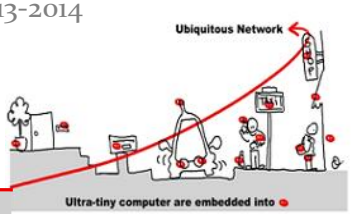
Tutorial : .Net Micro Framework et .Net Gadgeteer



1 Co-développement émulateur personnalisé et application pour une cible.

1.1 Utilisation d'un émulateur personnalisé

Après l'installation du SDK .Net Micro, dans le répertoire d'exemples, Framework (ex. C:\Users\Jean-Yves\Documents\Microsoft .NET Micro Framework 4.1\Samples\) ou dans le fichier « Temperature avec emulateur.zip », vous trouverez deux projets :

TemperatureEmulator

TemperatureSample

Le premier est une extension de l'émulateur de base et le second une application.

Chargez et exécutez le second sur l'émulateur standard. Comment pouvez-vous faire évoluer la température émulée ?

Charger et exécuter le premier projet, que se passe-t'il ? Pourquoi ?

Si maintenant vous revenez dans le premier projet et dans ses propriétés, quelles sont les émulateurs que vous pouvez utiliser maintenant ? Essayer avec l'émulateur « TemperatureEmulator » et testez la différence.

1.2 Développement d'un émulateur personnalisé

Créer un émulateur personnalisé pas à pas en suivant le tutorial de Microsoft : http://msdn.microsoft.com/en-us/library/cc532994.aspx

2 Annexe :

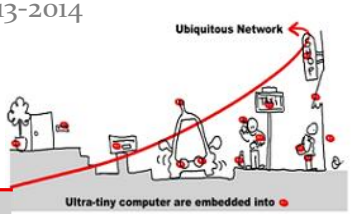
Creating A Custom Emulator: A Step-by-Step Example
In this walkthrough, you will create an emulator for a hypothetical massage chair unit, as well as a test application that runs on the emulator.

The following steps for creating a custom emulator are described.

Creating the emulator project using the Emulator template in Visual Studio.
Defining emulator components in code.

tigli@polytech.unice.fr
Université de Nice – Sophia Antipolis – Polytech'Nice Sophia – Département Science Informatique
930, Route des Colles – B.P. 145 - 06903 Sophia Antipolis Cedex – France

Tutorial : .Net Micro Framework et .Net Gadgeteer



Defining emulator components in an XML File.
Writing the application software that will run on the emulator.

2.1 Hardware Description

The device you will emulate is a massage chair unit connected to a main board. The operation of the massage chair is controlled by a number of GPIO pins listed in the following table. The main mechanical part of the motor consists of two motors. The massaging unit can be moved up and down by the motor controlled by the signals UD_ONOFF and UD_FFREW, while the signal M_ONOFF_L, M_ONOFF_H, and M_FFREW control the motor in charge of massaging.
In this walkthrough, you will focus on two output signals, UD_ONOFF and UD_FFREW on the massage chair, and two inputs for two buttons connected to the main board via pin 20 and 21. The application will move the massaging unit up when the Up button is pressed, and down when the Down button is pressed.

Description	Input/Output	Pin
Up / Down motor power	Output	61
Up / Down motor direction select	Output	63
Up Button	Input	20
Down Button	Input	21

2.2 To Create the Emulator Project

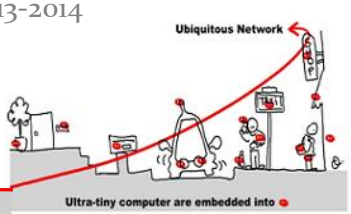
On the **File** menu in Visual Studio, point to **New** and then click **Project**. The **New Project** dialog box appears.

1. In the **Project types** tree, expand the **Visual C#** node if it is not already expanded.
2. In the expanded node, click **Micro Framework**.
3. In the **Templates** pane, choose the **Device Emulator** templates.
4. In the **Name** box near the bottom of the dialog box, type a name for your project. For this example, name your project MassageChairEmulator.
5. In the **Location** box, specify the folder that will contain your project, and then click **OK**.

