/s)

RS 232

Suite du code

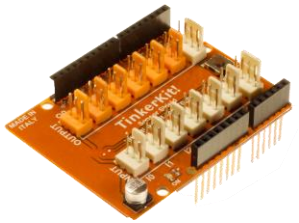
Page Web de la classe SerialELCD162

```
// 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;
        }
    }
  }
}
```



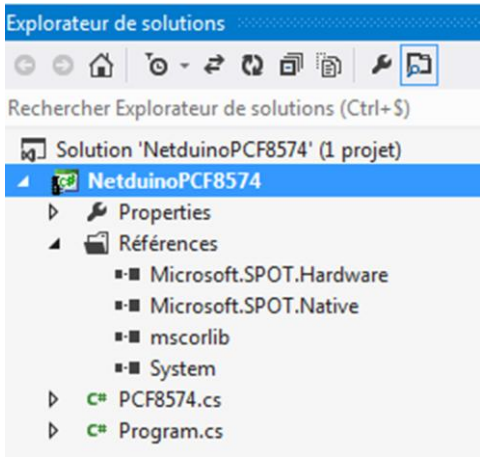
3.4. 2C : Chenillard sur huit LED reliées à un port d'E/S PCF8574A

Code C# de l'exemple Com4



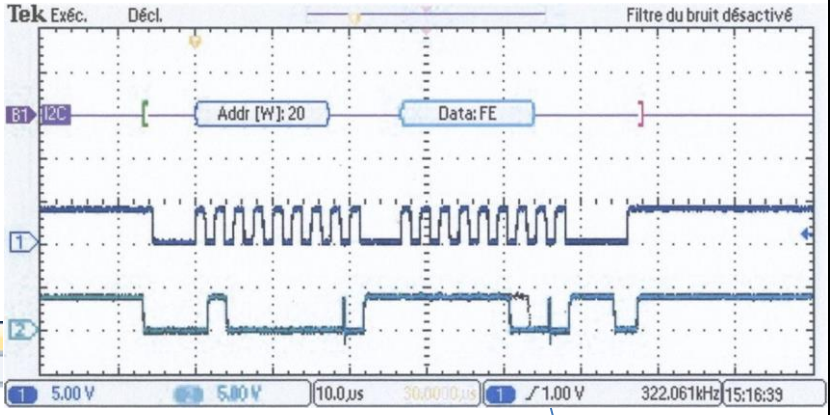
Shield TINKERKIT V2
sur Netduino plus 2

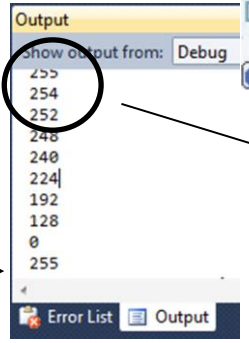
X 8




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


using ToolBoxes;

namespace TestNetduinoPCF8574
{
    public class Program
    {
        public static void Main()
        { // Pour accéder au bus I2C, relier le PCF8574A 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
            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
            PCF8574 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



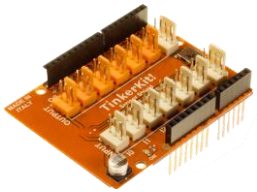


Utilisation du **debugger** pour visualiser les valeurs placées sur le bus I2C

Pas à pas
F10 : Step Over
F11 : Step into

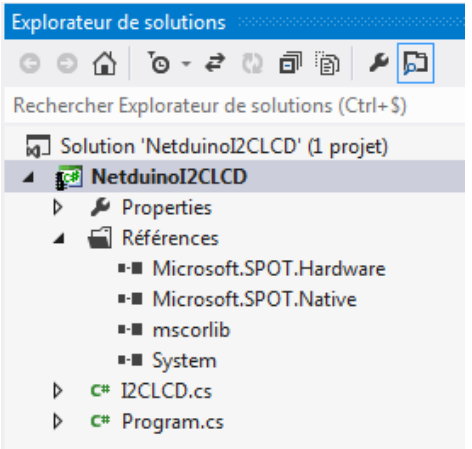
Remarque : L'éclairage d'une LED est commandé par un niveau logique 0.

