SysLink MigrationGuide

Introduction

This document is the Migration Guide for SYS/Link, which is also known as SysLink.

The purpose of this document is to:

- Explain differences between DSPLink and SysLink
- Describe some of the architectural and functional improvements in SysLink over DSPLink and gives reasons why users should migrate to SysLink for devices where SysLink is supported.
- Give information on how application writers can migrate to SysLink from the previous product DSPLink.
- Give migration information between significant releases of SysLink.

Before reading this document, it is strongly advised to first go through the SysLink UserGuide

Terms and Abbreviations

Abbreviation	Description
CCS	Code Composer Studio
IPC	Inter-Processor Communication
GPP	General Purpose Processor e.g. ARM
DSP	Digital Signal Processor e.g. C64X
CGTools	Code Gen Tools, e.g. Compiler, Linker, Archiver
SysLink	SYS/Link
IPC product	IPC product available for SYS/BIOS 6.xx which is used in conjunction with SysLink to provide Inter-processor communication servies

This bullet indicates important information. Please read such text carefully.

Migration from DSPLink

DSP/BIOS Link is foundation software for inter-processor communication across the GPP-DSP boundary. This software can be used across platforms:

- Using SoC (System on Chip) with GPP and one DSP.
- With discrete GPP and DSP.

Supported devices include OMAP3530, OMAP2530, DM6446, DM357, DM6467, DM6437, DA830, OMAPL137, DA850, OMAPL138, DM648, TNETV107X. Supported HLOS operating systems include Linux, WinCE. DSP/BIOS 5.xx is supported on the DSP.

SysLink is the next generation of DSPLink. New multi-core device architectures such as TI81XX are complex with one master core running HLOS and 3+ cores running SYS/BIOS. They require:

- All to all IPC communication

- Processor Manager master for a slave core may be on any other core in the device, and may not always be on the host processor.
- It should be possible to communicate between two cores running SYS/BIOS in exactly the same way as the communication would happen between cores running two different OSes.
- Only some basic IPCs may be required between a pair of cores, while others may need more complex IPCs. It should be possible to pick and choose which IPC is required without pulling in the full software

The above requirements inherently make usage of DSPLink for such devices impossible, since the DSPLink architecture does not support all of the above requirements.

The below sections detail migration information from DSPLink to SysLink.

Overview

The below table gives a high level overview of differences between DSPLink and SysLink. Additional details are given in later sections.

Items in red indicate modules/features that are currently not implemented

Feature	**DSPLink**	**SysLink**
Multi-processor support	Star-topology only (one master, multiple slaves)	Grid-topology (any-to-any connections)
Configuration	Custom configuration on GPP, TCF/CFG on BIOS	XDC configuration on RTOS. On HLOS, non-XDC configuration
DSP/BIOS support	5.xx & 6.xx (legacy support only)	6.xx only
Application build	Makefiles	On RTOS-side: Makefiles – Configuro / XDC. On HLOS-side: Makefiles
Build DSPLink/SysLink	Makefiles	On RTOS-side: XDC build. On HLOS-side: Makefiles
Data Types	DSPLink types on HLOS, std.h on BIOS	XDC std.h on BIOS & equivalent on HLOS
APIs and data structures	Different APIs and data structures on HLOS & BIOS	Shared API interface header file between HLOS & RTOS
IPCs	MSGQ in BIOS different from DSPLink, other IPCs in DSPLink only	All RTOS IPCs in common IPC/SysLink product. Single place for all IPC needs.
Linux	Linux product still using OSAL, custom build system	Kernel Linux product made suitable to mainline merge
Key new IPCs	-	All in DSPLink + FrameQ + HeapMemMP
New Processor Manager features	-	All in DSPLink + Dynamic Memory Mapping (DMM), Slave Error Handling, Power Management, Dynamic Linking & Loading
New SysLink features	-	Dynamic System Memory Manager – Supports managing physically contiguous regions and dynamic mapping to devices with slave MMU
Slave Loader	COFF	ELF/COFF
Physical links	Assumes Shared Memory or memory mapped IO – very difficult to port in absence of any shared/mapped memory	Portable architecture to support completely different types of physical links

