Dot NET: Embedded Development for the Rest of Us

John Leier, Digi International

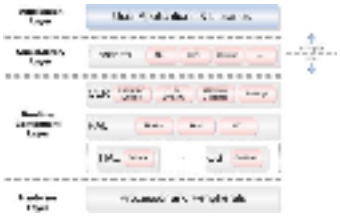


Figure 1. The essential layers in an application that relies on the .NET Micro Framework. Programmers usually work at the User Applications level. Courtesy of Microsoft (click to enlarge)

On a recent flight I talked with a software-engineering manager about the challenges of finding good embedded-system developers. She told me she has a team of 12 developers, but only three had proficiency with drivers, board-support packages, and boot-loaders. The other nine -- all good application developers -- lacked low-level coding experience. I asked if she had heard of the Microsoft .NET Micro Framework.
Microsoft's .NET Micro Framework, in combination with the C# language and Microsoft's Visual Studio tools, gives engineers and programmers more options for their embedded-system designs. A product team can reduce its design time because engineers no longer implement a boot-loader, drivers, or other low-level code. The .NET Micro Framework lets engineers who are new to embedded-system design jump right into coding without the need for special tools and a detailed understanding of processor and operating-system architectures.

The .NET Micro Framework has its roots at Microsoft Research where computer scientists aimed to simplify the development of small, low-power designs. They believed future electronic devices would use 32-bit processors, batteries, networks, and new protocols, such as Z-Wave and ZigBee. To design the growing number of networked embedded devices, the pool of developers -- and their productivity -- would have to increase dramatically.

Figure 2. A Digi Connect ME module provides an Ethernet connection on one side and a serial port on the other, which makes it easy to add Ethernet-communications capabilities to embedded computers.

The Microsoft Research team wrote the .NET Micro Framework from scratch to address these concerns. The framework is not a cut-down version of older software or tools. Unlike Windows Embedded CE and other embedded operating systems, the .NET Micro Framework does not require a memory-management unit (MMU), so developers can use it on low-power and lower cost ARM7 processors, as well as ARM9, and Blackfin processors. The framework software requires only several hundred kbytes of RAM and Flash/ROM. By comparison, a managed-code "environment" for Windows Embedded CE needs about 10-12 Mbytes. The smaller amount of memory required for a device that relies on the .NET application programming interfaces (APIs) leads to a lower-cost product.
The .NET Micro Framework provides a C# managed-code environment for application programming. I find the C# language easy to learn and use, and it can raise productivity above that available from C or C++ programmers. A managed-code environment means developers do not have to track down memory overwrites and mishandled pointers because these problems do not exist. However, garbage collection within C# code takes place in a non-deterministic fashion, so the developer cannot create real-time applications to run within the .NET Micro Framework.

Microsoft has integrated the .NET Micro Framework into its Visual Studio 2005 package, so programmers familiar with this software can quickly start on a project. In February 2007, Microsoft released Version 2 of the .NET Micro Framework for general embedded-application development. Digi International, Freescale, Embedded Fusion, and other companies offer hardware compatible with .NET Micro Framework, so developers can get a quick start with useful designs. (See “Get a Hardware Head Start”.)

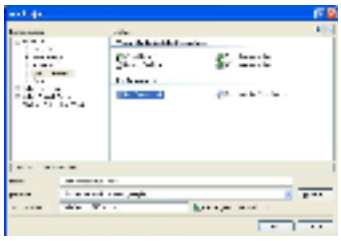
Microsoft first used the .NET Micro Framework in its Smart Personal Object Technology (SPOT) wristwatches. These watches -- based on an ARM7TDMI processor -- receive updates of news, sports, weather, and traffic on an FM sub-carrier signal. The SPOT-watch project provided a realistic test for the .NET Micro Framework, and it helped Microsoft's researchers improve the framework software.

The Dig
Kit for
Framew
templa
2005. Y
templa
a Conn

The architecture of the .NET Micro Framework comprises several pieces (Figure 1). A look at the layers from the top down shows what the framework provides for developers. Although the stack has many layers, developers find Microsoft's .NET Micro Framework easy to learn and use. At the top of the stack, the developer or user application and libraries can be found. Typically, embedded-system developers write programs that interact with the framework only at this level.

The .NET Micro Framework offers only a subset of features available in Microsoft's larger .NET Framework for standard desktop-PC and server applications. But the subset implements the functionality most applicable to embedded devices. This approach lets developers use existing code when practical, and it avoids the need for a larger base of embedded .NET Micro Framework code.