3.5. I2C : Commander un afficheur LCD (PCF2119)



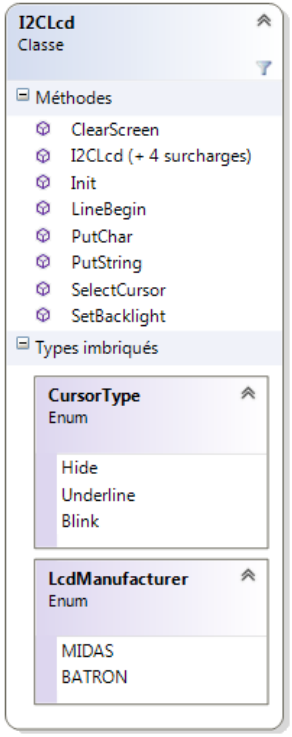
Shield TINKERKIT V2
sur Netduino plus 2

LCD I2C 2 x16



Code C# de l'exemple Com5

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

namespace TestNetduinoI2CLCD
{
    public class Program
    {
     public static void Main()
     { // Pour accéder au bus I2C, relier le LCD au connecteur TWI de la carte Tinkerkit.
       // Placer des résistances de rappel entre le +5V et les sorties SCL et SDA
       byte InitJauge = 0x5A; // Etat initial d'un caractère personalisé "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
       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;
       }
     }
    }
}
```



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

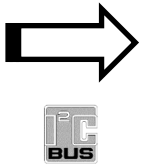
3.6. I2C : Mesurer une distance avec un télémètre à ultrasons SRF08

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

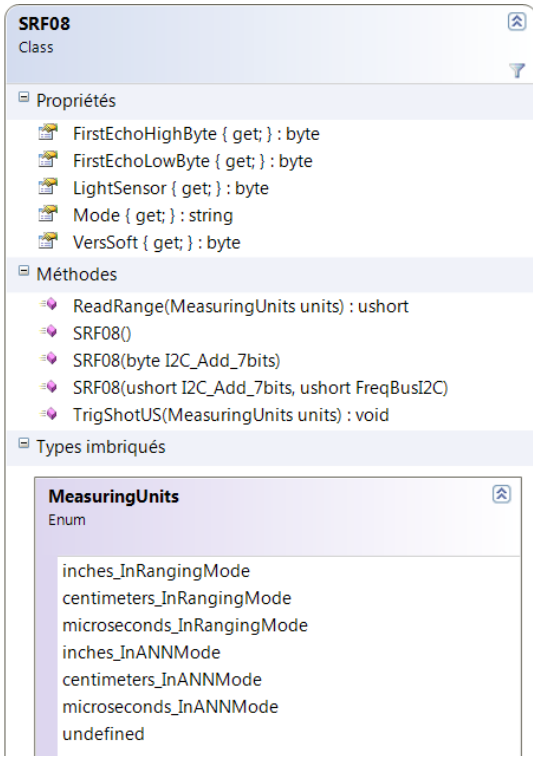
Code C# de l'exemple Com6

[Page Web de la classe TelemetreUS +
Code de l'exemple]

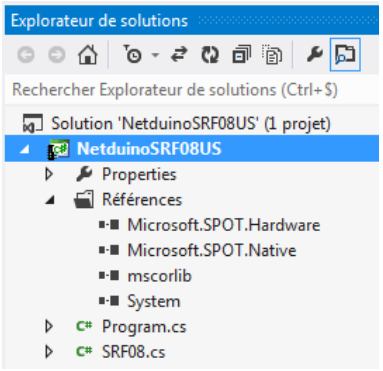
[Shield TINKERKIT V2]
sur Netduino plus 2



[SRF08
Transducteur à ultrasons]




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

using ToolBoxes;

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**

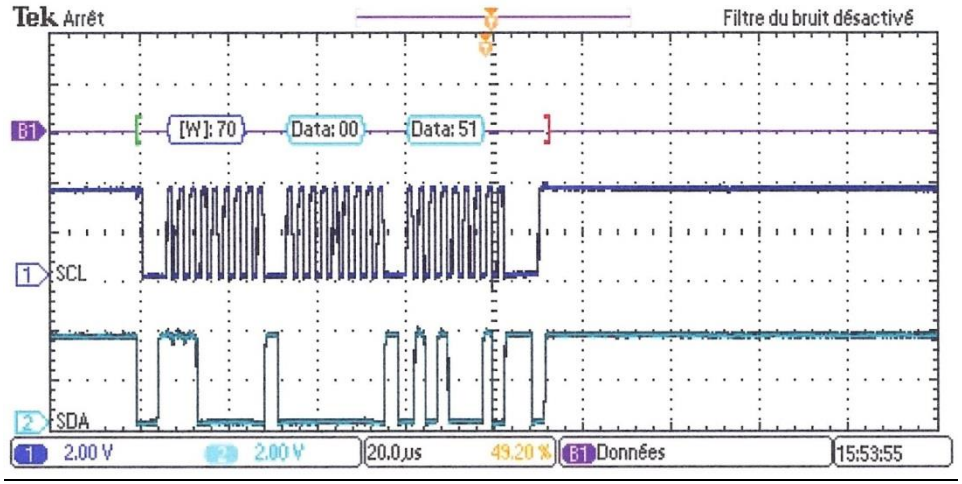
Pas à pas
F10 : Step Over
F11 : Step into

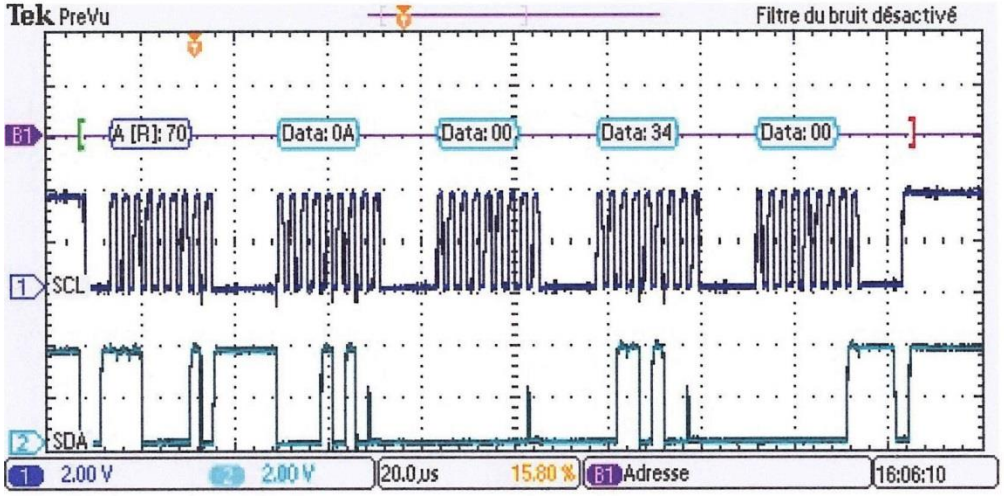
```
_____________________________________________
Distance: 55cm
1st Echo HighByte: 0  1st Echo LowByte: 55
Distance: 21inches
1st Echo HighByte: 0  1st Echo LowByte: 21
Distance: 6824µs
1st Echo HighByte: 12  1st Echo LowByte: 168
Light: 139
_____________________________________________
Mode: Ranging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

I2C : Mesurer une distance avec un télémètre à ultrasons SRF08 (Suite)

Chronogrammes

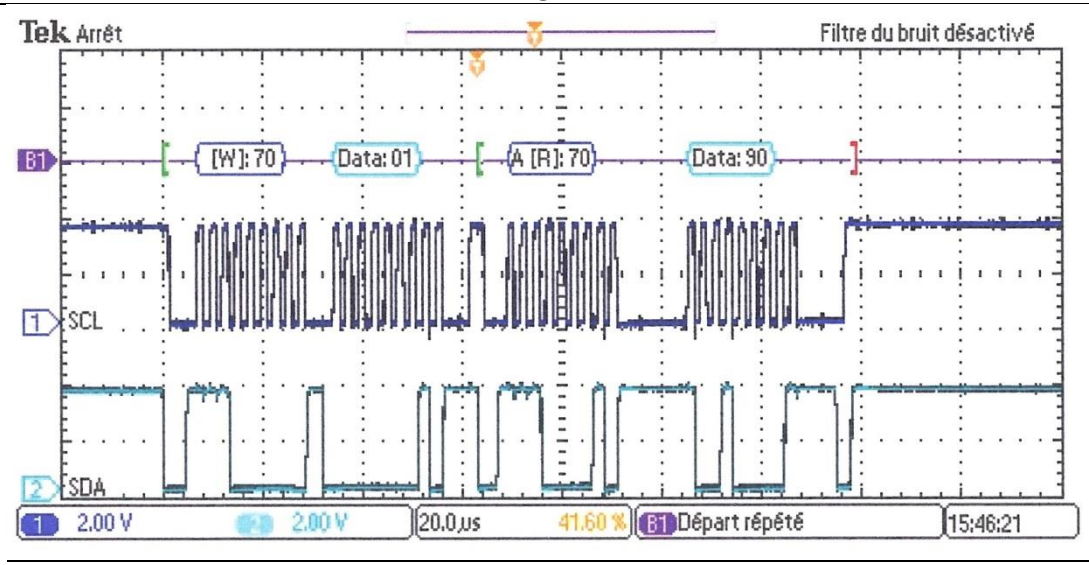
<table>
<tr><th>Méthode</th><th>Transaction</th><th>Chronogrammes</th></tr>
<tr><td rowspan="3">ReadRange(SRF08.UnitType.centimeters)</td><td>Déclenchement du tir US

