Is C Dead?

by Wendy Logan, National Instruments

The C language has been the choice for test and measurement applications for years. But with the popularity of object-oriented languages such as C++ and C#, some may wonder if C has a place in tomorrow's test systems. If you do have a significant amount of C code investment, it is important to understand not only the key C programming language strengths and weaknesses but also best practices when integrating C code into applications written in another language.

Introduced more than 35 years ago, the C language has been used in test and measurement, embedded systems, and device driver development for decades. Initially created to address code reuse challenges commonly associated with assembly languages, the C programming language was developed to be platform-independent by abstracting the hardware layer. Because C often is only one layer away from the hardware, it generally is accepted that C provides much of the speed and memory efficiency associated with assembly while increasing code portability and decreasing development time.

The Strengths of C

C programming language strengths include the following:

Powerful Low-Level Capability to Optimize Code

Because C often provides direct access to operating-system or hardware-specific function calls, you have a greater ability to control which specific operations and how many operations are performed at the machine level. Optimized compilers and the capability to call assembly code also help you control program size and execution speed.

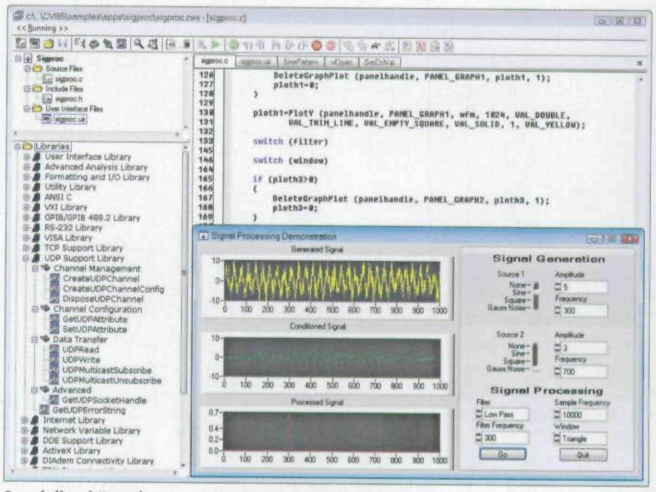
Highly Portable and an Extensive Set of Code Reuse Tools

You can compile a C program that adheres to a recognized industry standard, like ANSI C, and is hardware- and OS-independent for a wide variety of computer platforms and operating systems with minimal change to its source code. As a result, the language has amassed a sizable programming community and millions of lines of example code and hardware drivers that have been written in C.

General-Purpose and Industry-Specific Development Environments

Many general-purpose C development environments, such as Microsoft Visual Studio, provide additional out-of-the-box functionality, such as database connectivity APIs, in addition to standard C programming language functions. Industry-specific textual development environments like NI LabWindows/CVI also offer functionality commonly used in vertical application areas including extensive analysis libraries such as signal ramps, filters, and PID control algorithms as well as support for a diverse set of I/O device drivers.

With the capabilities to perform direct hardware manipulation, easily control how data is stored in memory, and reuse millions of lines of preexisting textual math code, the C language is well suited for developing applications with strict space and performance requirements or applications involving extensive signal processing and analysis. Specifically, device drivers, embedded systems, and real-time applications can benefit from the increased speed and processing available in C if the code is written efficiently.


```
        DeleteGraphPlot (panelhandle, PANEL_GRAPH1, ploth1, 1);
        ploth1=0;
    }

    ploth1=PlotY (panelhandle, PANEL_GRAPH1, wfm, 1024, VAL_DOUBLE,
          VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_YELLOW);

    switch (filter)

    switch (window)

    if (ploth3>0)
    {
        DeleteGraphPlot (panelhandle, PANEL_GRAPH2, ploth3, 1);
        ploth3=0;
    }
```

Specialized Development Environments

According to the VDC 2007 embedded systems market statistics report, C is used in more than 70% of embedded applications. But these same applications also are well served by graphical and object-oriented programming languages that directly address common C programming challenges.

The Challenges of C

C programming language challenges include the following:

Object-Oriented Programming

With complex applications, there usually is a requirement to group data together as a structure. This is a standard feature in most high-level languages including ANSI C.

Continued on page 46

As an extension, object-oriented languages offer support not only for creating custom data types but also for defining specific methods that work on that data. In particular, object-oriented programming features the concept of organizing an application into distinct modular components called objects. Programmers can encapsulate real-world test hardware, such as data acquisition devices and instruments, as objects.