The below table gives a high level overview of differences between DSPLink and SysLink modules/features

DSPLink module/feature	**SysLink module/feature**
MSGQ (also DSP/BIOS MSGQ)	MessageQ
RingIO	RingIO
NOTIFY	Notify
MPLIST	ListMP
POOL	Memory module (HeapBufMP/HeapMultiBufMP/HeapMemMP)
MPCS	GateMP
CHNL	Not supported
PROC	ProcMgr
CFG_<PLATFORM>.c	Ipc config, slave-specific info read from RTOS-side.
PROC_setup (DSPLink setup)	Ipc sync
PROC_read, PROC_write	ProcMgr_read, ProcMgr_write
mem=<> boot args	mem=<> boot args (current), System Memory Manager (future)
-	Utils like NameServer, MultiProc
-	FrameQ IPC

Architectural differences

This section details architectural differences between SysLink and DSPLink.

DSPLink architecture supports the following types of devices and configurations:

- GPP-DSP devices such as DM6446, OMAP3530
- Discrete DSPs connected to host over links such as PCI, VLYNQ
- One or more slave processors connected to one master processor
- The master may run any HLOS (e.g. Linux, WinCE, QNX, PrOS) and the slave must run DSP/BIOS

SysLink architecture enables support for the following features, which are not possible in DSPLink:

- SysLink/IPC can be used in devices that do not have master-slave configuration, e.g. multi-core DSP devices such as TCI6488.
  - In BIOS 5.x/DSPLink, the only available IPC for people to use is the DSP/BIOS MSGQ, with custom Message Queue Transports (MQTs)
- SysLink/IPC can be used in configurations where both master & slave are running SYS/BIOS
  - In BIOS 5.x/DSPLink, DSPLink provides the MSGQ and other such module implementations on GPP-side.
  - If DSP/BIOS is also run on the GPP, the DSP/BIOS MSGQ conflicts with the DSPLink MSGQ. Other such modules also have similar issues (e.g. CHNL vs. SIO)
  - Hence DSPLink porting for DSP/BIOS on the GPP cannot be done by just porting the OSAL, but requires significant architectural changes. In such requirements, it has always been advised to simply go ahead and use the DSP/BIOS MSGQ with a custom written MQT, instead of using DSPLink.
  - SysLink/IPC can be used in such configurations without any issues.
- SysLink can be used in configurations where both processors are running an HLOS
  - DSPLink only provides communication between master (HLOS) and slave (DSP/BIOS). It does not support configurations or IPCs where multiple processors may be running HLOS.
  - In DSPLink, the IPCs are not independent of each other. For example, if a use-case only needs MSGQ for a different physical link than is supported in the standard product, the entire DSPLink needs to be ported,

including PROC module to enable MSGQ to be used.

  - In SysLink, the IPCs are independently configured and interact with each other at the interface level. ProcMgr for device management is an independent module without dependence on any IPCs. Similarly, IPCs can be used independent of ProcMgr.
- In DSPLink, the IPCs depend on the master processor for initialization. They cannot work intra-processor on the slave seamlessly. They require workarounds or shared memory initialization from the master to be able to be used on the slave.
  - In SysLink/IPC, all types of configurations are supported, including inter-processor and intra-processor on HLOS or SYS/BIOS. Configuration of the IPC on all processors happens independently.
- DSPLink works very well with shared memory systems or memory mapped IO such as VLYNQ/PCI. However, completely discrete links such as ethernet, USB, serial etc. pose major difficulties, and the porting effort would involve rewriting all the DSPLink protocols.
  - Most of the existing protocol implementations cannot be used as-is.
  - The implementations would also not be easily pluggable into DSPLink
  - In SysLink/IPC, major modules such as Notify, MessageQ and NameServer have pluggable transports, which can be independently written to work over different types of physical links. In future, such implementations would also be provided as part of the default product.
- DSPLink does not provide HLOS kernel-side APIs, since all parameter validation etc. happens on the user-side
  - In the current SysLink implementation, identical kernel-side APIs are provided for all SysLink modules. The modules can be used from user or kernel-side with the same APIs.