Start
@SLA + W
N° registre
Commande
Stop</td><td> 
</td></tr>
<tr><td colspan="2">Attente 75ms</td></tr>
<tr><td>Lecture de la mesure

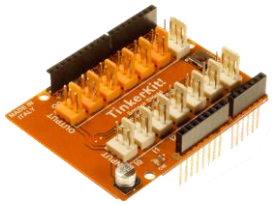
Start
@SLA + R
Reg0 (Software Rev)
Reg1 (Light Sensor)
Reg2 (1st Echo low)
Reg3(1st Echo High)
Stop</td><td> 
</td></tr>
</table>

I2C : Mesurer une distance avec un télémètre à ultrasons SRF08 (Suite)

Chronogrammes

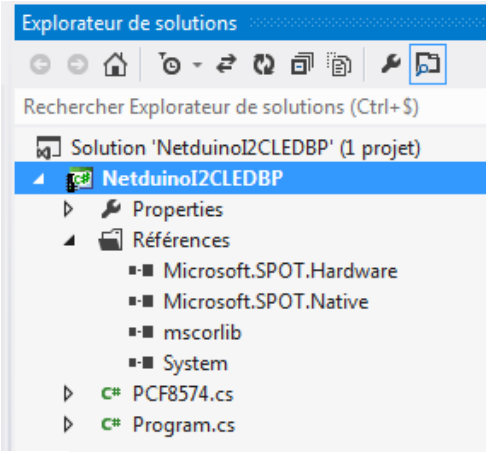
Propriété	Transaction	Chronogrammes
`I2CTelemeter.LightSensor`	**Start** @SLA+W n°registre **Repeated Start** @SLA+R Donnée située en n° registre **Stop**	

3.7. I2C : Recopier l'état de BP (PCF8574A) sur des LED (PCF8574A)



Shield TINKERKIT V2
sur Netduino plus 2

PCF8574A



Code C# de l'exemple Com7

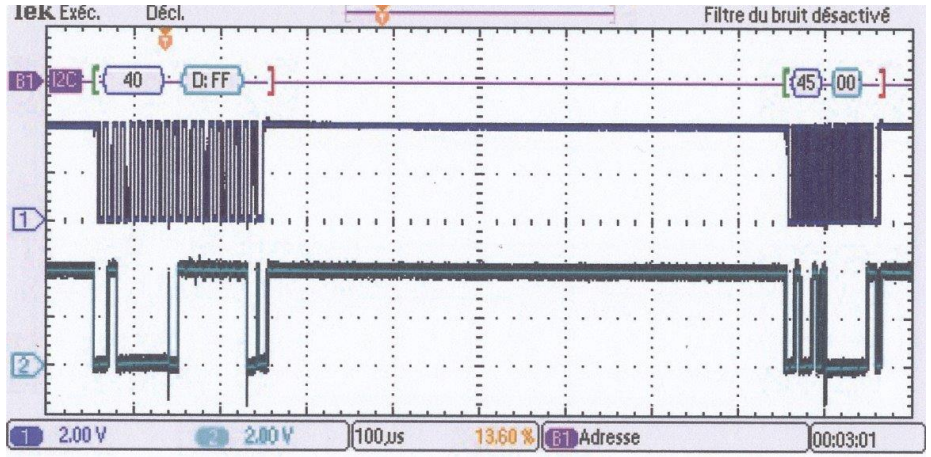
```
using System;
using ToolBoxes;

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 sortie SCL et SDA

         // Paramètres du bus I2C
         byte addLeds_I2C = 0x38; // Adresse (7 bits) du PCF8574A relié aux LED
         byte addBPs_I2C = 0x3F;  // Adresse (7 bits) du PCF8574A relié aux BP
         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);
         }
      }
   }
}
```

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

PCF8574
Classe

Méthodes
- PCF8574(ushort I2C_Add_7bits, short FreqBusI2C)
- Read() : byte
- Write(byte value) : void

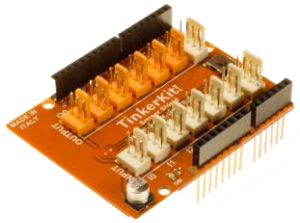
Page Web de la classe PCF8574 + Code de l'exemple



Modification de la fréquence du BUS I2C !

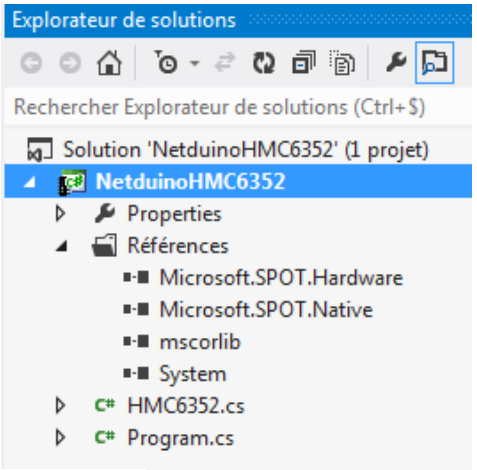
3.8. I2C : Lire la direction donnée par une boussole HMC6352

Code C# de l'exemple Com8



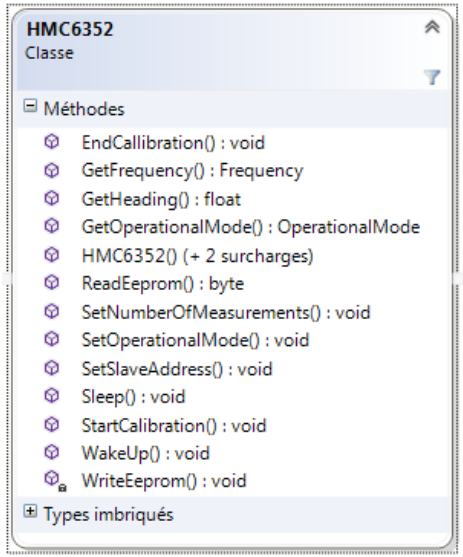
Shield TINKERKIT V2
sur Netduino plus 2

HMC6352
Sparkfun SEN-07915




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

using ToolBoxes;

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



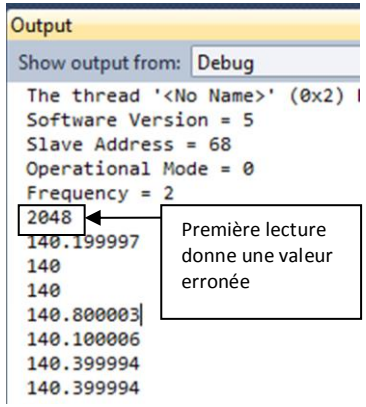
Page Web de la classe HMC6352 + Code de l'exemple

