

# Blackhawk™ Emulation

TECHNICAL ARTICLE

## POSIX® API Extends DSP/BIOS™ Reach for More DSP Applications

*Adding support for IEEE 1003 POSIX Standard provides  
standardized programming interface*

BHTailwind-TA-01  
September 2008



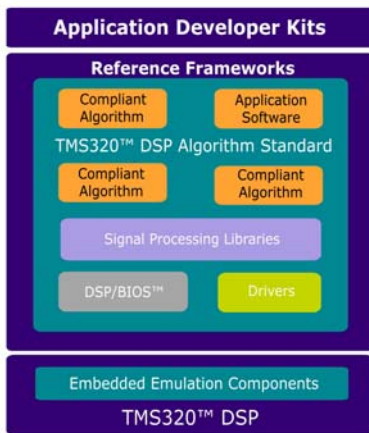
# POSIX® API Extends DSP/BIOS™ Reach for More DSP Applications

**Adding support for IEEE 1003 POSIX Standard provides standardized programming interface**

*Written by Peter Petrov, FADATA, AD and Andrew Ferrari, EWA Technologies, Inc.*

Digital Signal Processor (DSP) applications are increasingly complex, advanced systems, often served by embedded Real Time Operating Systems (RTOS) to manage their size, numerous tasks, and external interfaces. Source code portability, a progressively more important requirement for these systems, is being mandated for commercial applications and government contracts.

Today, wireless networks and devices must be compatible over a wide variety of changing communication standards.



**Figure 1 - eXpressDSP™ Block Diagram (DSP Portion Only)**

DSP/BIOS is a part of the Texas Instruments (TI) eXpressDSP™ Tools as shown in Figure 1. Extending DSP/BIOS to meet these code portability requirements and the assortment of wireless communication protocols by adding a POSIX layer is the basis for this article.

## What is POSIX?

First, POSIX, or Portable Operating System Interface is a set standards specified under IEEE Std. 1003 and accepted by ISO, IEC and ANSI. The most well known POSIX standard is IEEE Std. 1003.1 (also referred to by ISO Std. 9945), referred to as "POSIX.1". This POSIX.1 standard specifies application programming interfaces (APIs) at the source level and is about source code portability. If you are knowledgeable about UNIX or Linux OSS, you are already familiar with POSIX.

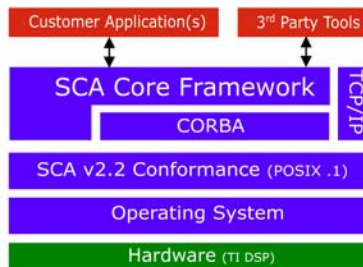
These standards govern how to write application source code so that the applications are portable between operating systems. You can find more information using the many resources on the Internet, as well as, purchasing the standards from the IEEE at [www.ieee.org](http://www.ieee.org).

## Why POSIX?

Why is POSIX conformance necessary for DSP/BIOS? To answer this we have to look at the applications and driving forces to show how POSIX fits. Before we do that, it would be

appropriate to mention that POSIX provides an easier to use programming interface for porting applications than DSP/BIOS; and one that today's programmers are more familiar with, especially those developing general purpose processor (GPP) applications for the ARM®-based TI devices.

We hear of different local agencies (fire, police, medical) having trouble assisting one another because they cannot communicate across communication systems. Or of different units within the armed forces that cannot communicate with each other during a crisis or conflict. These types of issues are driving the need for and creation of the Software Communication Architecture (SCA) operating environment (OE) standard.



**Figure 2 - SCA Operating Environment Diagram**

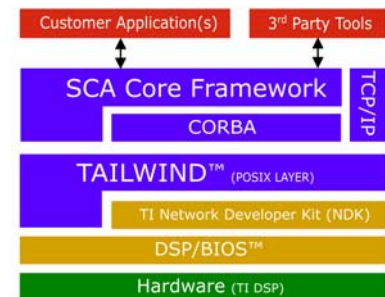
The SCA sets the rules that, if followed, will allow the growing market of wireless networks and devices to communicate. Figure 2 shows the OE as defined by the SCA standard, which consists of:

- POSIX operating system
- TCP/IP
- Common Object Request Broker Architecture (CORBA)
- SCA core framework

There are SCA-compliant OSs for programming the ARM side of TI OMAP™ and DaVinci™ devices but none, to our knowledge, for the DSP side. DSP/BIOS alone cannot support the SCA operating environment. To support the SCA OE standard, DSP/BIOS has to be extended to support POSIX. Tailwind, the Blackhawk POSIX API for DSP/BIOS, enables SCA conformance. Figure 3 shows how Tailwind fits into the SCA operating environment, allowing DSP/BIOS to support applications that require SCA compliance.

