Data Sheet

Continuum IPC
Inter-processor Communications Library

continuum
INTER-PROCESSOR COMMUNICATIONS

Features

- High-performance, low latency message passing library
- POSIX-style application interface
- Message passing features priorities, time-outs, and copyless transfers
- Bulk data transfer from any processor's memory to any other processor's memory
- Memoryless signaling from any processor to any other processor
- Global buffer and global semaphore API
- Segmented block transfer capability supports memory striding, matrix manipulation and submatrix selection
- Support for multi-core processors
- Supported under VxWorks on CHAMP multi-processor boards, FPGA boards, and single board computers

Benefits

- High performance, low latency and easy-to-use messaging and data transfer capabilities
- Task-based communications between processor cores, processors on-board, or processors within a fabric
- Platform and fabric independence allow easy migration to emerging interconnect standards while architectural details are hidden
- Distributed design with no single points of failure
- Supports dynamic entry of nodes for high availability applications
- Dynamic name space management allows message endpoints and data buffers to be identified by application-determined names at run-time

Continuum IPC Overview

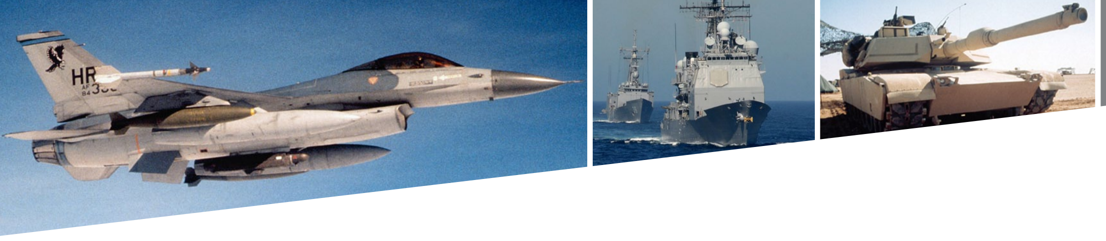
Real-time sensor based systems require status, control and bulk data movement to coexist within the same communication architecture. The Inter-processor Communication (Continuum IPC) Library provides all the capabilities needed to control applications running on multiple processors having data movement requirements. The software hides the hardware architectural details from the application designer. For example, message passing functions send messages using the same application interface and parameters, whether the destination task resides on the same processor, same board, or any other board within the fabric. The hardware abstraction provided by Continuum IPC ensures that applications written for current generation processors and fabrics will be easily portable to the next generation. Depending on the board and fabric architecture, IPC uses Serial RapidIO® (sRIO), StarFabric, PCI Express® (PCIe) or PCI/X as on-board and/or inter-board interconnects.

The library determines the location and end-point addressing for source and destination as applications open endpoints for input (receiving messages) or output (sending messages). With routing information stored within each processor, the software does not impose a single point of failure as would result from a centralized table of buffer or destination mapping information.

Learn More
Web / sales.cwcembedded.com
Email / sales@cwcembedded.com

ABOVE & BEYOND

CURTISS WRIGHT Controls
Embedded Computing
cwcembedded.com



Continuum IPC Capabilities

The Continuum IPC Library provides the following capabilities:

- Message passing, from task to task
- Data transfer from any processor's memory space to any other processor's memory space
- Signals (implemented using sRIO doorbells or hardware interrupts) from any processor to any processor
- Dynamic name space management; message endpoints and data buffers are identified by application-determined names at run time

Message passing is provided through POSIX-style open, close, read and write functions, and also through an extended interface that provides control over more features than the POSIX interface. Message passing is priority-based, queue-driven, flow-controlled, and reliable in nature.

The bulk data transfer capabilities move large blocks of data from one processor memory space to another. An application creates a receive buffer, and announces it to the network by providing the buffer name, address, and size. Other processors open the buffer, using the same name, for output operations. Write operations cause DMA transfers from the source processor's buffer to the destination buffer. Because writers do not need to know the physical address of the buffer (only the name need be known), system architects are free to move the buffer anywhere within the system to meet performance requirements, without changing source code in the writing tasks.

A block striding feature assists with corner turning and facilitates scatter/gather operations. Writers specify a destination offset within the target buffer, allowing multiple processors to write to different areas of a shared buffer.