Multiple types of software solutions are currently available for Inter-processor communication on TI devices:

- DSPLink
- DSPBridge
- MSGQ / MQTs
- Custom IPCs

The goal of SysLink is to combine the requirements from all these different usage scenarios, into a single product that everyone can use for all IPC and device management needs. Due to this, the SysLink product must meet the needs of existing DSPLink customers, as well as those of DSPBridge. SysLink must provide a super-set of essential features and functionality as is present in these existing products. In addition, the SysLink product, in conjunction with the BIOS-side IPC product is also expected to meet the needs of existing users of DSP/BIOS Message Queue Transport products on DSP-only multi-core devices not covered by DSPLink or DSPBridge.

Build differences

HLOS-side

DSPLink is built with a custom build system that is present within the 'make' folder in the DSPLink package.

SysLink uses kconfig for building the kernel-side and a simple make-based build system for building the user-side. All the rules for building the kernel and user-side are present within single files:

```
$SYSLINK_ROOT/ti/syslink/buildutils/hlos/knl/Makefile.inc
$SYSLINK_ROOT/ti/syslink/buildutils/hlos/usr/Makefile.inc
```

RTOS-side

DSPLink is built with a custom build system that is present within the 'make' folder in the DSPLink package.

SysLink uses xdc build for building the RTOS-side SysLink library as well as sample applications.

Interface differences

Type	DSPLink	SysLink
Product partitioning	DSPLink - DSP/BIOS product partitioning	SysLink - IPC product partitioning
HLOS modules	All the HLOS-side modules are available as part of DSPLink product	All the HLOS-side modules are available as part of SysLink product. In addition, BIOS-side FrameQ and RingIO modules are also available as part of the SysLink product
RTOS modules	NOTIFY, MPCS, MPLIST, RingIO module are part of DSPLink product. MSGQ, SIO are part of BIOS.	BIOS-side FrameQ and RingIO modules are part of the SysLink product. All others, including MessageQ, Notify, ListMP, their transports and subsidiary modules are part of IPC product
SIO/CHNL module	The SIO module on BIOS-side maps to CHNL module on HLOS-side.	Streaming based protocol mapping to issue-reclaim model is not currently present in SysLink.
API differences between HLOS and RTOS	The APIs have some differences between the HLOS and RTOS-side. For example, MSGQ APIs are not identical	For the user-interface, a header file package available as part of the IPC product in ti/ipc is used on both HLOS and RTOS-side. All modules, APIs and data types that are usable by applications are part of this common header file package. These headers are implemented on BIOS-side as part of IPC product and on HLOS-side as part of SysLink product. This ensures that the APIs and user-interface is identical.
Header files	Header files are present in gpp/inc and dsp/inc folders. User-specific headers and internal headers are mixed so application may not know which headers to use.	Header files that application uses are in the package folder. e.g. ti/syslink/ProcMgr.h and should be included by applications accordingly. Internal header files are in syslink/inc folder.
Header files for build	For exported header files also, application must give the base paths for the inc folders and all their sub-folders for building either DSPLink or applications since the headers are mixed.	For building applications, users must give only the base package path as a base for include paths, since all headers are included as relative from the base path (e.g. ti/syslink/ProcMgr.h - refer to sample apps). For building SysLink, the paths in syslink/inc and their sub-folders must be given as base.
Data types	The data types used on BIOS-side are BIOS std.h, whereas those used on HLOS-side are DSPLink-defined specific types.	BIOS-side uses types from xdc/std.h, whereas HLOS-side uses identically defined types available as part of the SysLink product in ti/syslink/Std.h. Applications must include the appropriate types header file before including any SysLink/IPC header files.

