



### Benefits

- Reduces learning curve and time-to-market by providing a high-level Application Programming Interface (API) to *flexComm* hardware, abstracting the details of the board, bus, host and host operating system (OS)
- Provides maximum flexibility and configurability through a simple API
- Increases portability and interoperability with other *quicComm* enabled platforms
- Designed to provide high-performance and low latency in a multiprocessor communications environment
- Fully integrated with Spectrum's high capacity Solano~links as well as standards-based interfaces such as VME, PCI, CompactPCI®, Raceway and RapidIO™
- Operating System abstraction ensures application portability across different operating systems
- Facilitates the integration of Spectrum and third-party components

### Features

- Supports high-performance inter-processor communications
- Simplified board and system setup and control
- Provides a standard application interface across Spectrum's *flexComm* family of products
- Supports dynamic reload of executables on a per processor or FPGA basis
- Supports synchronous and asynchronous communication
- Supports logical channels for inter-processor communications
- Provides a means to integrate application software with the FPGA firmware
- Compatible with Texas Instruments DSP/BIOS™ (C6x systems), Wind River® VxWorks® (PPC systems) and Green Hills® INTEGRITY® OS
- Provides an I/O porting kit enabling support of third-party modules

### Description

*quicComm* is a high-performance software library that enables board and system features not available through standard operating system calls. The *quicComm* library facilitates the implementation of complex signal processing and data acquisition applications by abstracting low-level details, allowing the programmer to focus on their signal processing application. The *quicComm* library includes support for high-performance inter-processor communications hardware and other board-level features such as interrupt handling. The API is designed to be simple and user friendly, isolating the programmer from the complexities of the hardware while maintaining near-hardware level performance.

Figure 1 illustrates the software operating environment supported on Spectrum's *flexComm* products. *quicComm* is an integral component that enables the efficient transfer of data between application resources operating on disparate processing devices and communication over multiple independent buses and fabrics using a common API.

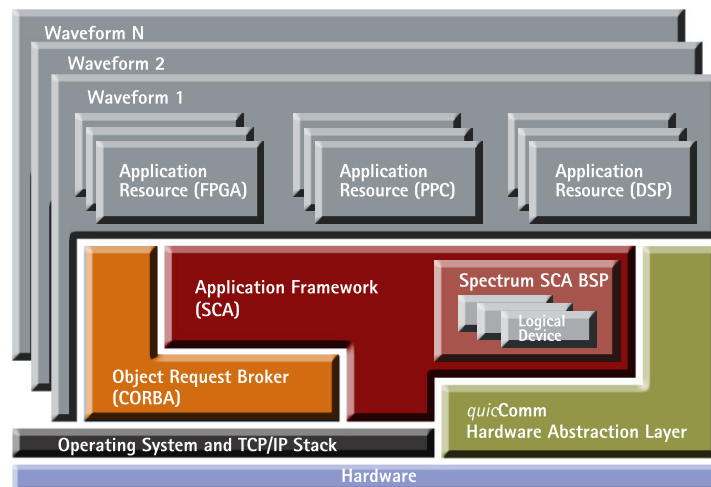


Figure 1. Spectrum Software Operating Environment

## Architecture

The *quicComm* software library is designed to support a multiprocessor communications environment that is layered on both the run-time host and the target processor. *quicComm* enables all the features of the hardware without bogging the developer down with the hardware details.

The *quicComm* library functions operate on entities known as *quicComm* resources. *quicComm* uses a System Definition File (SDF) to define and describe the set of resources available to an application. Each *quicComm* supported module and board has its own set of resources and an accompanying *quicComm* software development kit (SDK) to access these resources. The type and number of modules and boards in a signal processing platform determine the resources available to a *quicComm* application.

The diagram in figure 2 shows the internal architecture of the *quicComm* software stack. The Board Support Package provides a software interface between the Operating System (OS) and the underlying hardware. For example, each Spectrum board with a PowerPC has a Board Support Package that allows the Operating System to execute on that processor. The Driver Interface provides a low-level OS-independent interface to the hardware that can be implemented as either OS-independent or OS-specific. In some cases, it will use OS facilities while in others it will interface directly to the hardware. Using a common API for all device drivers in one subsystem allows these drivers to be implemented in the most efficient manner possible.