The project you created includes the following files:

- `Program.cs` – Contains the entry point of the emulator, and the **`Program`** class that extends the base **`Emulator`** class.
- `Form1.cs` (and `Form1.Designer.cs`) – Contains the WinForm user interface for the emulator.
- `Emulator.config` – the default XML configuration file for the emulator.

Tutorial : .Net Micro Framework et .Net Gadgeteer



2.3 To Declare Input Pins in the Emulator Code

Open `Program.cs` by double-clicking on **Program.cs** in **Solution Explorer**.
Add the following `using` statement to the top of the file:

```
using Microsoft.SPOT.Emulator.Gpio;
```

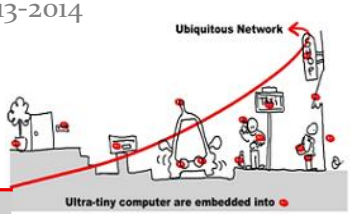
Find the method **SetupComponent**, and replace it with the following code:

```
public override void SetupComponent()
{
    this.GpioPorts.MaxPorts = 128;

    GpioPort motorUp = new GpioPort();
    motorUp.ComponentId = "MotorUpButton";
    motorUp.Pin = (Microsoft.SPOT.Hardware.Cpu.Pin)20;
    motorUp.ModesAllowed = GpioPortMode.InputPort;
    motorUp.ModesExpected = GpioPortMode.InputPort;

    GpioPort motorDown = new GpioPort();
    motorDown.ComponentId = "MotorDownButton";
    motorDown.Pin = (Microsoft.SPOT.Hardware.Cpu.Pin)21;
    motorDown.ModesAllowed = GpioPortMode.InputPort;
    motorDown.ModesExpected = GpioPortMode.InputPort;

    this.RegisterComponent(motorUp);
    this.RegisterComponent(motorDown);

    base.SetupComponent();
}
```

2.4 To Create User Interface for Input Pins

Create two buttons in the WinForm user interface, one for each of the input ports we created in the last step.
Open the designer view of `Form1.cs` by double-clicking on **Form1.cs** in the Solution Explorer sidebar. The tab should read **Form1.cs [Design]**.
In the **Toolbox** sidebar, under **Common Controls**, select **Button** control. If the **Toolbox** sidebar is not present, select **View | Toolbox**
In the form control, draw two buttons, one one above the other.
Rename the buttons and change the text on them.

Tutorial : .Net Micro Framework et .Net Gadgeteer



Click on the top button. In the **Properties** sidebar, change the **Name** property to motorUpButton, and change the **Text** property to Up.
Click on the other button. In the **Properties** sidebar, change the **Name** property to motorDownButton, and change the **Text** property to Down.
Generate the stub methods for the **OnMouseUp** and **OnMouseDown** events.
Click on the top button. In the **Properties** sidebar, click on the **Events** button.
Double-click on the **MouseUp** event. A method named **motorUpButton_MouseUp** should be generated.
Go back to the designer view of `Form1.cs`.
Double-click on the **MouseDown event** in the **Properties** sidebar. A method named `motorUpButton_MouseDown` should be generated.
Repeat the previous four steps on the button named MotorDownButton.

Populate the event handlers to send signals to the input ports.
Open the code view of `Form1.cs`.

Add the following statements to the top of the file:

```
using Microsoft.SPOT.Emulator;
using Microsoft.SPOT.Emulator.Gpio;
```

Add the following private member variables to the class **Form1**.