Configuration differences

Type	DSPLink	SysLink
RTOS configuration	TCF / CFG in DSP/BIOS. No DSPLink configuration on RTOS-side	CFG in IPC and SysLink
HLOS configuration	CFG_<PLATFORM>.c	Module configuration to be matched up inside platform-specific Platform.c file
Configuration passing	Configuration is taken on HLOS-side and passed to BIOS-side	Configuration is taken into RTOS-side and read from the slave target memory to configure the HLOS-side
Memory map configuration for slave executable	Memory map is configured in CFG_<PLATFORM>.c for HLOS and tcf/platform_<PLATFORM>.xs for BIOS-side	Memory map for slave executable is using Platform.xdc on BIOS-side and HLOS-side configures itself by doing the required mappings when loading BIOS code and data through the ProcMgr loader.
Memory map configuration for shared memory	Memory map is configured in CFG_<PLATFORM>.c (DSPLINKMEM, DSPLINKMEM1 regions) for HLOS and tcf/platform_<PLATFORM>.xs for BIOS-side	Memory map for shared memory is using cfg file on BIOS-side (SharedRegion configuration) and HLOS-side configures itself by reading the configuration information for SharedRegion mappings from the slave target memory. No matching up is required on HLOS and RTOS.

Module differences

PROC -> ProcMgr + Ipc

The PROC module in DSPLink is responsible for device management of the slave DSP as well as for overall DSPLink setup and configuration of the IPC modules. It is also responsible for synchronization between the GPP and DSP sides of DSPLink.

In SysLink, the ProcMgr module is responsible for device management of the slave processor. The Ipc module is responsible for overall SysLink setup and configuration, and for synchronization between the IPC modules between each pair of processors.

Functionality	DSPLink	SysLink
Overall setup	PROC_setup does overall DSPLink setup of all DSPLink modules from the calling process	On HLOS-side, SysLink_setup does overall SysLink setup of all modules from the calling process. On RTOS-side, internal module startups take care of overall IPC setup.
Configuration of all modules	PROC_attach does configuration of all modules	On HLOS-side, SysLink_setup and Ipc_control (Ipc_CONTROLCMD_LOADCALLBACK), which internally calls Ipc_start, do configuration of all modules. On RTOS-side, Ipc_start does configuration of all modules.
Hardware configuration and power up for the slave processor	PROC_attach	ProcMgr_attach
Loading the slave with executable from HLOS file system	PROC_load	ProcMgr_load
Starting the execution of slave processor from its entry point	PROC_start	ProcMgr_start
Synchronizing IPC modules between each pair of processors	PROC_start	On HLOS-side, Ipc_control (Ipc_CONTROLCMD_STARTCALLBACK), which internally calls Ipc_attach. On RTOS-side, Ipc_attach.

Stopping the execution of slave processor by placing it in reset	PROC_stop	ProcMgr_stop
Cleanup synchronizing IPC modules between each pair of processors	PROC_stop clears the synchronization information	On HLOS-side, Ipc_control (Ipc_CONTROLCMD_STOPCALLBACK), which internally calls Ipc_detach and Ipc_stop. On RTOS-side, Ipc_detach.
Power down for the slave processor	PROC_detach	ProcMgr_detach
Overall cleanup	PROC_destroy	SysLink_destroy
Read from slave memory	PROC_read	ProcMgr_read
Write into slave memory	PROC_write	ProcMgr_write
Get the state of the slave processor	PROC_getState	ProcMgr_getState
Map to slave MMU (if present)	PROC_control	ProcMgr_map (generic API for all mappings)
Get address of symbol in slave executable	PROC_GetSymbolAddress	ProcMgr_getSymbolAddress
Loader type	COFF, NO_LOADER, BINARY, STATIC	ELF (default), COFF, if no load/start is needed, corresponding ProcMgr APIs don't need to be called.