Software objects model real-world entities, and like real-world objects, each software object is defined by a state and behavior. The state of the object is the set of variables you would use when describing the object. Likewise, objects have certain unique behaviors or actions they are capable of performing. As a result, once you create an object, you have a new data type on which distinct operations can be performed.

For example, a test station may be responsible for testing multiple components on a circuit board. In software, you can organize your application to model this behavior. You can create a class to represent a component and a method that is part of the class to test the component. You then can create a class to represent the circuit board, which is made up of an array of components. To test your board, you simply call the method responsible for testing each of the components that make up a board.

In addition to the base functionality of C, C++ contains some language extensions that make object-oriented programming more convenient. While C emphasizes simplicity, efficiency, and speed, C++ focuses more on abstraction. C++ does not force object-oriented design but allows for it if you deem it feasible. You can implement object-oriented programming in C as well but C++ makes it simpler and less error-prone.

Memory Management

With the low-level memory access available in C comes the responsibility for handling memory allocations and deallocations. Lack of clean-up code that explicitly deallocates memory can significantly influence the performance and determinism of an application. Along with other languages, .NET languages handle many low-level tasks such as memory management. You can allocate new memory on demand and then rely on the garbage collector to dispose of the memory when you no longer need it.

Parallel Execution

Multithreading was a key advance in the history of textual languages. With the capability to virtually execute multiple function calls at once, you can create rich user interfaces that seamlessly perform file I/O, database logging, and hardware control without seeing a performance decrease.

To realize this benefit, programmers use specific threading and timing libraries to arrange the scheduling of different sections of code. For instance, when developing an application that performs PID control while displaying a user interface, you must create at least two application threads to maintain the consistent execution timing required for PID control and allocate enough resources to maintain a smooth and responsive user interface. Also, you must be careful when accessing global data and be proficient in the use of specific multithreading constructs, such as semaphores and locks, to ensure data is not accessed from multiple threads at the same time.

There are various extensions to C/C++ and the Windows Software Development Kit (SDK) threading functions such as OpenMP and Intel Thread Building Blocks that simplify the process of creating parallel applications by introducing highly optimized constructs that allow data to efficiently scale across multiple processors. Other languages, like NI LabVIEW, automatically create multithreaded applications that can simplify the process of achieving maximum performance on multicore systems without the additional programming associated with thread management.

Because there are multiple programming languages that provide unique benefits to application development, it is no wonder that the automated test industry is moving toward open software architectures. As highlighted by the 2005 Frost & Sullivan survey *World Data Acquisition Board and Software Market*, it has become increasingly important that users be able to integrate software from multiple vendors. This is possible only through software standardization based on widely accepted technologies.

Reusing Existing C Code

There are many ways to integrate existing C code in external programming languages. The most common include the following:

Directly Compile and Link C Source Files

Most C++ compilers support directly calling C code from a C++ source file. First, you must ensure that the C and C++ compilers define basic types such as int, float, or pointer in the same way. The C++ language provides a linkage specification with which you declare that a function or object follows the program linkage conventions for a supported language. The extern keyword indicates to the C++ compiler that the following declaration is a C function.

```
// specifying_linkage.cpp

// Declare printf with C linkage.
extern "C" int printf( const char *fmt, ... );

// Cause everything in the specified header files
// to have C linkage.
extern "C"
{
 // add your #include statements here
 #include <stdio.h>
}

// Declare the two functions ShowChar and GetChar
// with C linkage.
extern "C"
{
 char ShowChar( char ch );
 char GetChar( void );
}
```

Continued on page 48

```
// Define the two functions ShowChar and GetChar
// with C linkage.
extern "C" char ShowChar( char ch )
{
  putchar( ch );
  return ch;
}

extern "C" char GetChar( void )
{
  char ch;
  ch = getchar();
  return ch;
}

// Declare a global variable, errno, with C linkage.
extern "C" int errno;

int main()
{
```