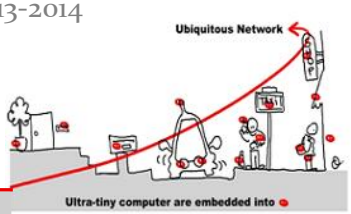
```
Emulator _emulator;
GpioPort _motorUpButtonPort, _motorDownButtonPort;
```

Modify the constructor of **Form1** to retrieve the **GpioPort** components, as follows:

```
public Form1(Emulator emulator)
{
    _emulator = emulator;

    _motorUpButtonPort =
        _emulator.FindComponentById("MotorUpButton") as GpioPort;
    _motorDownButtonPort =
        _emulator.FindComponentById("MotorDownButton") as GpioPort;

    InitializeComponent();
}
```

tigli@polytech.unice.fr
Université de Nice – Sophia Antipolis – Polytech'Nice Sophia – Département Science Informatique
930, Route des Colles – B.P. 145 - 06903 Sophia Antipolis Cedex – France

Tutorial : .Net Micro Framework et .Net Gadgeteer



In `Form1.cs`, create a helper function, **GpioPortSafeWrite**, along with a delegate, **GpioPortWriteDelegate**, to marshal the **GpioPort.Write** calls to the system thread. Add the following code to the **Form1** class.

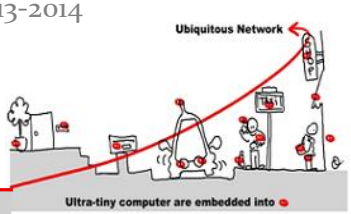
```
delegate void GpioPortWriteDelegate(bool state);

private void GpioPortSafeWrite(GpioPort port, bool value)
{
    port.Invoke(new GpioPortWriteDelegate(port.Write), value);
}
```

Modify the event handlers in `Form1.cs` to match the following code, so that they respond to the button clicks by changing the values of the input ports.

```
private void motorUpButton_MouseDown(object sender, MouseEventArgs e)
{
    GpioPortSafeWrite(_motorUpButtonPort, true);
}

private void motorUpButton_MouseUp(object sender, MouseEventArgs e)
{
    GpioPortSafeWrite(_motorUpButtonPort, false);
}

private void motorDownButton_MouseDown(object sender, MouseEventArgs e)
{
    GpioPortSafeWrite(_motorDownButtonPort, true);
}

private void motorDownButton_MouseUp(object sender, MouseEventArgs e)
{
    GpioPortSafeWrite(_motorDownButtonPort, false);
}
```

Build and run the emulator.
Build the solution by pressing the **F6** key or by selecting **Build | Build Solution**. As part of the build process, Visual Studio will register this new emulator with the system, so the Micro Framework applications will recognize this application as an emulator.
Run the emulator by pressing the **F5** key or by selecting **Debug | Start Debugging**. An error message containing the following text appears:

Tutorial : .Net Micro Framework et .Net Gadgeteer




```
Error: No assembly file is found. Please specify at least one exe, dll,
manifest, or pe file to load.
```

The emulator initialized successfully, but it failed as expected during the assembly loading stage of the start up process, since we have not yet created a Micro Framework application to run on it.

2.5 To Create an Application to Run on the Emulator

So far, we have created a simple emulator with two GPIO input ports that are controlled by the two buttons on the UI. Next, we'll build a simple application that will interact with the input ports created by this simple emulator.
Create a new .NET Micro Framework project in the current solution.
Select **File | New | Project**
Under **Project types**, select **Micro Framework**
Under **Templates**, select **Console Application**
Choose the name **MassageChairApplication** for your project.
For the **Solution** drop down list, select **Add to Solution**
Click **OK**
Add a reference to **Microsoft.SPOT.Hardware**, so that the application can access the managed drivers, including GPIO.
Select **Project | Add Reference**.
In the **Add Reference** dialog box, under the **.NET** tab, select **Microsoft.SPOT.Hardware**
Click **OK**.
Create a simple program that prints messages in the debug output in when the value of either of the two input ports changes.
Open `Program.cs` by double-clicking the filename `Program.cs` under **MassageChairApplication** in Solution Explorer.