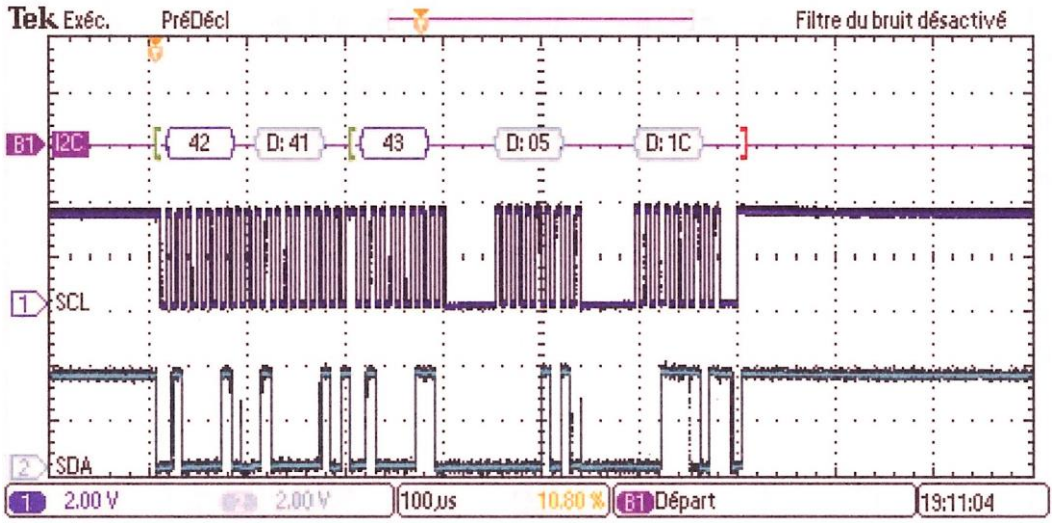
I2C : Lire la direction donnée par une boussole HMC6352 (suite)

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

                float heading = compass.GetHeading();

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

            }
        }
    }
}
```

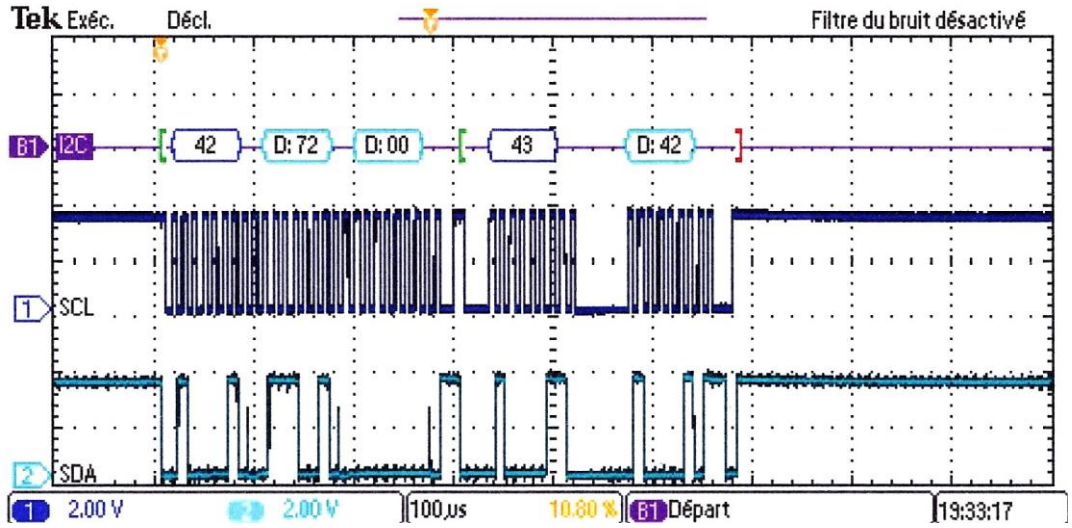
Utilisation du **debugger** pour visualiser la communication avec la boussole.

Pas à pas
F10 : Step Over
F11 : Step into





Le µC demande à lire l'angle mesuré par la boussole (commande **41h = 'A')**
La boussole répond $051C_{(16)} = 1308_{(10)} = 130,8°$

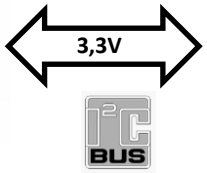


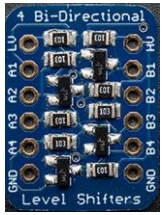
Le µC demande l'adresse de la boussole (commande **72h = 'r' 00h = 0)**
La boussole répond **42h : SLA+W. Son adresse SLA est donc 21h**

3.9. I2C : Mesurer la température ambiante avec un capteur TMP102

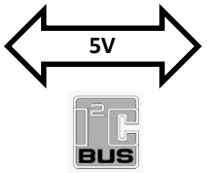
Code C# de l'exemple Com9

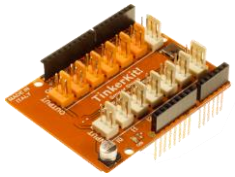
[TMP102](#)
Sparkfun [SEN-11931](#)





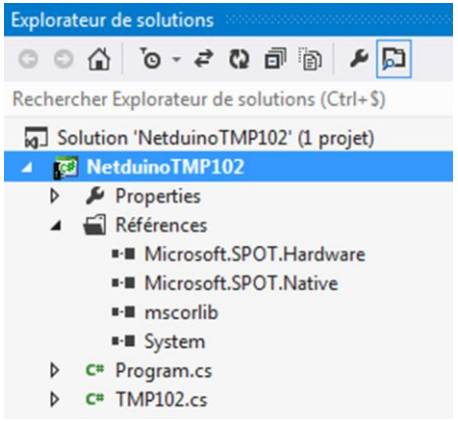
Convertisseur
5V <-> 3.3V
([Adafruit BSS138](#))



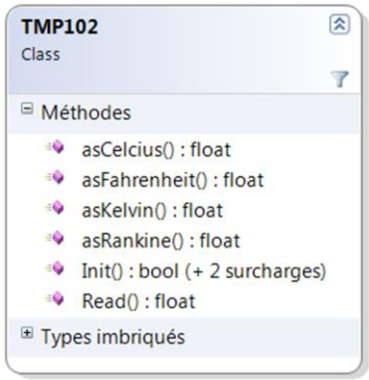


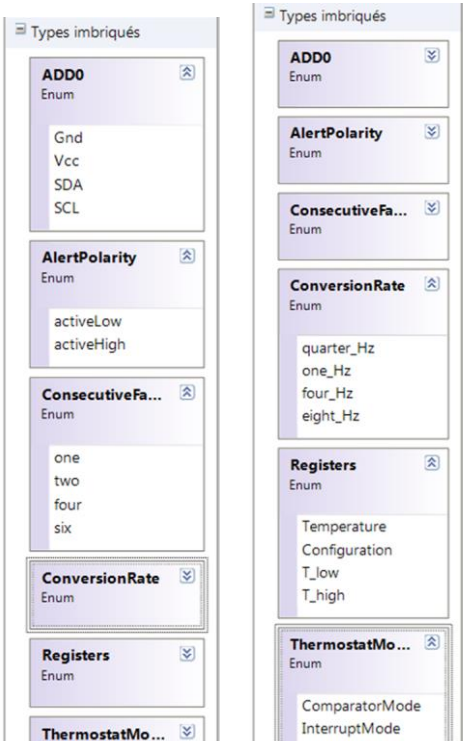
[Shield TINKERKIT V2](#)
sur Netduino plus 2

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



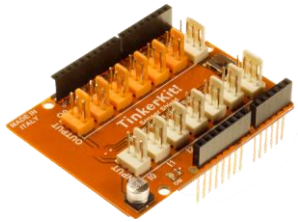