MSGQ -> MessageQ

The MSGQ module in DSPLink is responsible for exchanging messages of variable length between the clients on one or more processors. In SysLink, the MessageQ module provides the same features as the MSGQ module in DSPLink

Functionality	**DSPLink**	**SysLink**
API names	MSGQ_<API name>	MessageQ_<API name>
Module location	HLOS-side MSGQ module in DSPLink and BIOS-side MSGQ module in BIOS	HLOS-side MessageQ module in SysLink and BIOS-side MessageQ module in IPC product
Module configuration on BIOS	Is not configured using standard TCF/CFG, except for basic useMSGQ flag. All actual configuration happens through an instance of type MSGQ_Config called MSGQ_config that must be built with the BIOS executable	Configuration happens in the same way as all other modules in BIOS/IPC, using XDC config
MessageQ instance creation	MSGQ_open	MessageQ_create
Opening a handle to a MessageQ instance on local/remote processor	MSGQ_locate	MessageQ_open
Memory managers used for messaging	On BIOS-side, POOL_config instance of type POOL_Config must be built in with the BIOS executable. IDs thus defined through the array of pools are used for passing to MSGQ_alloc	Heap handle must be passed to MessageQ_registerHeap API through arbitrarily defined (but matching on all processors) heapId. This heapId is then used for passing to MessageQ_alloc
Transports configuration on RTOS	Must be explicitly configured through MSGQ_Config structure	Internally automatically created and linked up based on platform being used.
Transports configuration on HLOS	Must be explicitly configured by calling MSGQ_transportOpen/Close APIs	Internally automatically created and linked up based on platform being used.

Local messaging	Available using MSGQ module	Available using MessageQ module
Readers/Writers	Single reader, multiple writers	Single reader, multiple writers

NOTIFY -> Notify

The NOTIFY module in DSPLink abstracts physical hardware interrupts into multiple logical events. It is a simple and quick method of sending up to a 32-bit message. In SysLink/IPC, the Notify module provides the same features as the NOTIFY module in DSPLink

Functionality	**DSPLink**	**SysLink**
API names	NOTIFY_<API name>	Notify_<API name>
Module location	HLOS-side and BIOS-side NOTIFY module in DSPLink	HLOS-side Notify module in SysLink and BIOS-side Notify module in IPC product
Module configuration on BIOS	Is not configured using standard TCF/CFG. Configuration happens from HLOS-side and is sent to BIOS-side as part of overall DSPLink configuration exchange mechanism.	Configuration happens in the same way as all other modules in BIOS/IPC, using XDC config
Support for multiple physical interrupts	Identified through IPS ID in APIs and configured as a separate IPS instance	Identified through lineId in APIs and configured automatically internally depending on platform being used.
Register for Notify event	NOTIFY_register	Notify_registerEvent, Notify_registerEventSingle. The Notify_registerEventSingle API is used when only a single client will ever register for the eventId, and gives a faster path, as well as returns error if additional registrations are attempted for this event ID.
Unregister Notify event	NOTIFY_unregister	Notify_unregisterEvent, Notify_unregisterEventSingle
Send Notify event	NOTIFY_notify	Notify_sendEvent
Notify callback function signature	Parameters are eventNo, arg (provided during register) and info (payload)	Parameters are procId, lineId, eventId, arg and payload.
Separate module initialization	Not possible. Done as part of DSPLink	Can be used and initialized separately, independent from the rest of SysLink (if other modules are not required) by using Notify_attach/detach directly.
Disable all notifications	Not available in DSPLink	Notify_disable disables receiving all notifications for a particular procId-lineId combination. Equivalent to disabling the physical interrupt line. Notify_sendEvent called when other side has disabled notifications gives an error on sending processor.
Restore all notifications	Not available in DSPLink	Notify_restore restores receiving all notifications for a particular procId-lineId combination. Equivalent to enabling the physical interrupt line.
Disable notifications for a specific eventId	Not available in DSPLink	Notify_disableEvent disables receiving notifications for a specific procId-lineId-eventId combination. Notify_sendEvent called for this procId-lineId-eventId when other side has disabled notifications gives an error on sending processor.
Enable notifications for a specific eventId	Not available in DSPLink	Notify_enableEvent enables receiving notifications for a particular procId-lineId-eventId combination.
Reserved events	Some events are used internally by DSPLink modules, and checks are done internally if application attempts to register for any of these events	Concept of reserved events is available. A configurable number of events can be specified as reserved events. When using these events, a special mask needs to be applied to the event ID before calling the APIs. This ensures that applications cannot inadvertantly use these reserved events.