Add the following two **using** statements to the top of the file:

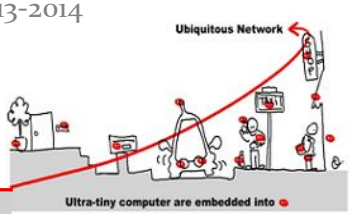
```
using Microsoft.SPOT.Hardware;
using System.Threading;
```

Place the following member variable declaration in the **Program** class.

```
InterruptPort _motorUpButton, _motorDownButton;
```

Add a constructor for the **Program** class. Inside the constructor, add code as follow to instantiate the 2 **InterruptPort** objects, one for each of the two input ports. In addition, hook up the event handlers.

Tutorial : .Net Micro Framework et .Net Gadgeteer




```
public Program()
{
  _motorUpButton = new InterruptPort((Cpu.Pin)20, false,
  Port.ResistorMode.PullDown,
  Port.InterruptMode.InterruptEdgeBoth);
  _motorDownButton = new InterruptPort((Cpu.Pin)21, false,
  Port.ResistorMode.PullDown,
  Port.InterruptMode.InterruptEdgeBoth);

  _motorUpButton.OnInterrupt +=
  new GPIOInterruptEventHandler(MotorUpButton_OnInterrupt);
  _motorDownButton.OnInterrupt +=
  new GPIOInterruptEventHandler(MotorDownButton_OnInterrupt);
}
```

Create the event handlers, and populate them to print debug messages, as follows:

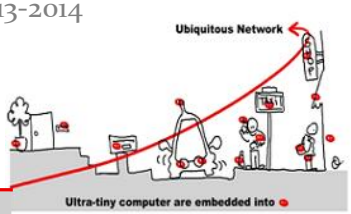
```
void MotorDownButton_OnInterrupt(Cpu.Pin port, bool state, TimeSpan time)
{
  Debug.Print("MotorDownButton " + ((state) ? "Down" : "Up"));
}

void MotorUpButton_OnInterrupt(Cpu.Pin port, bool state, TimeSpan time)
{
  Debug.Print("MotorUpButton " + ((state) ? "Down" : "Up"));
}
```

Change the **Main** method to instantiate a **Program** object, and then to sleep forever, as follows:

```
public static void Main()
{
  new Program();
  Thread.Sleep(-1);
}
```

Set up the Micro Framework application to deploy to the emulator that we created earlier.
Open the project properties page for the Micro Framework application by right-clicking on the project name (**MassageChairApplication**) and select **Properties** in **Solution Explorer**.
Click on the **Micro Framework** tab.
In the **Transport** drop-down box, select **Emulator**.
In the **Device** drop-down box, select the name of the emulator.

Tutorial : .Net Micro Framework et .Net Gadgeteer



Run the Micro Framework application under the debugger.
Set the Micro Framework application as the startup project by right-clicking on the project name (**MassageChairApplication**) and select **Set as StartUp Project** in **Solution Explorer**.
Start the debugger by pressing the **F5** key or select **Debug | Start Debugging**. The emulator starts.
Inside Visual Studio, open the Output window by selecting **View | Output**
Click on the two buttons in the emulator and look for the debug messages in the output window. Since the debugger is attached to the Micro Framework application, we can also place breakpoints in the code and step through the program.
Close the debugger by either closing the emulator, or by pressing the stop button on the **Debug** toolbar.
Add the two output ports that control the motor to the application, and update the interrupt handlers to produce the correct outputs.
Open **Program.cs** in the Micro Framework application by double-clicking **Program.cs** under **MassageChairApplication** in **Solution Explorer**.
Place the following member variable declaration in class **Program**.

```
OutputPort _motorOnOff, _motorDirection;
```

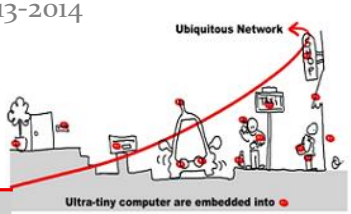
Inside the constructor of the class **Program**, add the following code to instantiate the two **OutputPort** objects, one for each of the two output ports. Set them to **false** initially.

```
_motorOnOff = new OutputPort((Cpu.Pin)61, false);
_motorDirection = new OutputPort((Cpu.Pin)63, false);
```