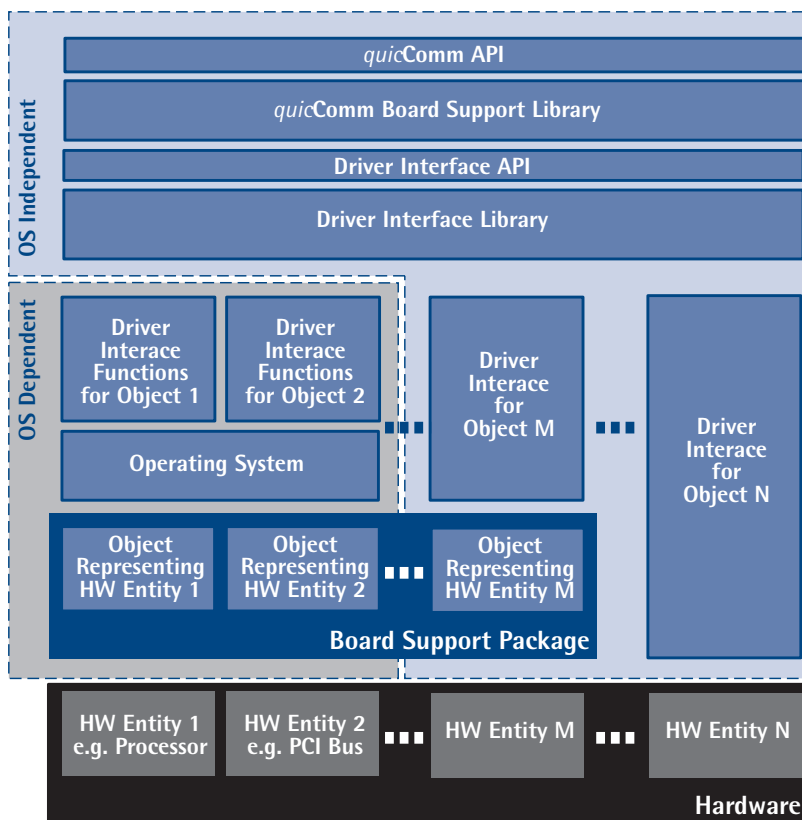


Figure 2. *quicComm* Architecture Conceptual Diagram

*quicComm* provides a uniform API across all platforms. To do this, *quicComm* employs an integrated board support library (BSL) for each carrier board or module allowing special handling of resources specific to the carrier board or module hardware as shown in Figure 3. The *quicComm* library contains everything required to initialize the hardware, download programs to target processors, transfer data between processors and host or module hardware, control data link communication and to generate and respond to interrupts and signals.

### Resources

The most common *quicComm* resources are:

- hardware devices (e.g. processors, memory, bridge chips)
- boards (e.g. XMC-3311, XMC-3321, TM1-3300, PRO-4600, PRO-3500, PRO-3100, PRO-2900, PRO-1900)
- systems (e.g. SDR-4001, SDR-3001, HCDR-1002)
- logical communication paths

All *quicComm* resources are referenced through a logical name or name-tag. Programatic access to the resource is provided through a 'handle' to the resource, defined at the time a resource is 'opened' via the `QC_Open()` call. When a resource is opened, it is automatically configured for appropriate operation and internal data structures are initialized to allow the resource to be seamlessly accessed via other *quicComm* library calls.

*quicComm* resources can be 'claimed' to ensure availability and to minimize latency. This functionality is especially important for real-time software applications that require assurance of critical limited resources or bus-bandwidth. The ability to claim the required resources allows for deterministic real-time performance, avoiding run-time problems that arise from resource contention, which are often very difficult to debug. The *quicComm* resource is simply 'released' once it is no longer required by the application.

Every *quicComm* resource has attributes – properties that can be accessed by the software application. Typical attributes include the resource name, size and physical and virtual (mapped) address. The attributes can be accessed by the software application through a standard *quicComm* API interface.

The System Descriptor File (SDF) defines which resources are available. The Board Support Library determines which attributes the resources supports.