Check to see if event is available	No API that returns information whether event is available to be used by applications	Notify_eventAvailable returns information that indicates to applications whether the specified eventId is free for their use.
Check number of interrupt lines available	No run-time API that returns information about number of available interrupt lines	Notify_numIntLines returns information about number of available interrupt lines
Check whether interrupt line is registered	No run-time API that returns information about whether specified interrupt line is registered	Notify_intLineRegistered returns information about whether specified interrupt line is registered.
Local notifications	Not available in DSPLink	Available using Notify module
Readers/Writers	Multiple readers, multiple writers	Multiple readers, multiple writers

RingIO -> RingIO

The RingIO module provides Ring Buffer based data streaming, optimized for audio/video processing. In SysLink/IPC, the RingIO module provides the same features as the RingIO module in DSPLink

Functionality	**DSPLink**	**SysLink**
API names	RingIO_<API name>	RingIO_<API name>
Module location	HLOS-side and BIOS-side RingIO module in DSPLink	HLOS-side and BIOS-side RingIO module in SysLink
Module configuration on BIOS	Is not configured using standard TCF/CFG. Configuration happens from HLOS-side and is sent to BIOS-side as part of overall DSPLink configuration exchange mechanism.	Configuration happens in the same way as all other modules in BIOS/IPC, using XDC config
Creation of RingIO instance	RingIO_create returning status	RingIO_Params_init for initialization of parameters with default values. RingIO_create for creation of the instance. It returns handle, NULL handle means failure. The handle returned from create is not used in applications, and the one returned from RingIO_open is used.
Opening of RingIO instance	RingIO_open returning handle, which is used for all further RingIO APIs	RingIO_open to open by name, or RingIO_openByAddr to open by shared address (does not use NameServer and is faster if the sharedAddr is known. Both APIs return status. Handle returned as return parameter is used for all further RingIO APIs.
Set notification	RingIO_setNotifier for both registering and unregistering notifications	RingIO_registerNotifier for registering notifications and RingIO_unregisterNotifier for unregistering notifications.
Notification types	5 types: RINGIO_NOTIFICATION_NONE, RINGIO_NOTIFICATION_ALWAYS, RINGIO_NOTIFICATION_ONCE, RINGIO_NOTIFICATION_HDWRFIFO_ALWAYS, RINGIO_NOTIFICATION_HDWRFIFO_ONCE	The same 5 types: RingIO_NOTIFICATION_NONE, RingIO_NOTIFICATION_ALWAYS, RingIO_NOTIFICATION_ONCE, RingIO_NOTIFICATION_HDWRFIFO_ALWAYS, RingIO_NOTIFICATION_HDWRFIFO_ONCE
Send force notification	RingIO_sendNotify	RingIO_notify
Acquire/Release data	RingIO_acquire/RingIO_release	RingIO_acquire/RingIO_release
Set/get fixed size attribute	RingIO_setAttribute, RingIO_getAttribute	RingIO_setAttribute, RingIO_getAttribute
Set/get variable size attribute	RingIO_setvAttribute, RingIO_getvAttribute	RingIO_setvAttribute, RingIO_getvAttribute
Setting attribute at any offset	Available in DSPLink	Not available in SysLink. Attribute is set only at offset 0 for simplication of RingIO attribute logic.

Cancellation of acquired RingIO buffer	RingIO_cancel	RingIO_cancel
Hard/soft flush of RingIO released data	RingIO_flush	RingIO_flush
Helper APIs for RingIO to get RingIO status information	Helper APIs for RingIO_getValidSize, RingIO_getEmptySize, RingIO_getValidAttrSize, RingIO_getEmptyAttrSize, RingIO_getAcquiredOffset, RingIO_getAcquiredSize, RingIO_getWatermark	Helper APIs for RingIO_getValidSize, RingIO_getEmptySize, RingIO_getValidAttrSize, RingIO_getEmptyAttrSize, RingIO_getAcquiredOffset, RingIO_getAcquiredSize, RingIO_getWatermark
Local RingIO on RTOS-side	Possible in DSPLink, but needs special gel file to initialize some modules from HLOS-side	Available using RingIO module on RTOS-side without any special changes
Local RingIO on HLOS-side	Not possible in DSPLink	Available using RingIO module on HLOS-side without any special changes.
RingIO implementation on HLOS-side	RingIO implementation on user-side in HLOS	RingIO implementation in kernel-side on HLOS and exposed through user APIs.
Readers/Writers	Single reader, single writer	Single reader, single writer