Call Precompiled C DLLs

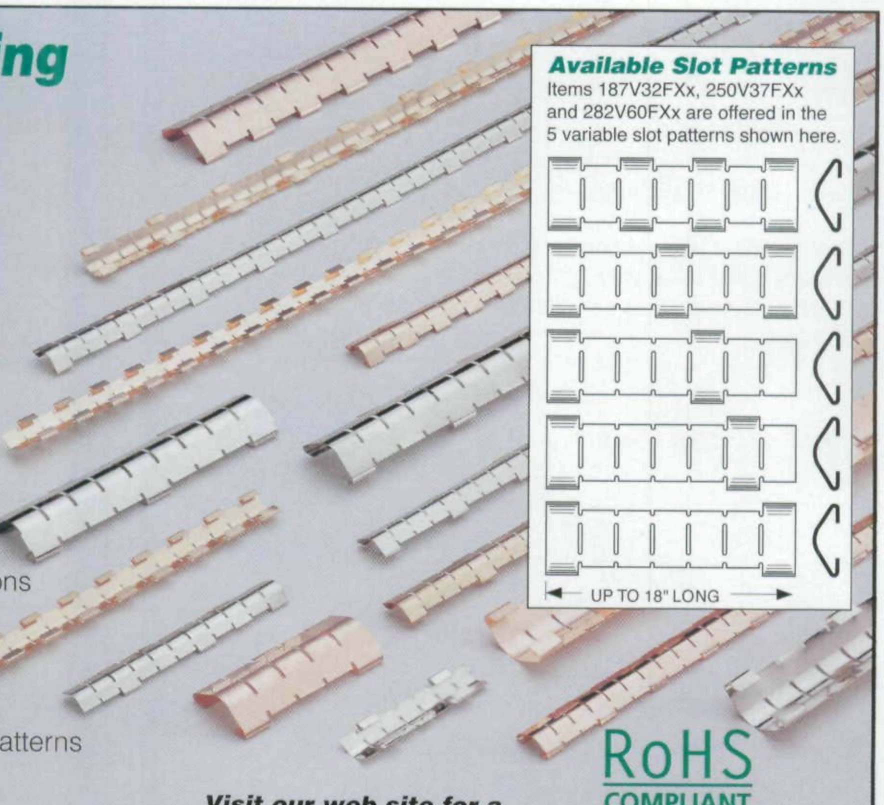
You also can create C code in a dynamic link library (DLL) and then call it from other languages that support DLLs. A DLL contains code and data that can be used by more than one program at the same time. For example, calling a DLL from .NET requires Platform Invoke (P/Invoke) to call unmanaged code from the managed world.

To reuse C functions, you need to wrap them with DllImport in C# or Declare Function in Visual Basic .NET and then ensure that all the .NET data types match the data types in the DLL. For example, character arrays become strings, C pointers become IntPtr, and so on.

```
[Visual Basic .NET]
Declare Function MessageBeep Lib
        "User32.dll" (ByVal beepType As Uint32) As Bollean
[C#]
[DllImport ("User32.dll")]
static extern bool MessageBeep (Uint32 beepType);
```

Although most, if not all, automated test applications require access to hardware, the managed world of .NET isolates you from the underlying hardware. The runtime engine of the .NET Framework, called the Common Language Runtime (CLR), acts as a buffer between your application and the actual memory on the host computer, preventing direct access to memory and registers.

To control devices such as GPIB boards, serial ports, and data acquisition devices, your program still needs direct memory access. This is a situation where using a C device driver in

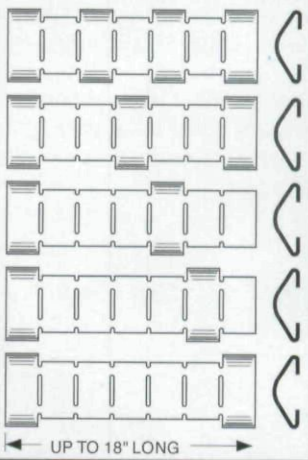


EMI/RFI Shielding

Variable Snap-on Gaskets

- Available in both slot and edge mount configurations for easy and secure mounting
- Fewer slots improve shielding effectiveness
- For gaps as small as .02"
- Ideal for bidirectional applications
- 100 db attenuation
- Low compression force
- High durability
- 3 standard profiles with 5 slot patterns
- In stock for immediate delivery

Available Slot Patterns
Items 187V32FXx, 250V37FXx and 282V60FXx are offered in the 5 variable slot patterns shown here.