Data transfer operations are asynchronous, completing after the function returns. The data transfer functions support callbacks, allowing the application to know when the transfer completes. Data transfers are optionally accompanied by signals. Signals, associated with receive buffers, allow a writing task to assert a notification to a receiving processor after a DMA completes. This mechanism can be used to signal a "data ready" condition to the receiving application.

The library provides the same capabilities, using the same interface, whether the source and destination are the same processor, same board, or anywhere within the fabric. All functions are designed to support high throughput and low latency needed by real-time systems.

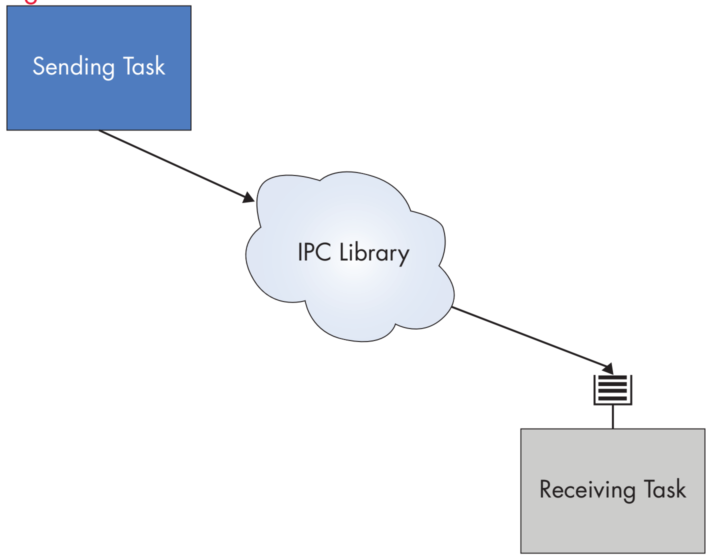
Message Passing

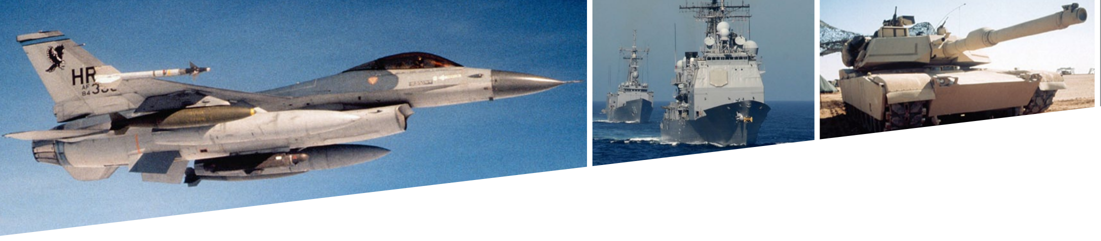
The Continuum IPC Library provides two forms of applications interface for message passing functions. Message passing is based on the standard I/O *open*, *close*, *read*, *write*, and *ioctl* functions, shown in Table 1. Special features are controlled through *ioctl* functions. Messages are data structures sent from one task to another. Sending and receiving tasks may reside anywhere within the system. Message passing functions move messages from sender to receiver, using the appropriate hardware paths connecting sender and receiver.

Table 1: POSIX Compliant Message Passing Functions

Function	Description
open	Open an endpoint
close	Close an endpoint
read	Read (receive) a message
write	Write (send) a message
ioctl	I/O control for messages

Figure 1: IPC Software Model





The Continuum IPC Library also supports a more sophisticated form of message passing (Advanced Message Passing), as seen in Table 2. For example, the *message send* function has two additional parameters: *priority* and *time-out*. This allows an application to specify the time-out behavior and priority with each send operation, rather than through changing defaults using *ioctl*.

Table 2: Advanced Message Passing Functions

Function	Description
msgOpen	Open an endpoint
msgClose	Close an endpoint
msgReceive	Receive a message
msgReceiveBuf	Receive a message (copyless form)
msgSend	Send a message
msgIoctl	I/O control for messages

Other functions in the library offer significant performance improvements. For example, it may not be necessary to copy input data from a message buffer into an application's data area (*read* performs such a copyless operation). Instead, an alternate *read* function provides the address of input data within a message buffer, eliminating the need for a copy.

Bulk Data Transfers

Portions of an application may not require the queuing and priority control provided by the message passing functions. When data volume is high and data is time-perishable, the use of a reliable protocol can increase latency and consume additional CPU time. The introduction of data queues, copying and other mechanisms consumes processor cycles while the queued data ages and data continues to stream in from the input sensor. In this case, it may be advantageous to use a lower overhead transfer mechanism.