```
using System.Threading;
using Microsoft.SPOT;

using ToolBoxes;

namespace TestNetduinoTMP102
{
    public class Program
    {
        public static void Main()
        {
            TMP102 CPTTEMP102 = new TMP102();
            CPTTEMP102.Init(TMP102.ADD0.Gnd);

            while (true)
            {
                CPTTEMP102.Read();
                Debug.Print("Temperature: " + CPTTEMP102.asCelcius() + " C");

                // Sleep for 1000 milliseconds
                Thread.Sleep(1000);
            }
        }
    }
}
```





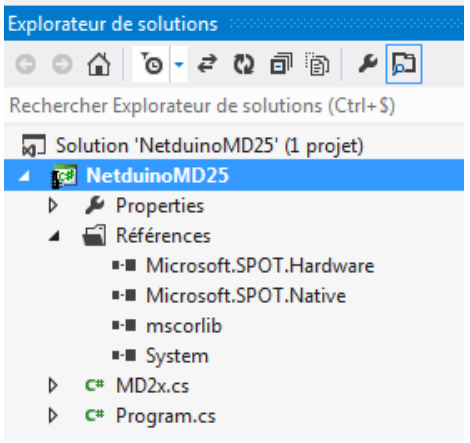
[Page Web de la classe Sensor + Code de l'exemple](#)

3.10. I2C : Commander deux motoréducteurs, équipés d'encodeurs, avec une carte Devantech MD25



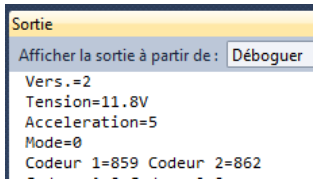
Shield TINKERKIT V2
sur Netduino plus 2

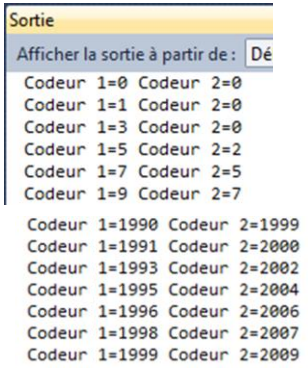
MD25 : Ensemble de pilotage pour moteurs "DCM2"



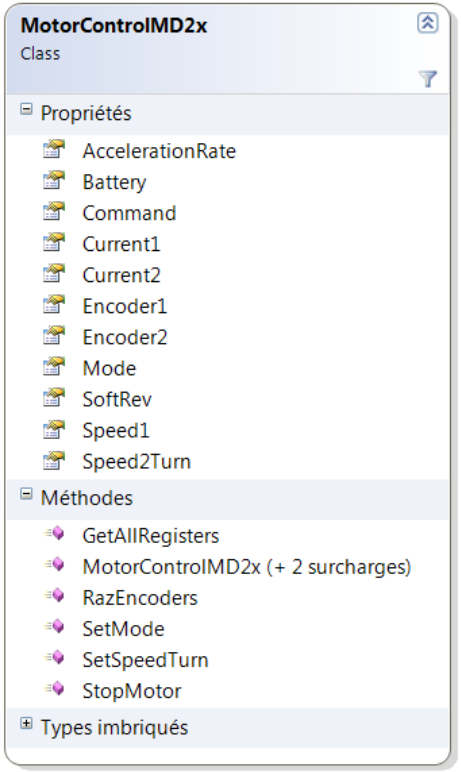
Code C# de l'exemple Com10

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

using MD2x;

namespace TestNetduinoMD25
{
    public class Program
    {
        public static void Main()
        {
            // Programme de test de la carte MD23/MD25
            // Création d'un objet MotorControl (carte MD23 ou MD25)
            // avec l'adresse 0x58 et la fréquence de bus F = 100kHz
            MotorControlMD2x CarteMD25 = new MotorControlMD2x();

            // 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);
            }
        }
    }
}
```