RoHS COMPLIANT

Visit our web site for a FREE SAMPLE!
www.tech-etch.com

ISO 9001:2000 REGISTERED

Tech-Etch

TECH-ETCH, INC., 45 Aldrin Road, Plymouth, MA 02360 • TEL 508-747-0300 • FAX 508-746-9639

Visit www.rsleads.com/805ee-033

a .NET environment would be ideal. The C code would run in the low-level kernel mode of your OS and not in the user mode to which .NET applications are restricted.

Systems that allow engineers to select the best software tool for the job and integrate diverse code modules and technologies into one solution help maintain legacy code relevance while enabling development teams to take advantage of the latest software platforms. Implementing highly modular and interoperable systems leads to decreased cycle time, cross-functional support teams, and reusable components whether or not the components are developed in C or other common programming languages.

The Bright Future of C

The C programming language has enduring value especially for test applications because of its direct access to memory and minimal software overhead—important aspects lacking from most modern .NET applications. In addition, C gives developers the freedom to write code for a large number of platforms: everything from microcontrollers to advanced process control systems. With the limited number of language restrictions and the capability to develop diverse applications, C is convenient and effective for many test and measurement tasks, giving test engineers one more reason to keep their C/C++ skills sharpened.

Additional Reading

1. *The Embedded Software Strategic Market Intelligence Program*, Venture Development Corp., 2007.
2. *World Data Acquisition Board and Software Market*, Frost & Sullivan, 2006.
3. "Linkage to non C++ Functions," Microsoft, http://msdn2.microsoft.com/en-us/library/0603949d(VS.71).aspx
4. "About Dynamic Link Libraries," Microsoft, http://msdn2.microsoft.com/en-us/library/ms681914.aspx
5. Ritchie, D. M., "The Development of the C Language," *Second History of Programming Languages Conference*, April 1993.
6. Stroustrup, B., *The C++ Programming Language*, Second Edition, 1991.

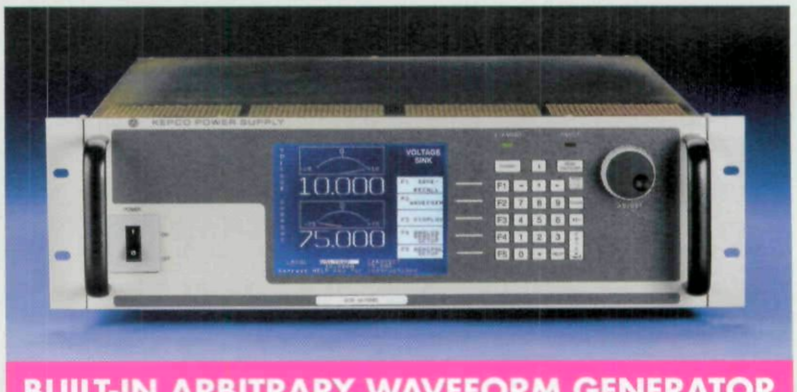
About the Author

Wendy Logan is a product marketing engineer at National Instruments. Her current projects include outbound marketing and product strategy for Measurement Studio and LabWindows/CVI. Ms. Logan began work at National Instruments in 2004 after receiving a bachelor's degree in computer science from Rice University. National Instruments, 11500 N. Mopac Expwy., Austin, TX 78759, 512-683-9311, e-mail: Wendy.Logan@ni.com

EE

A NEW GENERATION of BIPOLAR POWER

BOP 1 KILOWATT from KEPCO



BUILT-IN ARBITRARY WAVEFORM GENERATOR

The BOP 1KW from Kepco are true 4-quadrant programmable voltage and current supplies capable of full source and sink operation. To achieve low dissipation and high efficiency, when sinking power from a load, the BOP 1KW from Kepco recuperate the energy for re-use. The key to this is a bi-directional power factor correction (PFC) circuit, which allows transparent energy interchange without dissipative sinking. Keypad controls allow for automatic creation and display of various waveforms and complex patterns. They meet the EN61000-3-2 harmonic limits. A built-in EN55022 Class B input EMI filter is provided.

For more information visit the Kepco website:
www.kepcopower.com/bophi.htm

AN ISO 9001 COMPANY
KEPCO®
THE POWER SUPPLIER™
SINCE 1946

KEPCO, INC. • 131-38 Sanford Ave. • Flushing, NY 11355 USA • Tel: (718) 461-7000
Fax: (718) 767-1102 • Email: hq@kepcopower.com • **www.kepcopower.com**

Visit www.rsleads.com/805ee-017