The Bulk Data Transfer (BDT) functions are designed for high volume, low-latency data, such as that which occurs in data flow architectures. The BDT functions provide a different class of service than the message passing functions. Within the BDT functions, a processor creates and owns a buffer. The owner is the receiver; other processors open the buffer (using the same name) and write data to the buffer. The BDT functions allow:

- the owner to "advertise" its buffer, by name, to other processors
- senders to open the same buffer for output, by name
- senders to DMA data from their local memory space to any place within the destination buffer
- the sender to optionally signal itself upon completion of a DMA operation
- the sender to optionally signal the receiver that data has been sent

The senders may reside anywhere within the system, including on the same processor that owns the buffer. As with message passing, the application is isolated from the actual data movement mechanism.

The BDT functions are shown in Table 3.

Table 3: Bulk Data Transfer Functions

Function	Description
msgBlkCreate	Create a buffer
msgBlkOpen	Open a buffer
msgBlkClose	Close a buffer
msgBlkWrite	Write data to a remote buffer
msgBlkWriteSeg	Write data segments to a remote buffer

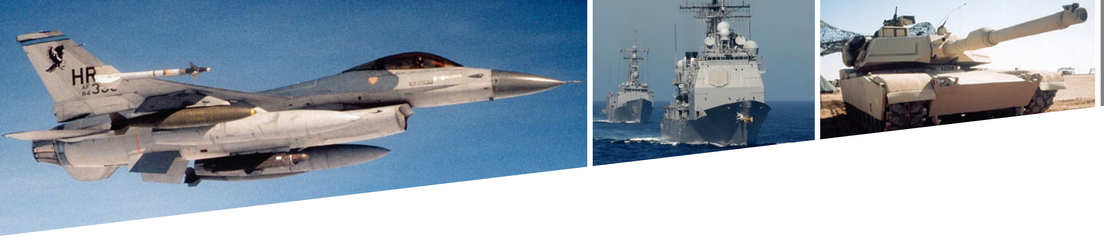
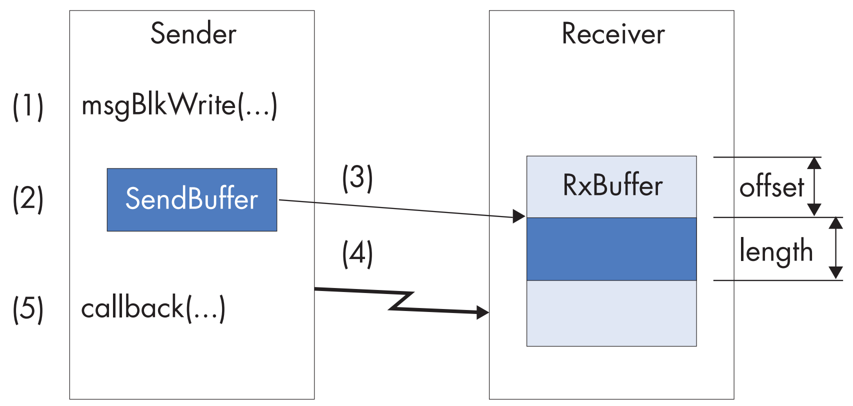


Figure 2: Example of a Bulk Data Transfer



(1) Sender calls msgBlkWrite
(2) Having started DMA transfer, control returns to calling program
(3) msgBlkWrite transfers length bytes from sendBuffer into RxBuffer starting at offset
(4) At end of transfer, msgBlkWrite interrupts owning processor
(5) msgBlkWrite calls callback function

Segmented Block Transfers

It is sometimes necessary to transfer multiple blocks of data, each separated by some fixed number of bytes (a source stride), to a destination buffer, where each block is separated by a different destination stride. For example, an application may need to transfer a submatrix into a larger matrix, or extract a submatrix from a larger matrix.

In RADAR and SONAR applications, matrices of data require transposition between various computation stages. If data is to be moved from one processor to another between computational stages, the transpose performance can be improved by partitioning the matrix and transposing the partitions during the data movement.

The segmented block transfer operation can improve performance through matrix partitioning. A large matrix can be divided into several submatrices. The submatrices can then be moved to a target matrix, transposing the partitions (but not the elements within the partitions) using *msgBlkWriteSeg*. The resulting individual partitions can then be transposed using a matrix transpose function. Since these submatrices are smaller than the overall matrix, they lend themselves to improved cache utilization, improving performance.