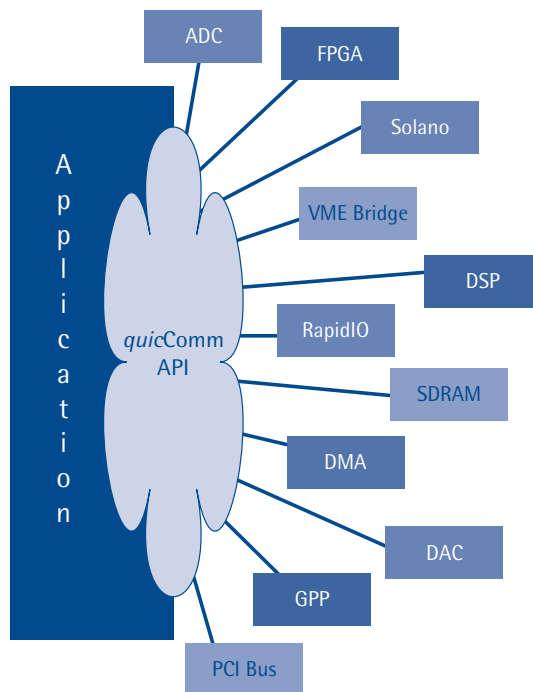


Figure 3. *quicComm* Conceptual Diagram

## Inter-processor Communication

A fundamental function of most signal processing applications is the transfer of large quantities of data through multiple processors within a signal processing platform while performing analysis and manipulation as the data flows through the platform. Efficient inter-processor communication is critical to ensuring real-time sustained signal processing performance. *quicComm* provides a common communication API to all processors that support *quicComm*, including the host, facilitating efficient transfer of data through various methods.

*quicComm* provides all data transfer functions in both blocking and non-blocking mode. A blocking function means that the calling thread will not return until the data transfer is complete. This provides the software application a very simple means to synchronize code execution to the data transfer. Non-blocking functions return immediately, allowing DMA-driven data transfers and core processing to occur concurrently. A callback mechanism is included with the non-blocking functions to allow synchronization.

### [ Link API ]

*quicComm* Link APIs facilitate the transfer of data between processors through data links or pipes. Links allow two processors to transfer data to each other using a simple, link-level protocol. The basic mechanism of the data transfer is that one processor sends data to the link, while the other reads it.

The types of *quicComm* links include Solano~links and processor-to-processor links. The Solano~links are physical high-speed inter-processor links that allow data transfer. Processor-to-processor links are virtual links that allow data transfers between memory on two different processors. These processors can either be on the same board or on different boards and may span multiple buses such as PCI and VME.

To establish a *quicComm* link, all that is required is a simple open, initialize and bind process using the *quicComm* library calls. Communication can then commence using the `QC_LinkRead()` and `QC_LinkWrite()` calls.

### [ Memory Mapped API ]

Direct memory copies are supported by *quicComm* through the Memory Mapped APIs, providing extremely flexible and versatile interfaces to the processor, board and platform. The Memory Mapped APIs provides blocking memory read/write functions for reading data from or writing data to specified memory locations.

On some processors such as Texas Instruments' C6x, software applications can access the DMA engines as a *quicComm* resource. This provides a very efficient method of transferring large data blocks between memory locations in the processor address space including internal and external memory.

### [ Signal API ]

Signals are physical or logical interrupts sent and received by applications. Using *quicComm* signal APIs, an application can send a signal to another application, which can respond to the signal as required. This provides a very efficient method of asynchronous notification between two applications.

## CORBA Support

CORBA (Common Object Request Broker Architecture) is an OMG (Object Management Group) middleware specification and widely adopted industry standard. CORBA enables and promotes software interoperability by abstracting almost all details of platform, language, location, vendor and network architecture away from application code.

Spectrum's PowerPC-based flexComm products support an Object Request Broker (ORB), enabling CORBA applications to be developed for a heterogeneous environment. Although many ORBs are commercially available from various vendors, Spectrum selected the TAO open-source ORB due to its performance, in combination with the level of industry adoption and price. CORBA applications may be developed directly onto Spectrum's platform using TAO. Support for additional ORBs is underway.

## Performance

Designed from the ground up to provide near-hardware level performance and low latency, *quicComm* is a standard part of Spectrum's software operating environment. The *quicComm* library is optimized to provide maximum performance while providing high-level abstraction of the hardware details. Even high-speed Solano~links or RapidIO are treated as *quicComm* resources, providing ease of programming at an API level, while achieving optimal inter-processor communication performance.