Update the event handlers, so when the up button is pressed, the motor goes up, and the motor goes down when the down button is pressed. The code is as follows:

```
void MotorDownButton_OnInterrupt(Cpu.Pin port, bool state, TimeSpan time)
{
  if (state == true)
  {
    _motorDirection.Write(false);
    _motorOnOff.Write(true);
  }
  else
  {
    _motorOnOff.Write(false);
  }
}

void MotorUpButton_OnInterrupt(Cpu.Pin port, bool state, TimeSpan time)
{
  if (state == true)
  {
```

Tutorial : .Net Micro Framework et .Net Gadgeteer




```
    _motorDirection.Write(true);
    _motorOnOff.Write(true);
  }
  else
  {
    _motorOnOff.Write(false);
  }
}
```

Build the solution by pressing the **F6** key or by selecting **Build | Build Solution**.

2.6 Adding Output Pins to the Emulator using Emulator.Config

In this step, we will return to the emulator, and create two GPIO output pins on GPIO port 61 and 63, one for each of the two buttons we will be creating to control the motor.
Open `Emulator.config` under the emulator project by double clicking on **Emulator.config** in **Solution Explorer**.

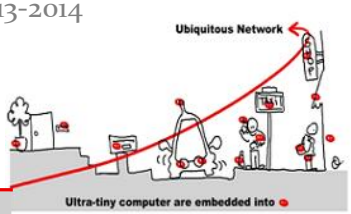
Create two GPIO output pins on port 61 and 63 by modifying the <Types> node and the <EmulatorComponents> node as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<Emulator>
  <Types>
    <GpioPort>Microsoft.SPOT.Emulator.Gpio.GpioPort</GpioPort>
  </Types>
  <EmulatorComponents>
    <GpioPort id="MotorOnOff">
      <Pin>61</Pin>
      <ModesAllowed>OutputPort</ModesAllowed>
      <ModesExpected>OutputPort</ModesExpected>
    </GpioPort>
    <GpioPort id="MotorDirection">
      <Pin>63</Pin>
      <ModesAllowed>OutputPort</ModesAllowed>
      <ModesExpected>OutputPort</ModesExpected>
    </GpioPort>
  </EmulatorComponents>
</Emulator>
```

2.7 Connecting a Scroll Bar to the Output Pins

The following steps hook up the output pins defined in the previous step to a scroll bar control.
Create a **VScrollBar** control in the WinForm user interface to represent the motor position.

Tutorial : .Net Micro Framework et .Net Gadgeteer



Open the designer view of `Form1.cs` by double-clicking on **Form1.cs** in **Solution Explorer**. The tab should read **Form1.cs [Design]**.
In the **Toolbox**, under **All Windows Forms**, select **VScrollBar** control. If the **Toolbox** sidebar is not visible, select **View | Toolbox**.
In the form control, draw a vertical scroll bar.
Rename the **VScrollBar** to **motorPosScrollBar**.
Change the **LargeChange** and **SmallChange** properties to 0, to prevent user scrolling.
Add code to listen to the newly created output port, and move the vertical scroll bar up and down accordingly to represent the motor position.
Right-click on **Form1.cs** in **Solution Explorer**, and select **View Code**, to open the code view of `Form1.cs`.
In class **Form1**, add the following member variables:

```
GpioPort _motorOnOff, _motorDirection;
System.Timers.Timer _motorTimer;
```

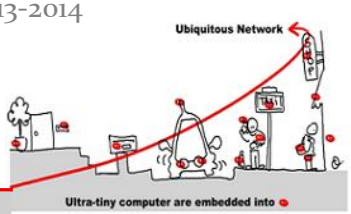
The timer will be used to periodically move the motor when the `_motorOnOff` signal is continuously on.
Add the following code in the constructor of the **Form1** class to retrieve the output ports and create the timer object. In addition, register an event handler, **_motorOnOff_OnGpioActivity**, for the **GpioActivity** event on the **_motorOnOff** output port.