Migration FAQs

- Will SysLink APIs be backward compatible to DSPLink APIs?
  - No. Due to significant changes in architecture, configuration approach and modules, the SysLink APIs will not be backward compatible to DSPLink APIs.
- Do application writers also need to build with XDC tools?
  - No, only configuro needs to be used to configure IPC on RTOS-side. Makefiles can be used for the application build. HLOS-side shall provide non-*XDC based configuration.
- Will SysLink/IPC take care of all IPC scenarios such as multi-core DSP (TCI6488), heterogenerous multi-core (OMAP3530)?
  - Yes, it is architected to take care of all such scenarios.
- Can I use SysLink with BIOS 5.xx?
  - No, SYS/Bios 6.xx must be used.
- My s/w stack is based on DSPLink with WinCE.Can I port SysLink to other OS like WinCE?
  - Yes, SysLink product is also available as a porting kit similar to DPSLink. The presence of an OS Abstraction Layer makes it easy to port to other Operating Systems. While the first product release of SysLink is on Linux, it is planned to make ports of SysLink to other OSes such as WinCE also available in the future.
- My s/w stack is based on DSPLink with DM6446. Can I port SysLink to other platforms like DM6446 etc?
  - It is possible to port SysLink to devices such as DM6446. The existing port of SysLink, similar to DSPLink, only requires shared memory and h/w interrupts between the pair of cores to be connected for IPC. It is possible to port the relevant SysLink modules to make it work on DM6446 also.
- I already have a s/w stack on DSPLink. Why should I move to SysLink?
  - Please see the above sections in this document: Migration from DSPLink, especially the section on Architectural differences. It explains why architecturally, DSPLink does not support porting to some of the new multi-core devices such as TI81XX. In addition, these sections also describe some of the other design as well as functional improvements in SysLink over DSPLink.
- What is the performance difference between the two?

  - In profiling the common modules between DSPLink and SysLink on Linux, the performance between the two is seen to be comparable, with SysLink giving slightly better performance.
- Can DSPLink and SysLink co-exist?
  - Since DSPLink and SysLink compete for the same hardware resources, it is not possible for both DSPLink and SysLink to coexist for communication between the same pair of processors. However, it is possible to use SysLink between some of the processors, and DSPLink between any one pair. For example, on TI81XX, it is possible to use DSPLink between the ARM Cortex A8 <-> DSP, while SysLink is used between ARM Cortex A8 <-> Video-M3, ARM Cortex A8 <-> VPSS-M3, Video-M3 <-> VPSS-M3.
  - However, note that if used in such a configuration, it is not possible to use SysLink/IPC for communication between Video-M3/VPSS-M3 <-> DSP, since DSPLink only works with SYS/BIOS 6 legacy layer, whereas SysLink/IPC use the new modules added in SYS/BIOS 6 for IPC.
  - Also, note that since the system heap would only work between the processors controlled by SysLink, any buffers allocated using system heap cannot be freed on DSP, and similarly any buffers allocated using POOL module in DSPLink on the DSP/ARM Cortex A8 cannot be freed by any other processors where POOL module is not supported.
- Will you stop supporting DSPLink now that SysLink is available.
  - There are no plans as of now to stop support for DSPLink.

Article Sources and Contributors

SysLink MigrationGuide *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=75517 *Contributors*: MugdhaKamoolkar

Image Sources, Licenses and Contributors

Image:TIBanner.PNG *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:TIBanner.PNG *License*: unknown *Contributors*: SekharNori