# C-Language Programming for DSP

*Programming in C for digital signal processing applications has moved into widespread use and many powerful tools have been developed to assist the programmer. By first understanding some of the basic concepts and special considerations involved, the experience can be extremely rewarding.*

*As the software tools strive to keep pace with the rapid development of powerful new DSP devices, the mastery of special programming skills will continue to be an exciting and dynamic process.*

The C programming language has become the language of choice for many engineering applications, especially digital signal processing. The C language is extremely portable, compact, and lends itself well to structured programming techniques. It has been ported to virtually every major programming platform and is the predominant system programming language for the major operating systems used today.

Programmers familiar with the C language on PC or UNIX platforms should be aware of some key differences between C programming for general-purpose workstations and C programming for DSPs. Areas of attention include differences between native and cross compilers, simulators, I/O library support, run-time libraries, linkers, and memory modules.

The major difference in C compilers for DSP, as compared to typical C compilers found in most workstations, is that the object code produced does not execute on the host CPU, but rather on the DSP chip in the target board. This type of C compiler is called a ***cross compiler***. The more conventional C compiler which produces code for the host CPU is known as a ***native compiler***.

The reason that cross compilers are so much more appropriate than native compilers for DSP is that virtually no DSP processor environment can come close to providing all of the supporting I/O resources available on even the most modest workstation.

Since the DSP code produced by the cross compiler cannot execute on the host CPU, there are two methods of testing and developing DSP code.

First, the DSP processor can be simulated by a host program known as a ***simulator***. The simulator closely mimics the operation of the DSP chip so that executable DSP object code is processed as it would be by the DSP chip. Memory and I/O is also simulated so that the DSP target environment is matched as closely as possible. Simulators can be very useful for developing algorithms and for developing program flow and integrity.

However, this rather sterile environment insulates the programmer from the hardware-specific features of the target board which often represent some of the most critical aspects of application development, usually dealing with I/O.

As an alternative to simulation or as the next step after simulation, one can move the executable code directly into a DSP target environment and then use debug software to start it running and control execution. Running code on "the real thing" has the advantages of using actual hardware for input/output data and in evaluating real-time performance.

## Standard I/O

A fundamental aspect of the C language is that it contains no environment-specific input or output resources. Therefore, each incarnation of a C compiler requires a standard I/O library tailored specifically for each operating environment.

During debugging and development it may be required that standard I/O be redirected to the host. In this case, the I/O function must accommodate a communication link from target to host. Whatever the link, the collection of standard I/O library functions must be chosen to support the desired operation.

In complex DSP environments utilizing an operating system executing on the DSP, an abstract path may exist between target and host, often involving several layers of calls.

## Run-time Libraries

The C language itself contains no inherent routines for the many math functions, string operations, and memory management tasks required by all DSP applications. Instead, as with the standard I/O functions, the ANSI C standard defines a set of run-time library functions to be supplied as part of a C compiler package.

Most DSP C compilers incorporate an extensive set of run-time library functions, including trigonometric and transcendental functions; range limit functions for different types of variables supported; string manipulation functions; data type declaration functions; and time functions handling seconds to years with calendar notations.

The run-time libraries are often written as very efficient assembly language routines, optimized to take advantage of the special architectural features of the target DSP.

## DSP Libraries

The need to extend the scope of the run-time libraries to include more exotic and powerful routines common to DSP applications is satisfied by a class of routines called DSP libraries. They include vector and matrix arithmetic operations, FFTs,

*[Continued on next page]*

digital filtering routines, windowing functions, and image processing routines.

These functions allow the DSP code developer to work at a much higher level and alleviate the burden of reinventing common signal processing tasks. In some cases, however, the routines involve trade-offs in execution speed vs. ease of use. Custom versions of these functions can be easily substituted in these cases.

Some of these library functions are supplied with the C compiler and some are available from third party vendors.

## Board Support Libraries

The Board Support Libraries offer a fast and tested approach to programming hardware. Functions provided within these libraries allow the programmer to access the hardware at different architectural levels.

For example, a board support library written for a 16-channel A/D interface may include high-level functions such as *reset_all_A/Ds()*, or *initialize_board()* which program the global settings of the board.

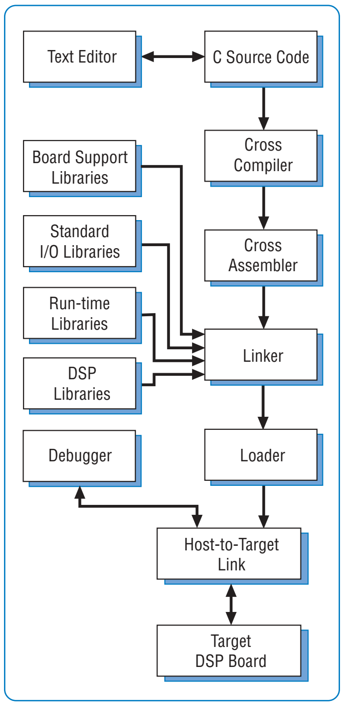
Pentek offers **ReadyFlow** Board Support Libraries, a collection of high-level C-callable functions, to simplify system development.

## DSP Memory Models

An important consideration when using a C compiler optimized for DSP applications is its linker and memory model. Most compilers, such as the TI C Compiler, produce an intermediate assembly language output file. The assembly language file is then processed by an assembler to produce a relocatable object file.

One advantage of the intermediate assembly language phase is that it allows inspection and modification of the assembly code, often necessary for debugging target hardware/software interactions.

In conventional native C compilers running on a host system, all memory sections are typically allocated to one region of system RAM. However, a C compiler for DSP must be capable of carefully allocating these sections of code to the many different kinds of memory typically found on DSP target boards—fast SRAM, slower DRAM, external ROM, and nonvolatile RAM. There are also many kinds of hardware devices which can be addressed like memory, including registers, FIFOs, counters, and UARTs.



***DSP programming is a multistep operation.***

Initialized code sections can be allocated to RAM or ROM, while the uninitialized sections must be allocated to RAM. This allocation is handled by the linker which processes all the relocatable object files from the assembler, the run-time library functions, standard I/O and DSP library functions. The linker allocates code sections from each of these structures into larger, homogenous regions of the same type.

The final result is a single file divided into sections for each code type, each with a header specifying where that section is to be placed when loaded into the hardware target memory space.

## Host-to-Target Links

One of the most challenging aspects of DSP code development is establishing an easy-to-use method of communication between the host workstation and the target DSP board. There are many different combinations of host/target environments which involve the use of bus adapters, Ethernet adapters, emulators, embedded board level host processors, integral card cages and backplanes, as well as many one-of-a-kind interfaces.

This problem is often complicated by having multiple DSP processors on one board, multiple target DSP boards, or even multiple card cages full of target DSP boards. Whatever the arrangement, in order for the loader to execute successfully, the host must be able to access each DSP target memory environment.

To solve this problem, Pentek has developed **SwiftNet,** a universal, bidirectional communication protocol between host and target.

## Debuggers

As stated earlier, one of the most powerful tools for code development is the interactive debugger. With this approach, once the object code has been loaded into the target DSP memory, the user can start, monitor, and control the execution of the processor.

This requires an even more powerful kind of host-to-target link than the one necessary for the loader. In addition to accessing target memory, the host must be able to examine all internal CPU registers, access other hardware resources on the target board, set breakpoints, single step through instructions, and, in general, manipulate the target DSP in many different ways. ❑