The Class Library Layer, often referred to as simply "libraries," contains .NET and Windows Presentation Foundation (WPF) libraries. The class library included with the .NET Micro Framework supplies an object-oriented collection of reusable classes you can use to develop embedded applications. The C# libraries include, among others, capabilities for encryption, graphics, and access to SPI and I^2C communication ports.



Screen 2. The Digi Connect ME JumpStart Kit for Microsoft’s .NET Micro Framework provides a template for Visual Studio 2005. You can select this template to start

programming a Connect ME module.

The Common Language Runtime (CLR) within the .NET Micro Framework provides the run-time environment needed by all application programs. The CLR manages memory, threading, code execution, garbage and exception handling, and other services. According to Microsoft, the CLR can perform approximately 15,000 calls/sec to managed methods in code written for an ARM7 processor with a 27.6 MHz clock frequency.

The next lower level provides a Program Abstraction Layer (PAL) that can control hardware, but that operates independent of hardware in an embedded system. The PAL provides a defined middle layer between the managed code interfaces and the hardware-specific code. At the lowest level, directly above the processor and peripherals, the framework provides a Hardware Abstraction Layer (HAL) or provisions for a compatible operating system. Thus, the framework can communicate directly with the underlying hardware or it can serve as a host to an operating system, perhaps a real-time operating system (RTOS), which provides hardware services and extensions to the .NET Micro Framework. This could let an application run real-time tasks in an RTOS thread, while it runs the entire .NET Micro Framework in a separate thread of the RTOS.
Hardware vendors such as Digi and Freescale supply the lower-level drivers and CLR implemented in C++. Developers' write application code in C# and link their code to the .NET Micro Framework "bootable" runtime. Because the framework provides a subset of the capabilities found in a complete operating system (OS), the framework does not require an overarching OS to manage an embedded system. Thus, developers refer to the .NET Micro Framework itself as a "bootable" runtime. In this case, bootable runtime refers to the .NET Micro Framework code that runs directly on the embedded hardware. That code provides boot-up support, interrupt handling, threading and process management, heap management, and other support functions that an OS would usually provide.[1]

A built-in hardware emulator for .NET Micro Framework offers developers a big advantage over other development tools. A hardware emulator lets a software team start to develop code right away, rather than wait until they have designed, built, and debugged prototype hardware. Users and vendors can use XML to extend the default emulator, so developers can create and debug an application without touching any hardware. Hardware vendors also can provide custom emulators that emulate their hardware, module, or processor. Of course, a large part of the fun is actually seeing an application run on real hardware.

Get to Working Code Quickly

The Connect ME module provides an example of a device you can program with the .NET Micro Framework. This embedded serial-to-Ethernet module includes an ARM7TDMI processor (55 MHz), 2 Mbytes of NOR Flash, 8 Mbytes of SDRAM, and an Ethernet PHY/MAC in a compact RJ-45 connector package (Figure 2).

A host operating system, based on the Express Logic ThreadX embedded kernel supports the .NET Micro Framework in the module. The kernel includes a complete IPv4 network stack available to application code as a sockets interface.

Applications written for the .NET Micro Framework have access to all of the hardware-independent APIs, as well as to the hardware classes that the HAL supports. The HAL for a Connect ME module supports TCP/IP sockets, RS-232 serial communication, and general-purpose I/O ports.

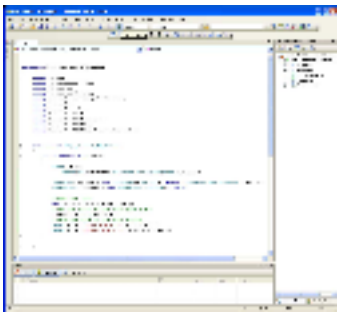
Try This at Home

Developers need only four steps to create and run an application that reads and writes to the general-purpose I/O ports. The steps that follow assume an Ethernet programming connection between a host development PC and the Connect ME module. In most cases, developers would use a Digi Connect ME Digi JumpStart Kit or similar baseboard for the Connect ME module:

Step 2. Double-click on Program.cs, the main module in the newly created project. Inside the C# Main() function, add the following lines:

```
InputPort MyInput = new
InputPort((Cpu.Pin)0,false,InputPort.ResistorMode.Disabled);

OutputPort MyOutput = new OutputPort((Cpu.Pin)1,false);
```



Screen 3. Code added to the Connect ME template lets software change and monitor the state of I/O pins on the module (click to enlarge)