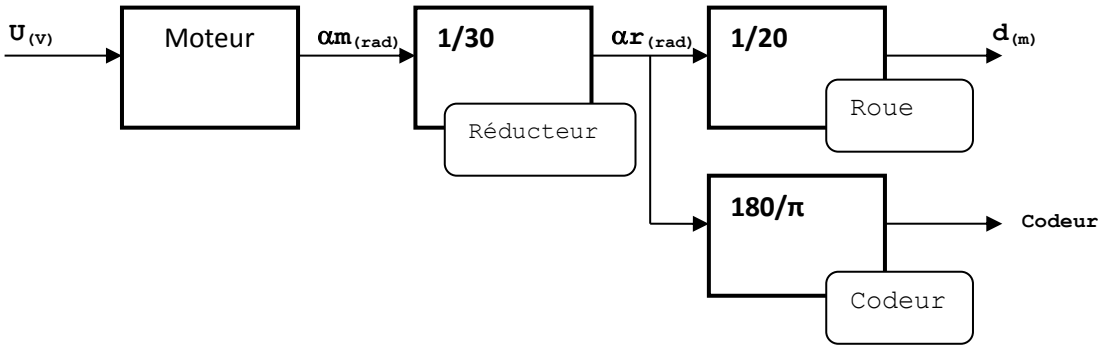




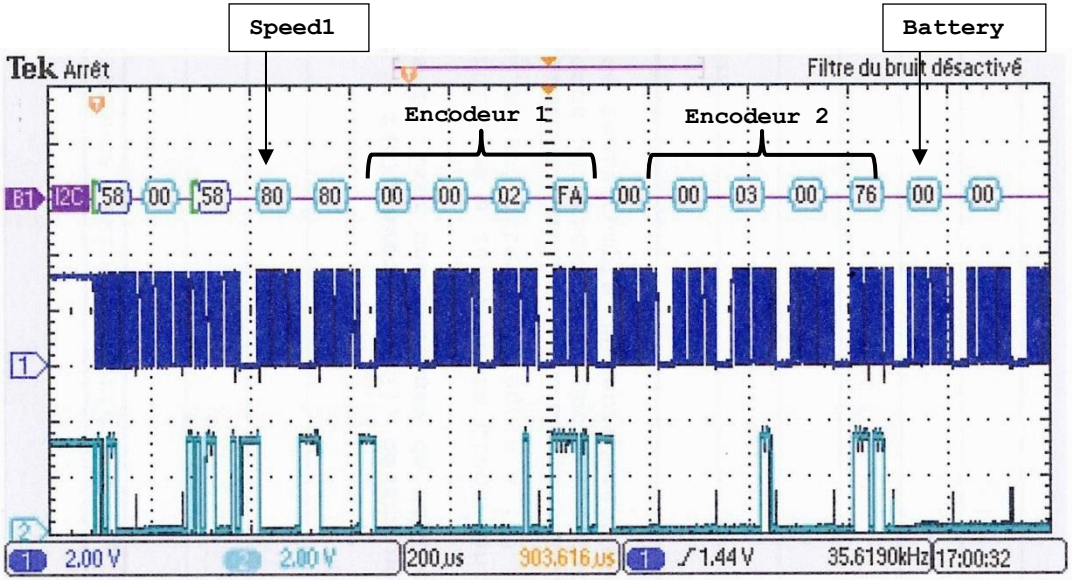
Page Web de la classe MD2x + Code de l'exemple



I2C : Commander deux motoréducteurs, équipés d'encodeurs, avec une carte Devantech MD25 (suite)



Lecture des registres de la carte MD23/MD25(partielle)

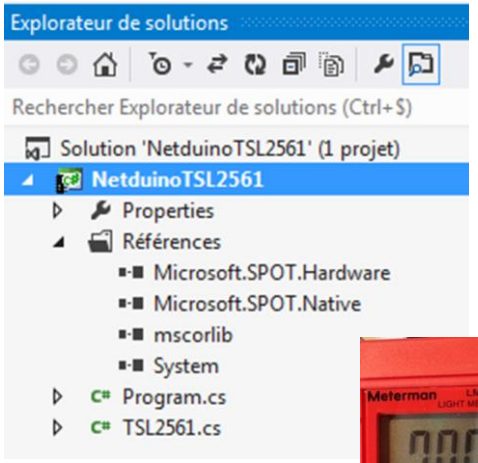


3.11. I2C : Mesurer la luminosité ambiante avec un capteur TSL2561

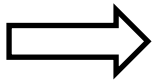
Code C# de l'exemple Com11

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

using ToolBoxes;

namespace TestNetduinoTSL2561
{
    public class Program
    {
        public static void Main()
        {   // Création d'un objet TSL2561
            // avec l'adresse 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;
            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);
            }
        }
    }
}
```

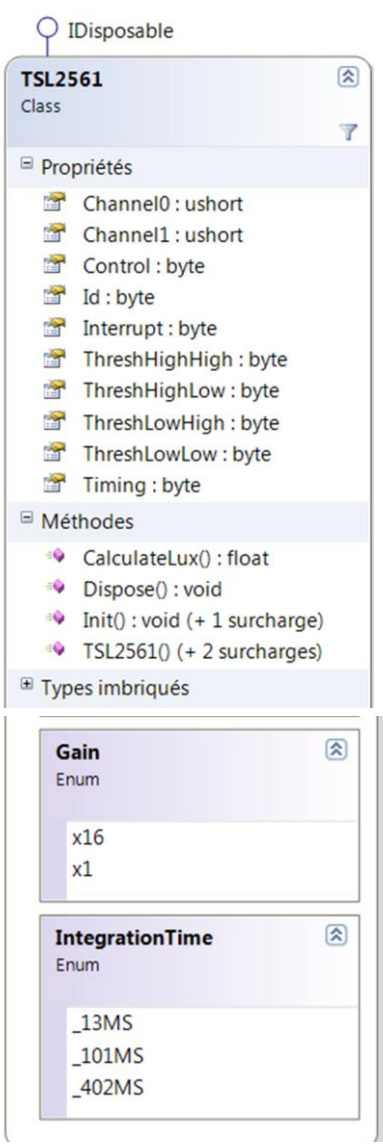
[TSL2561](#)
([Grove SEN-10171P](#))



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



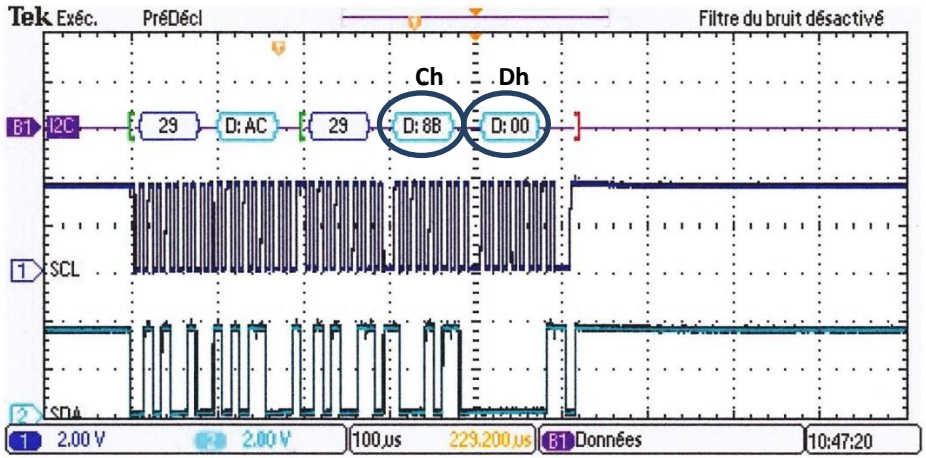
```
Lecture des registres
---------------------
00h Control             : 3
01h Timing              : 0
02h ThreshLowLow        : 16
03h ThreshLowhigh       : 32
04h ThreshhighLow       : 32
05h Threshhighhigh      : 64
06h Interrupt           : 0
0Ah Part number / Rev Id : 17
--------------------------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



Mesurer la luminosité ambiante avec un capteur TSL2561 (suite)