```
_motorOnOff = _emulator.FindComponentById("MotorOnOff") as GpioPort;
_motorDirection =
_emulator.FindComponentById("MotorDirection") as GpioPort;

_motorOnOff.OnGpioActivity +=
new GpioActivity(_motorOnOff_OnGpioActivity);

_motorTimer = new System.Timers.Timer();
_motorTimer.Interval = 100;
_motorTimer.Elapsed +=
new System.Timers.ElapsedEventHandler(_motorTimer_Elapsed);
```

The timer callback function, **_motorTimer_Elapsed**, which we'll create next, will move the motor position based on the signal on **_motorDirection**.
Create a helper function, **MoveMotor**, to move the motor position on the vertical scroll bar based on the direction. Note that since the **MoveMotor** function will always get called on a non-UI thread, we would need to call the **Invoke** method on the control to change the value. Also, create the event handler for the timer, **_motorTimer_Elapsed**, to call **MoveMotor**. The code is as follows:

```
delegate void SetMotorPosDelegate(int value);
```

Tutorial : .Net Micro Framework et .Net Gadgeteer



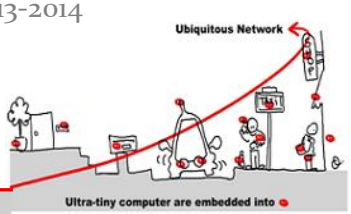

```
void SetMotorPos(int value)
{
  motorPosScrollBar.Value = value;
}

void MoveMotor(bool direction)
{
  // if the direction is up (true), we need to decrement, as the
  // vscrollbar value increases when it goes down
  int increment = (direction) ? -1 : 1;
  int newValue = motorPosScrollBar.Value + increment;

  if (newValue <= motorPosScrollBar.Maximum &&
  newValue >= motorPosScrollBar.Minimum)
  {
    motorPosScrollBar.Invoke(
    new SetMotorPosDelegate(SetMotorPos), newValue);
  }
}

void _motorTimer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
  MoveMotor(_motorDirection.Read());
}
```

Add the event handler, **_motorOnOff_OnGpioActivity**, to turn the timer on and off, as follows.

```
void _motorOnOff_OnGpioActivity(GpioPort sender, bool edge)
{
  if (edge)
  {
    MoveMotor(_motorDirection.Read());
    _motorTimer.Start();
  }
  else
  {
    _motorTimer.Stop();
  }
}
```

tigli@polytech.unice.fr
Université de Nice – Sophia Antipolis – Polytech'Nice Sophia – Département Science Informatique
930, Route des Colles – B.P. 145 - 06903 Sophia Antipolis Cedex – France

Tutorial : .Net Micro Framework et .Net Gadgeteer



Build the solution by pressing **F6** or by selecting **Build | Build Solution**.
Run the emulator under the debugger.
Set the startup project back to the emulator by right-clicking on the emulator project name (**MassageChairEmulator**) and select **Set as StartUp Project** in **Solution Explorer**.
Open up the project properties page by right-clicking on the project name (**MassageChairEmulator**) and select **Properties** in **Solution Explorer**.
Click on the **Debug** tab.
Set the **Command line arguments** field to the path to the .NET Micro Framework application's executable. Include quotation marks if the path has spaces. The path will normally be in the following form, with `Username` replaced by your Windows user name:

```
"C:\Documents and Settings\Username\My Documents\Visual Studio 2008\Projects\MassageChairEmulator\MassageChairApplication\bin\Debug\MassageChairApplication.exe"
```

This will let the emulator know to load our .NET Micro Framework application when it runs.
Run the emulator by pressing the **F5** key or by selecting **Debug | Start Debugging**.
Press and hold the **Up** and **Down** buttons, and see the motor position move.

tigli@polytech.unice.fr
Université de Nice – Sophia Antipolis – Polytech'Nice Sophia – Département Science Informatique
930, Route des Colles – B.P. 145 - 06903 Sophia Antipolis Cedex – France