## Dynamic Reload

*quicComm* enables dynamic or 'on-the-fly' reconfiguration of the target application. This is critical to applications requiring efficient, deterministic real-time reconfiguration of the platform's components. The dynamic reload capability is on a per processor basis, facilitating the development of application requiring the dynamic downloading of target executables such as in software defined radio (SDR).

## I/O Support

I/O is treated in the same way as processor boards in a *quicComm* system. Each I/O module comprises one or more resources, which are accessed through standard *quicComm* functions. Spectrum provides I/O specific APIs which layer on top of *quicComm* to provide additional functionality. For example, the ePMC-FPGA software development kit (SDK) includes API-level digital down converter (DDC) and digital up converter (DUC) calls, facilitating the development of signal processing applications.

A porting kit allows third-party I/O modules to be supported. Once ported, these I/O modules can be accessed just like any other *quicComm* resources, facilitating the development of the application. The porting kit is applicable to cPCI, PMC and VME I/O solutions.

## Standards

*quicComm* is Spectrum's programming standard for the *flexComm* line of products, and provides an operating system independent compact library of APIs consistent across all *quicComm* resources. Integrated with familiar standard development tools such as Texas Instruments' Code Composer Studio™, Wind River's Tornado® IDE, and Microsoft Visual Studio®, the developer learning time is minimized.

In addition, to ensure seamless and efficient integration of *flexComm* products into fully integrated digital signal processing applications, *quicComm* is compliant with many of the industry's interface standards. Supported standards include Raceway, RapidIO (Serial and Parallel) and access to many of the standard object and binary file types.

## Portability

*quicComm*'s unified API allows applications to be easily ported between operating systems and Spectrum hardware platforms. This greatly reduces the learning curve and the application development time. It also facilitates migration to next generation hardware. In the case of a heterogeneous system (e.g. Windows host and VxWorks target), *quicComm* abstracts away the specifics of the different operating systems, facilitating easy development.

## quicComm Software Development Kit

quicComm is packaged as a software development kit (SDK) for each of the various flexComm products. Run-time libraries, header files, reference examples (with source) and full documentation manuals are included in each quicComm SDK.

### [ quicComm SDK API Summary ]

The following table summarizes the quicComm API provided in the SDK. The APIs are common across the flexComm product line. quicComm is available on the majority of flexComm products and will be available on all future generations, allowing maximum code portability and reducing the learning curve.

Category	API	Description
Open	QC_Open()	open a resource in the system (including the system/processor itself)
	QC_Close()	close an open resource in the system
Execute	QC_Execute()	load resources with an image (with the option to execute image)
Reset	QC_Reset()	put the resource into a known state
Locking	QC_Claim()	claim the resource for exclusive use
	QC_Release()	release a claimed resource
Memory	QC_MemRead()	read from the resource's memory map - with callback
	QC_MemReadBlocking()	read from the resource's memory map - wait until done
	QC_MemWrite()	write to the resource's memory map - with callback
	QC_MemWriteBlocking()	write to the resource's memory map - wait until done
Flush	QC_Flush()	complete all prior writes to a resource
Link Access	QC_LinkBindAll()	bind all links to prepare them for use
	QC_LinkRead()	read from the link (pipe, channel, etc.) of a resource - with callback
	QC_LinkReadBlocking()	read from the link (pipe, channel, etc.) of a resource - wait until done
	QC_LinkWrite()	write to the link (pipe, channel, etc) of a resource - with callback
	QC_LinkWriteBlocking()	write to the link (pipe, channel, etc) of a resource - wait until done
LED	QC_LedOff()	turn one or more LEDs off
	QC_LedOn()	turn one or more LEDs on
	QC_LedStatus()	retrieve the status of a single LED
	QC_LedToggle()	toggle one or more LEDs
Attributes	QC_GetNumericalAttribute()	retrieve a numerical property of a resource
	QC_GetPointerAttribute()	retrieve an address (pointer) property of a resource
	QC_GetStringArrayAttribute()	retrieve a string array property of a resource
	QC_GetStringAttribute()	retrieve a string property of a resource
DMA	QC_Copy()	copy data from one memory location to another - with callback
	QC_CopyBlocking()	copy data from one memory location to another - wait until done
Signal	QC_SendSignal()	send a signal to a resource - with callback
	QC_SendSignalBlocking()	send a signal to a resource - wait until it arrives
	QC_SignalCallbackConnect()	connect a function to a signal
	QC_SignalCallbackDisconnect()	disconnect a function from a signal
	QC_WaitSignal()	test for a signal from a resource
	QC_TestWaitSignal()	wait for a signal from a resource