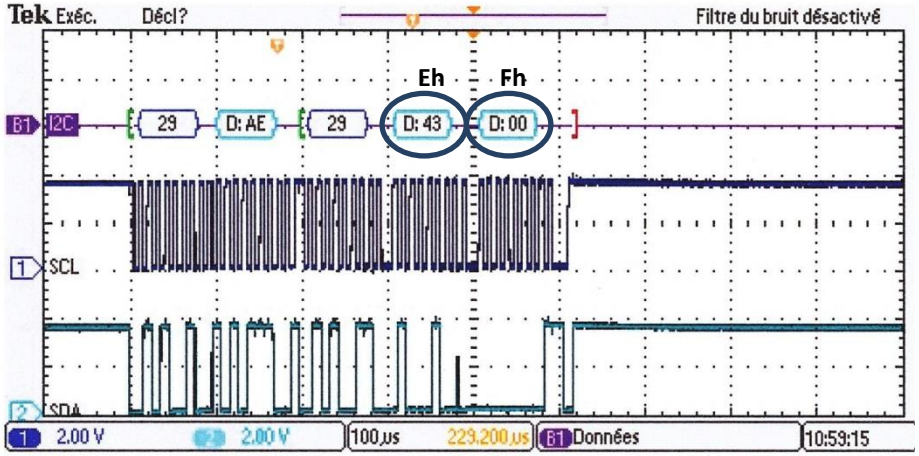
`Lecture du canal 0 en mode mot : registres Ch(`Low byte) **`et Dh(`**High byte)

Canal 0 = 008Bh



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

Canal 1 = 0043h



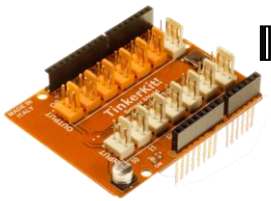
La méthode **calculateLux()** renvoie 420 lux pour ces valeurs des canaux.

3.12. Un fil : Mesurer la température ambiante avec un capteur DS18B20 (OneWire)

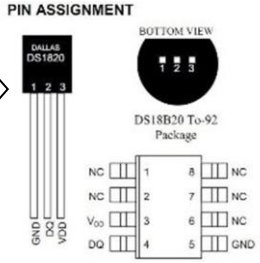
Code C# de l'exemple Com12

One Wire

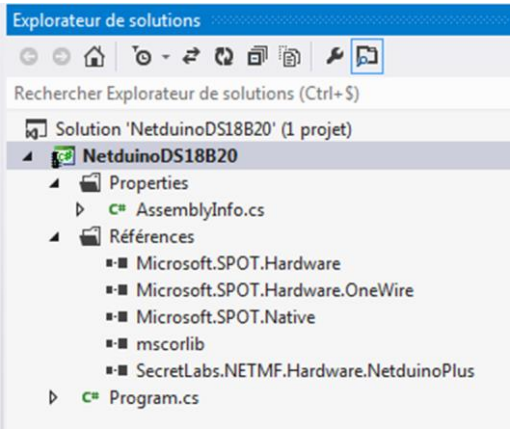
```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using SecretLabs.NETMF.Hardware.NetduinoPlus;

namespace TestNetduinoDS18B20
{
    public class Program
    {
        public static void Main()
        {
            // DS18B20 connecté sur O5 du shield Tinkerkit
            OutputPort Pin = new OutputPort(Pins.GPIO_PIN_D3,false);

            OneWire ow = new OneWire(Pin);
            ushort temperature;

            // read every second
            while (true)
            {
                if (ow.TouchReset()!=0)
                {
                    ow.WriteByte(0xCC);     // Skip ROM, we only have one device
                    ow.WriteByte(0x44);     // Start temperature conversion

                    while (ow.ReadByte() == 0) ;   // wait while busy

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

                    temperature = (byte)ow.ReadByte();          // LSB
                    temperature |= (ushort)(ow.ReadByte() << 8); // MSB

                    Debug.Print("Temperature: " + temperature / 16);
                    Thread.Sleep(1000);
                }else {Debug.Print("Device is not detected."); }
                Thread.Sleep(1000);
            }
        }
    }
}
```



Shield TINKERKIT V2



DS18B20
DFRobot



OneWire Class

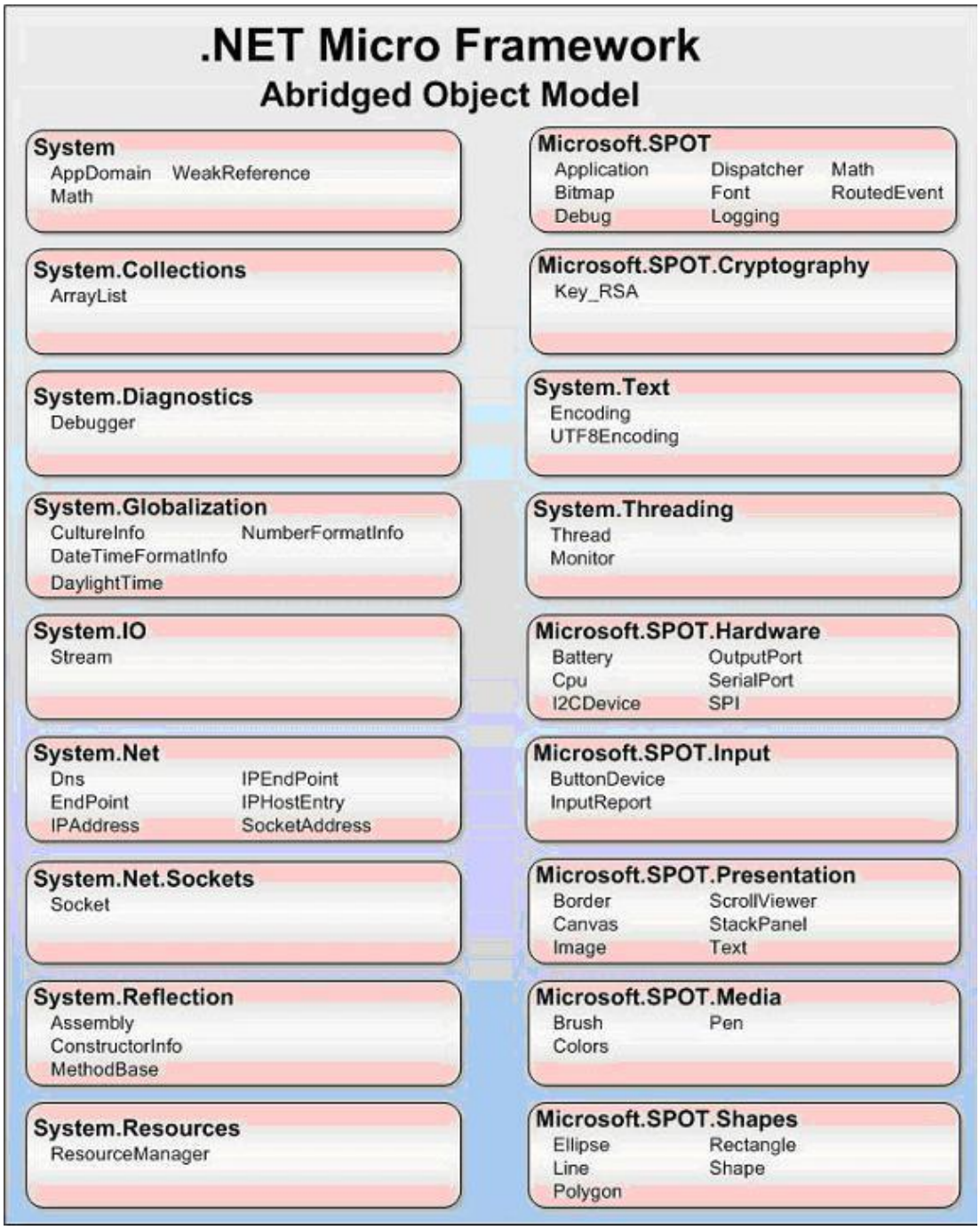
```
public OneWire(OutputPort port);

public int AcquireEx();
public ArrayList FindAllDevices();
public int FindFirstDevice(bool performResetBeforeSearch, bool searchWithAlarmCommand);
public int FindNextDevice(bool performResetBeforeSearch, bool searchWithAlarmCommand);
public int ReadByte();
public int Release();
public int SerialNum(byte[] SNum, bool read);
public int TouchBit(int sendbit);
public int TouchByte(int sendbyte);
public int TouchReset();
public int WriteByte(int sendbyte);
```

