

# Developers in the frame

Changing the embedded development model with Microsoft.NET Micro Framework. By **Mike Rohmoser**.

The development model for embedded devices is traditionally seen as being complex, with the need for specialised knowledge of a target's software and hardware design aspects. That approach brought prolonged design cycles and competing project timelines, high development costs and long time to market.

While it still holds true in some applications, there is also a range of embedded applications that can easily do without that approach. Given that almost everyone has to deal with engineering resource constraints, wouldn't it be ideal to engage the non embedded engineering software team and reuse existing code from the server/desktop side?

Today's constant push towards pervasive networked device collaboration also implies tight integration with the backend network infrastructure, which is typically built on software utilising non embedded desktop and server class systems. A trend to pull enterprise server and desktop application software components into the embedded sector is becoming visible across some platforms, but this is often at the expense of elevated hardware requirements. This is where the .NET Micro Framework provides a fresh perspective and a new embedded development option.

Introduced in early 2007, Microsoft's .NET Micro Framework is a lightweight implementation of the .NET Framework. It's a fraction of the size of the smallest managed configuration of Windows Embedded CE: a few hundred kilobytes instead of multiple megabytes. Even though it is compact and, by design, does not offer the complete Windows Embedded CE feature set, it scales from simple, single function devices up to powerful multifunction devices with state of the art user interfaces. The .NET Micro Framework, which is focused on the growing number of network enabled embedded devices with 32bit processors, was designed specifically to meet the requirements and extended capabilities of this



new generation of devices, and has been built from the ground up, instead of being a derivative of an existing Microsoft embedded platform.

From an internal architecture point of view, the .NET Micro Framework consists of the following layers:

- **Application layer**  
C# application using built-in and user libraries/services
- **Class Library Layer**  
Core subset of .NET libraries and application services
- **Runtime Component Layer**  
This comprises: Common Language Runtime (CLR), a runtime environment providing execution

engine, thread management, garbage collection, exception handling and other services; Platform Abstraction Layer (PAL), a hardware independent abstraction layer providing services to the CLR, including memory management, debugging and asynchronous procedure calls; and the Hardware Abstraction Layer (HAL), an interface between PAL and hardware or operating system providing access to hardware functions

- **Hardware Layer**  
Processor platform and integrated peripherals  
The CLR in the runtime layer interprets language independent intermediate code (Common Intermediate Language – CIL) generated by the Visual Studio compiler and handles aspects such

as type safety and garbage collection, including safeguards resolving very common application software problems caused by memory leaks and unsafe pointers.

While this level of code management results in non deterministic application behaviour, it also provides a robust and safe environment for embedded applications that do not require real time performance. The CLR concept is also independent from the application programming language and the underlying hardware, which makes the .NET Micro Framework a future proof platform with programming language flexibility and virtually unlimited porting capabilities.

The Framework offers a C# managed code environment and seamless integration into the Microsoft Visual Studio environment, including full on device debug support. This means developers that are already experienced with .NET Framework and Visual Studio can take advantage of their skills and design embedded applications without climbing a significant learning curve.