Signaling

Due to the nature of DMA operations, a processor is not aware that other processors have transferred data into a buffer. The advantage of DMA operations is that the data movement takes place independently of and concurrently with CPU activity. It is therefore necessary to provide some means for a writing process to gain the attention of the processor that owns the buffer. This could be accomplished using a Continuum IPC message. However, signals are more efficient.

The number of signals available on any given processor is board-specific. For example, IPC uses RapidIO doorbell messages, with a 16-bit software-defined information field, for inter-processor signaling. Signals may also be implemented as hardware interrupts on VME systems or when signaling between CPU cores. Table 4 shows the signaling functions available in the Continuum IPC Library.

Table 4: Signaling Functions

Function	Description
msgBlkSigEnable	Enable (unmask) a signal
msgBlkSigDisable	Disable (mask) a signal
msgBlkSigConnect	Connect signal handler function
msgBlkSignal	Generate a signal
msgBlkIoctl	I/O control for Blk transfers

Global Buffers

The Global Buffers and Global Semaphores provide an additional set of APIs enabling multiple CPUs to share memory and communicate more efficiently. This additional API also includes a set of functions to initialize remote memory for future transparent access.

Identical to the Message Passing and the Bulk Data Transfer APIs, Global Buffers are declared and published using application-defined names at run-time. Buffers are opened and mapped, and appear as the calling node's local memory, regardless of the physical location of the buffer within the multicomputer.

Table 5: Global Buffer Functions

Function	Description
gbm_buf_create	Create a buffer
gbm_buf_open	Open a buffer
gbm_buf_close	Close a buffer
gbm_buf_map	Map a buffer

Once a buffer is mapped, data transfers can occur as simple programmed I/O operations or by utilizing the more efficient DMA method. The Global Buffer API provides an additional set of APIs specifically designed to enable an application to configure and manage DMA chains. DMA chains contain DMA descriptors that describe where data is moved to and from as well as the size of the DMA. Without application intervention, DMA operations described in the DMA chain are activated and enabled sequentially until completion.

Table 6: DMA Chain Functions

Function	Description
gbm_dma_open	Create a DMA chain
gbm_dma_close	Close a DMA chain
gbm_dma_desc	Create a new DMA descriptor and add to DMA chain
gbm_dma_start	Start a DMA transfer

Global Semaphores

Global semaphores provide the capability to any processor or task to create/attach/give/take a semaphore created anywhere within the system. Semaphores are created by an application assigned name. Tasks attempting to attach to a global semaphore reference the semaphore by its name. The flexibility of the API allows the application programmer to attempt to attach to a semaphore before it is created, and be resolved once the creation has occurred. Additionally, an application may initialize the semaphore to a taken or available state, and specify a maximum count that defines the maximum number of tasks/processes that are queued. Tasks/processes are de-queued in the order in which they are queued, following a first-in-first-out method.

Table 7: Global Semaphore Functions

Function	Description
gbm_sem_attach	Finds an existing semaphore by name
gbm_sem_close	Detaches a semaphore
gbm_sem_create	Create a semaphore
gbm_sem_give	Give access to a pending task on a semaphore, or increment count
gbm_sem_take	Request access to a semaphore

Ordering Information

The development license permits software development for one project with unlimited users, and includes run-time licenses for 20 nodes. Continuum IPC is distributed on CDROM for development under VxWorks® with Windows-based workstations. Continuum IPC is currently supported for use on all CHAMP-AV multi-processor boards, 6U FPGA processor cards and select single board computers and memory cards. Please contact a Curtiss-Wright sales representative for current hardware and operating system version support.

For single card applications order DSW-DEV-IPC-000-CD. For multi-board systems using StarFabric interconnect, the StarLink VxWorks driver is also required, part number DSW-DEV-230-000-CD. For sRIO-based systems such as the CHAMP-AV6 and VPX6-185, order DSW-IPC/VPX-000-CD.

An additional run-time royalty is required for each node in excess of the base license running the Continuum IPC software. Pricing is cumulative over the life of a single project. Order part number: DSW-RTL-IPC-000.

Warranty

This product has a one year warranty.

Contact Information

To find your appropriate sales representative:

Website: www.cwcembedded.com/sales

Email: sales@cwcembedded.com

Technical Support

For technical support:

Website: www.cwcembedded.com/support1

Email: support1@cwcembedded.com