[ Documentation ]

The *quicComm* SDK includes a full set of manuals. The documentation details all of the information required to get programming quickly, including step-by-step instructions on the installation of the *quicComm* library, setting up the development environment, and a description of the *quicComm* resources available to the specific SDK along with the attributes specified for each *quicComm* resource.

The *quicComm* API documentation is also provided as a soft copy HTML-based help document, allowing very efficient retrieval of relevant information. A sample of the documentation is provided in Figure 4.

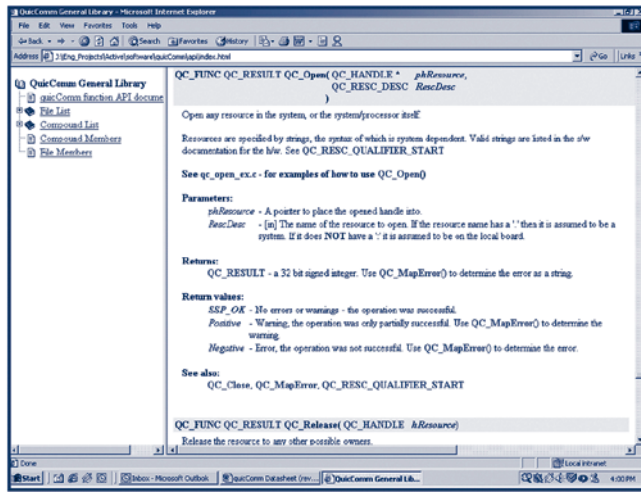


Figure 4. Sample On-Line Documentation

[ Reference Example ]

Reference examples demonstrate the use of the resources specific to the SDK and are included as source code in the *quicComm* SDK. These examples serve as excellent reference designs and provide a starting point for the application development. The reference examples are fully documented in the manual set as well as in the HTML-based help. A sample of the on-line documentation for the reference examples is provided in Figure 5.

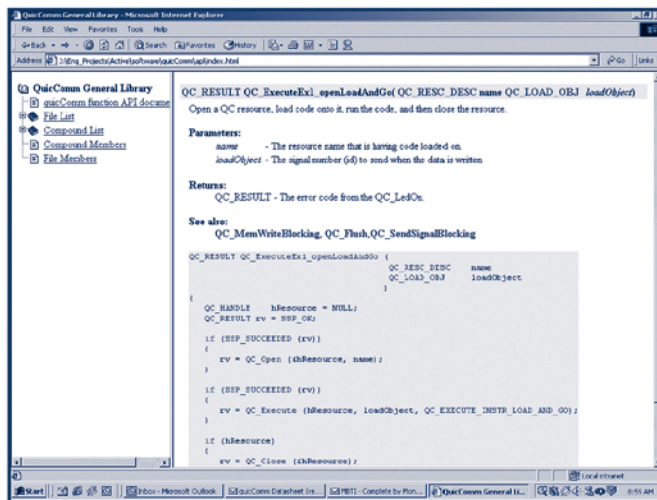


Figure 5. Sample Reference Example

## Specifications

[ compatible products ]	Spectrum Products	SDR-2000 Product Line SDR-3000 Product Line SDR-4000 Product Line HCDR PowerPC Product Line HCDR C6x Product Line
	Non-Spectrum Products	SBS Motorola
[ PowerPC-based ]	Development Operating System	Windows® 2000, Windows XP
	Target Operating System	VxWorks, INTEGRITY
	Host Operating System	Windows 2000, Windows XP, VxWorks and Linux
[ C6x based ]	Development Operating System	Windows 2000, Windows XP
	Target Operating System	DSP/BIOS Native Mode
	Host Operating System	Windows 2000, Windows XP, VxWorks and Linux
[ ordering information ]		Please consult the individual datasheets of the supported hardware platforms for specific ordering information or contact Spectrum sales.