Software Defined Radio (SDR) applications implement the SCA. SDR technology enables dynamically configurable and upgradable software that is being applied in the both commercial arena (public safety radios, general radios - walkie-talkies) and in military applications, such as Joint Tactical Radio Systems (JTRS).

The next section describes, in more detail, how Tailwind meets the SCA standard.



**Figure 3 - Tailwind Diagram for SC OE**

## Tailwind Support for SCA and POSIX

Tailwind's POSIX Layer provides the complete set of SCA-required POSIX interfaces, along with numerous other services and capabilities, e.g. timeout support for blocking functions. Figure 4 graphically depicts the groups of interfaces supported by Tailwind, which are described as follows.

**POSIX Threads Base API** - The SCA stipulates a set of 40 POSIX thread (pthread) routines, which are provided by Tailwind, plus an additional 34 functions including three to support synchronization with interrupt service routines (ISRs). The Threads Base API comprises the following groups of services: POSIX Threads API, POSIX Mutexes API, and POSIX Condition Variables API.

Pthreads have 32 priority levels, as required by POSIX, which can be scheduled first-in-first-out (FIFO) or round-robin. Thread cancellation allows for sophisticated threads management, while thread-specific data offers thread local storage. Mutex and condition blocking functions support timeout, and while not required, a per-thread `errno` is supported.

**POSIX Threads Safe API** - All 7 SCA-mandated thread-safe services are provided.

**POSIX Signals API** - All 15 SCA-mandated signal routines are provided plus support for alarm and signal jump. The POSIX semantics of asynchronous signal interruption is also completely supported. Queued signals, as well as signal handling via threads, are supported with respect to timers.

**POSIX Clock and Timers API** - The full set of 9 SCA-mandated time/timer functions are provided, with the addition of the `sleep()` function.

**POSIX Semaphores API** - The full set of 9 SCA-mandated semaphore routines are provided, plus a timed semaphore wait routine.

**POSIX Thread-Safe API** - All 7 SCA-mandated thread-safe services are provided.

**POSIX File System API** - Tailwind provides concise support for POSIX-style embedded file systems by introducing a specialized File System Interface allowing the creation of installable third party file systems. In addition, a facility for

file system symbolic links is provided that treats a directory as a root file system (mount-like feature).

The Tailwind distribution includes the following core file systems:

- Root File System (“/”)
- Device File System (“/dev”)
- Memory-based File System (“/fsmem”)
- Host FS (“/fshost”) using CCS host I/O

Device driver support for POSIX-style device special files is ensured by offering a specialized Device Driver Interface that allows the creation of installable third party device drivers. In addition, a facility for device symbolic links is provided supporting device aliases.

The Tailwind distribution includes the following basic device special files:

- Null device (“/dev/null”)
- Host Console (“/dev/tty”) via CCStudio host I/O

The full set of 22 SCA-mandated file system services is provided, augmented by non-standard services to create and destroy file systems, devices, device I/O control and direct file access. Some of these functions are required by the C standard, e.g. `tmpfile()` and are part of the RTS implementation.

**POSIX File/Directory API** – The full set of 6 SCA-mandated file and directory management service is provided, augmented by 6 non-required functions.

**POSIX File/Device I/O API** – The full set of 33 SCA-mandated file/device I/O services are provided augmented by 7 non-required functions. Many of these functions are implemented by the CCS RTS, but use the file descriptor level Tailwind services.

**POSIX File Attributes API** – Although not required by SCA, Tailwind provides some services to handle file attributes.

**POSIX C Language Support API** – The full set of 54 SCA-required services is provided augmented by 35 additional functions. Most of them are present in the CCS RTS and some are overridden to ensure POSIX conformance, such as `localtime()` and `gmtime()`.

**POSIX C Language Math API** – The full set of 22 SCA-required services is provided via the CCS RTS augmented by numerous non-mandated functions.