Within Visual Studio 2005, the online help information for the InputPort class describes all the parameters. The first parameter -- (Cpu.Pin)0 -- specifies the first general-purpose I/O pin on the Connect ME module. As implemented by Microsoft, the second parameter -- false -- calls a glitch filter that can smooth spikes on input state changes. The Connect ME module does not use this parameter in its HAL and

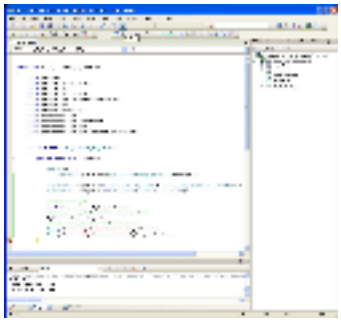
it will ignore any passed value. The last parameter -- InputPort.ResistorMode.Disabled -- lets the HAL configure inputs for one of three resistor modes; PullUp, PullDown, or Disabled. The Connect ME module also ignores this value, although the code example specifies Disabled.
The constructor for the OutputPort class only has two parameters: The GPIO pin to use -- (Cpu.Pin)1 -- and the initial state -- false -- for that output pin.

Step 3. To have the Connect ME module read the state of the input pin, simply assign it to a Boolean type.

```
bool InputState = MyInput.Read();
```

To write to the output, use the Write() method:

```
MyOutput.Write(true);
```



Screen 4. The Visual Studio debug window at the bottom of the screen displays the state of the I/O pins for developers (click to enlarge)

Step 4. To compile and deploy the application, select click on the "Start Debugging" button. The host PC will send your code to the flash memory on the Connect ME module via the Ethernet connection. Then the module will start your code from SDRAM. Debug output from your program comes from the module itself over an Ethernet debug connection to the Visual Studio Debug Output window.
The example shown illustrates all that is necessary to do to write, compile, deploy and debug a simple application that runs the .NET Micro Framework on an embedded target and manipulates GPIO pins. The .NET Micro Framework also offers an InterruptPort class, but the Connect ME module does not support this class through its I/O pins. Other hardware may provide an interrupt-input pin that the programmer can access through the InterruptPort class.

Similar classes, methods, and properties let developers easily control serial I/O ports and create socket servers and clients. The help documents within Visual Studio 2005 describe how to use commands so that even people without much C# experience can use them in an application.

To better understand how to apply the .NET Micro Framework, developers can use its emulator model at no charge, by downloading both the .NET Micro Framework Software Development Kit (SDK) and an evaluation version of Visual Studio 2005 from Microsoft. They also can purchase a development package such as the Digi Connect ME JumpStart kit (part number DC-ME-MF).

Reference

[1] Thompson, Daniel and Colin Miller, "Microsoft's .NET Micro Framework: Product Positioning and Technology Whitepaper:" http://download.microsoft.com/download/3/1/9/319f7469-70bd-4e7c-a6cc-e5ad96939af4/NET_Micro_Framework_Whitepaper_V_1.0.doc [1]

For further reading

Microsoft's .NET Micro Framework homepage: msdn.microsoft.com/embedded/netmf [2]

.NET Micro Framework SDK download - www.microsoft.com/downloads/details.aspx?familyid=32f5df20-6c95-4fe8-a76c-0ed56a839ad2&displaylang=en [3]

Trial versions of Visual Studio 2005 (The .NET Micro Framework requires the Standard, Professional, or Team version and will not work with Visual Studio Express.) www.microsoft.com/emea/msdn/visualstudio/enxu/getthetrials/ [4]

For a list of Microsoft's .NET Micro Framework Partners: msdn2.microsoft.com/en-us/embedded/bb267307.aspx [5]

Digi Connect ME: www.digiembedded.com [6]

John Leier is product manager of embedded software at Digi International. He has more than 15 years of software-engineering experience that include programming data-acquisition and control systems in C, Visual Basic, Delphi, and C#. For more information, contact Digi International, 11001 Bren Road East, Minnetonka, MN 55343; (877) 912-3444; www.digi.com [7].

Links:
[1] http://download.microsoft.com/download/3/1/9/319f7469-70bd-4e7c-a6cc-e5ad96939af4/NET_Micro_Framework_Whitepaper_V_1.0.doc
[2] http://msdn.microsoft.com/embedded/netmf
[3] http://www.microsoft.com/downloads/details.aspx?familyid=32f5df20-6c95-4fe8-a76c-0ed56a839ad2&displaylang=en

[4] http://www.microsoft.com/emea/msdn/visualstudio/enxu/getthetrials/
[5] http://msdn2.microsoft.com/en-us/embedded/bb267307.aspx
[6] http://www.digiembedded.com/
[7] http://www.digi.com/