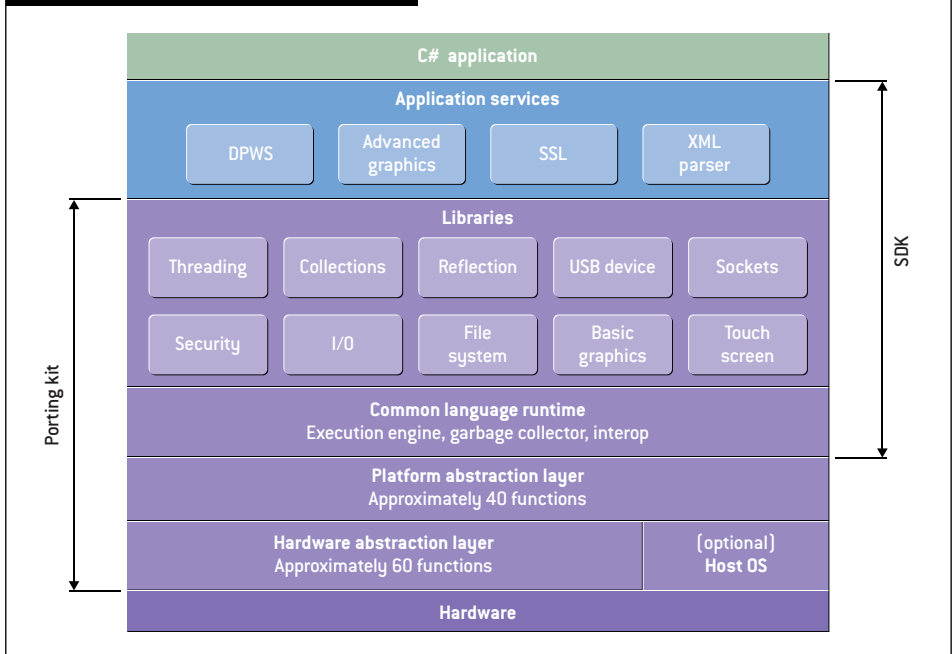
C# is powerful and easy to learn and allows developers to be productive quickly. The C# application development process is shielded from the low level design details of the hardware platform by simply using supplied .NET class libraries, application services such as Devices Profile for Web Services and hardware specific interface support.

An XML defined and extensible hardware emulator is also part of the development tools, which makes it possible to test and develop software without target hardware. If needed, managed drivers can be written in C# for components attached to device interfaces, given that they are already supported by the .NET Micro Framework libraries.

All this, and the fact that existing .NET code used in enterprise server and desktop applications is easily shared with a .NET Micro Framework application on an embedded device, reduces traditional design risks and accelerates the software development process.

While the software aspect is a key factor, the best software cannot live up to its expectations without the supporting hardware. Support is available from a variety of 32bit platforms, including ARM7, ARM9 and Analog Device's Blackfin. While the Microsoft Porting Kit allows customers to adapt .NET Micro Framework to their target hardware, porting .NET Micro Framework to custom hardware requires specific skill sets. A

Fig 1: NET micro framework 3.0 architecture



more efficient approach is the use of embedded processor modules, rather than discrete hardware.

Embedded processor modules provide a complete and functional system with processor, memory, and supporting circuitry, on a single component module design. Combining the benefits of the Microsoft .NET Micro Framework with off the shelf platforms facilitates rapid product development.

Modules change the hardware development model in the same way that .NET Micro Framework changes the software development model, with benefits such as:

- Shorter product design cycle and time to market
- Software design can start immediately on actual hardware
- Precertified module designs reduce hardware design risk
- Carrier board design is simplified, inexpensive and less likely to go through respins
- Single vendor support for hardware and software
- Optional migration path to meet high volume cost expectations and/or mechanical design considerations

Developers should look for module manufacturers with a strong .NET Micro Framework offering, including complete and easy-to-use development kits providing the Microsoft .NET Micro Framework SDK and full platform support, allowing immediate C# application

development. The ideal hardware solution is built on a manufacturer's own processor platform, which makes a potential future migration to a discrete design much easier from a final product integration and cost point of view.

An example is the Digi Connect ME, part of Digi's .NET Micro Framework product range. This embedded module integrates Digi's ARM7TDMI based NS7520 processor running at 55MHz, with 2Mbyte of flash, 8Mbyte of ram, a UART, GPIO and a 10/100Mbit Ethernet interface in an RJ-45 form factor. Complete development kits are available and these support the .NET Micro Framework 2.5 release.

Reducing designs risks and time to market, the approach creates a software development model complemented by a hardware development model of embedded modules. This powerful combination simplifies embedded development and makes it accessible to a new developer audience and class of devices.

**Author profile:**

Mike Rohrmoser is senior product manager for Digi International.

**ne** For further information on any subject visit: [www.newelectronics.co.uk](http://www.newelectronics.co.uk)