**POSIX Sockets API** – While SCA does not require networking functionality, Tailwind leverages the Network Developer Kit product of TI to offer POSIX-conformant sockets and select API. Not only Tailwind fully integrates the NDK sockets with the POSIX file system, but also allows for all blocking functions to be interruptible by signals and adds services missing in the NDK package, such as `socketpair()` and `sendmsg()/recvmsg()`.

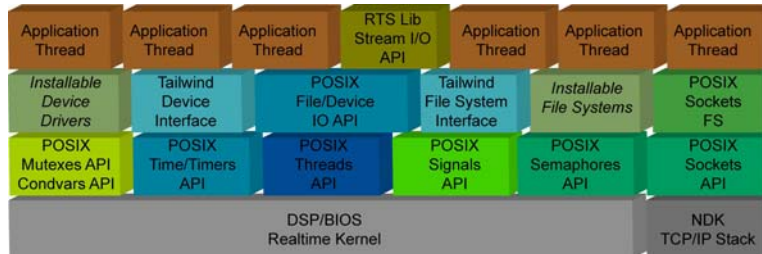


Figure 4 - Tailwind Block Diagram

**Tailwind-SCA API Matrix** – Table 1 presents the relationship matrix between the SCA-MANdated functionality and the services provided by Tailwind. It is evident that Tailwind offers a much richer set of API than required, facilitating the porting of POSIX-relying middleware such as CORBA.

AEP Unit of Functionality	SCA-MAN	Tailwind
POSIX_C_LANG_JUMP	2	2
POSIX_C_LANG_MATH	22	100+
POSIX_C_LANG_SUPP	54	89
POSIX_DEVICE_IO	33	40
POSIX_DEVICE_SPEC	0	1
POSIX_FD_MGMT	6	12
POSIX_FILE_ATTRIB	0	3
POSIX_FILE_SYSTEM	22	33
POSIX_JOB_CONTROL	0	0
POSIX_MULTI_PROCE	0	7
POSIX_NETWORKING	0	25
POSIX_PIPE	0	1
POSIX_SEMAPHORES	9	10
POSIX_SIGNAL_JUMP	0	2
POSIX_SIGNALS	15	15
POSIX_SINGLE_PROC	0	3
POSIX_THREADS_BAS	40	74
POSIX_THREAD_SAFE	7	8
POSIX_TIMERS	9	9

Table 1 - Tailwind-SCA API Matrix Table

### Tailwind Implementation

Now that we described how Tailwind meets POSIX standards and exceeds SCA requirements, we can describe how this has been implemented as a layer for DSP/BIOS.

**DSP/BIOS Relationship** – The DSP/BIOS features enumerated below outline the relationship with DSP/BIOS.

- Threads and Tasks - Every POSIX thread is represented by exactly one DSP/BIOS task. POSIX threads are blocked by setting their priority to minus one.
- Thread Priorities - While DSP/BIOS supports only 16 task priorities, Tailwind provides 32.

The priority zero, as in DSP/BIOS, is implicitly round-robin scheduled to allow for BIOS idle processing.

- Thread Scheduling - Tailwind supports both FIFO and round-robin scheduling of POSIX threads.

- PRD Function - Tailwind

needs a periodic function to be called implemented by DSP/BIOS PRD interface. This function is used to support timeouts for blocking functions, timers and thread round-robin scheduling.

- HOOK Function - Tailwind needs a task switch hook function to be called implemented by DSP/BIOS HOOK interface. This function is used to support asynchronous signal interruption and cancellation, as well as thread exception context switch for C++ with exceptions.

- Memory Management - Tailwind relies on DSP/BIOS `malloc()/free()` for all dynamically allocated storage.

- POSIX and DSP/BIOS Mixing - Although largely technically possible, mixing DSP/BIOS and POSIX API in the same application is not advisable.

### TI DSP Support and Availability

#### TI DSP Platform Support:

- C64x big and little endian
- C64x+ big and little endian
- C6000 Compiler versions 6.0.x and 6.1.x
- Socket library for C6400 and C6400+ big and little endian (NDK 1.93/1.94)
- DSK6455 and DSK6416 Target Boards

#### Available Distributions:

- Evaluation (limited runtime)
- Standard distribution (binaries, link libraries, examples)
- Full source-code distribution

Blackhawk  
123 Gaither Drive  
Mount Laurel, NJ 08054-1701  
Web: [www.blackhawk-dsp.com](http://www.blackhawk-dsp.com)