.net Micro Framework

Annexes

A1 - API Reference for .NET Micro Framework

http://msdn.microsoft.com/en-us/library/ee435793.aspx

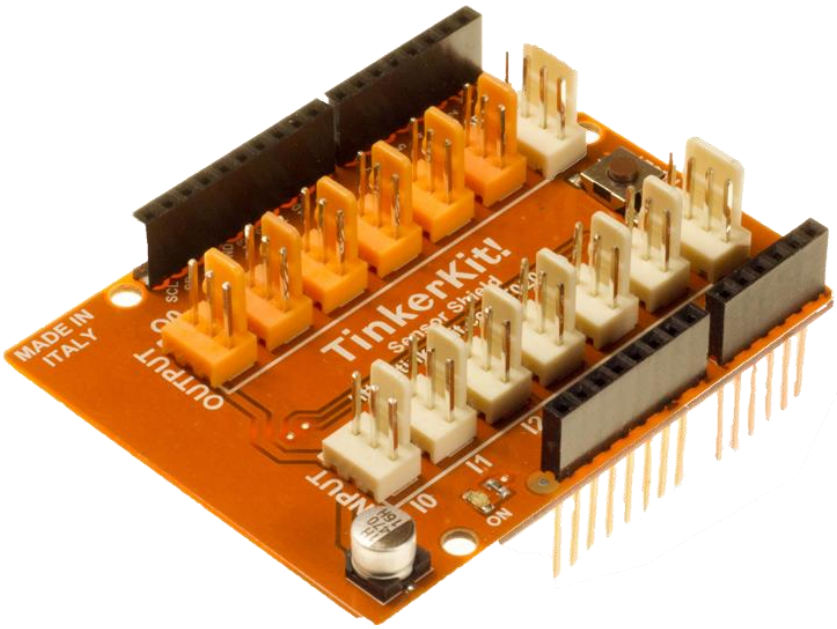


A2 – Les types reconnus par Microsoft Visual Studio et le .NET Microframework

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

A3 – 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.

Bibliographie

Livres

La programmation orientée objet Cours et Exercices en UML 2	Hugues Bersini	EYROLLES
C# 5 Développez des applications Windows avec Visual Studio 2012	Jérôme Hugon	ENI

Livres téléchargeables gratuitement sur internet

Visual Studio 2010	Microsoft
Expert .NET Micro Framework	E-Book
Getting_Started_with_the_Internet_of_Things	Make

Livres numériques payants

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

Matériels, documentation, liens pour le téléchargement des outils logiciels

NetDuino

http://netduino.com/

Projets, Communauté

NetDuino

http://forums.netduino.com/

Codeplex

https://www.codeplex.com/

GitHub

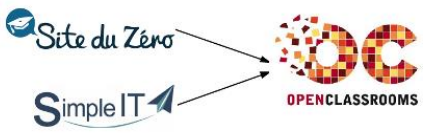
https://github.com/

Technologie .NET

Microsoft C# Express (téléchargement)

http://www.microsoft.com/france/visual-studio/essayez/express.aspx

Débuter en C#



http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-c-sur-net

Coach C#

http://msdn.microsoft.com/fr-fr/vstudio/bb409645.aspx

Centre de développement Visual C# (Ressources)

http://msdn.microsoft.com/fr-fr/vcsharp/aa336706

NETMF (Ressources, Téléchargement)

http://www.netmf.com/

Le Blog NETMF(Actualités)

http://blogs.msdn.com/b/netmfteam/

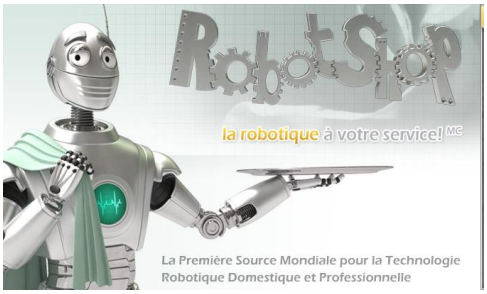
Distributeurs

Generation Robots
France
www.generationrobots.com
Telephone: +33 5 56 39 37 05

Lextronic
France
www.lextronic.fr
Telephone: 01-45-76-83-88

Mouser Electronics
http://www.mouser.fr/

Roboshop
http://www.robotshop.com/



Gotronic

http://www.gotronic.fr/

Index

ARDUMOTO 10
Boussole
  HMC6352 ..... *24*
Bouton-poussoir
  Led *6*
Codeur 10
COMFILE ELCD-162 15
Commander
  Led 6, 8, 9, 13
  Moteur pas à pas 7
  Motoréducteur 10, 27
  Servomoteur 12
EasyDriverStepperMotor V4.4 ..... *Voir* Moteur pas à pas
GHM-16 ..... *Voir* Motoréducteur
HMC6352 24
I2C
  Boussole 24
  Capteur de luminosité 29
  Capteur de température 26
  Carte de commande de deux moteurs CC 27
  Lcd 19
  Led 18
  Led + Bouton-poussoir 23
  Télémètre à ultrasons 20
Interruption
  Led + Bouton-poussoir 8
LED *5*
  Potentiomètre 13
MD25 27
Mesurer
  Direction 24
  Distance 20
  Luminosité 29
  Température 26, 31
Moteur pas à pas
  EasyDriver stepper Motor 7
Motoréducteur 10, 27
OneWire 31
  Mesure de température 31
PCF8574 ..... 18, 23
Potentiomètre 13
PWM ..... *ii*
  Codeur 10
  Led 9
  Servomoteur 12
Servomoteur 12
SRF08 20
TMP102 26
TSL2561 29
UART
  Lcd 15
  Sortie RS232 14
  XBEE ***16***
Un fil (One Wire)
  Mesure de température 31
XBEE ***16***