EXIMIUS
Excellence Everyday

IP User Guide

Logic Link (AXI-MM and AXI-ST)

Revision: 0.9.2

17 Dec 2021

Table of Contents

Eximius Design, Copyright 2021

Eximius Design, Copyright 2021

Eximius Design, Copyright 2021

Table of Figures

Eximius Design, Copyright 2021

Table of Tables

Eximius Design, Copyright 2021

Revision History

Revision	Date	Author	Summary of Changes
0.1	8/20/21	John	Initial draft
0.2	11/10/21	John	Added information about Asymmetric AXI-ST without Valid and without Ready.
0.9	12/12/21	John	Expanded on Tiered Logic links. Added details to packetization. General cleanup, wordsmithing
0.9.2	12/17/21	John	Documented new debug_status bits and their use in multi-tiered logic. Documented effects of USER inserted strobes/markers on Auto Synchronization

Table 1 Revision History

Eximius Design, Copyright 2021

1. Overview

The Logic Link (LLINK) IP is used to provide an interface between AMBA AXI Protocol (AXI-MM) and AMBA AXI-ST interface to interface to the AIB PHY and AIB Adaptors. The LLINK takes care of:

- Bundling the AXI data and packetizing
- Managing AXI flow control across the AIB link
- Interspersing AIB overhead signals (Markers, Strobes, DBI bits, etc).
- Support for USER inserted Markers / Strobes, both persistent and recoverable.

Most of this functionality is achieved via the llink_gen.py script that brings in configuration files to outline the various features. In addition to the common features mentioned above, there are higher level features available that enable more exotic configurations and better use of the AIB interface. Those features are briefly listed in the following sections.

1.1 Optional Ready / Valid

This allows the user to remove flow control or squeeze additional data out of the line by dropping the Valid bit. This is only available in AXI-ST.

1.2 Packetization

This allows the AXI-MM data to be packetized into smaller chunks, sent across the AIB line and then reassembled on the far side. This allows for wider AXI-MM and AXI-Lite interfaces to be transmitted over a narrower AIB interface, making the most of the available AIB bandwidth.

1.3 Dynamic Gen2/Gen1 Switching

This allows for an AXI-ST Logic Link to exist in both Gen2 and Gen1 operating ranges for maximum interoperability.

1.4 Asymmetric Gearboxing

This option allows for an AXI-ST Logic Link operating at Full, Half or Quarter Rate and communicate with a remote side operating at Full, Half or Quarter Rate, but not necessarily at the same rate as the near side, allowing for greater interoperability.

1.5 Summary

Note that not all features are supported in all protocols. The below table outlines which features are supported in which AXI protocol.

Protocol	Optional READY/VALID	Packetization	Dynamic Gen2/Gen1 Switching	Asymmetric Gearboxing
AXI-MM (and AXI-LITE)	no	yes	no	no
AXI-ST	yes	no	yes	yes

Table 2 AXI-ST VALID / READY Options

Eximius Design, Copyright 2021

2. LLINK Overview / IP

Each Logic Link is a construct suitable for a single AXI channel. The basic structure is shown below in situ with Channel Alignment and AIB blocks.

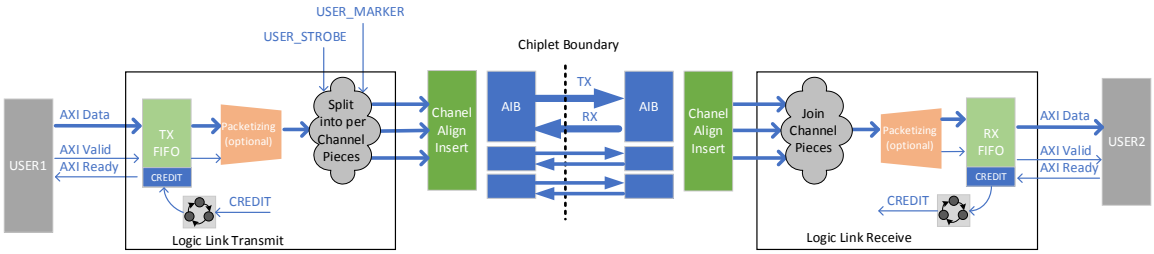


Figure 1 Logic Link Overview

An AXI-ST interface will use one Logic Link, while a standard AXI-MM will use 5 Logic Links. Both sets of IP are built from the same basic script and underlying RTL components. However, the script will generate RTL code unique to the configuration of the AXI and AIB channels. The following will outline the common features of the script, and those targeted at AXI-ST and AXI-MM.

At its lowest level, each Logic Link consists of some basic FIFOing for flow control, some logic to control credit returns and some logic to split and merge the Logic Link data (e.g. AXI) onto the AIB data channel with the various AIB overhead.

A traditional AXI Valid/Ready scheme requires instantaneous response, which is impractical across an AIB interface, so this is converted to a credit based scheme that can handle AIB multicycle latencies. The Transmit Credits are generally equal to the remote side's RX FIFO Depth, which ensures that all data sent from one chiplet necessarily has a place to land in the remote chiplet, even if the remote USER side asserts a prolonged "not Ready" flow control. If the transmit path runs out of credits, then data will be stuck in the TX FIFO and flow control will be asserted upstream.

On the receive side, credits are replenished every time an entry is transferred to the downstream AXI interface and is popped off of the RX_FIFO. These credits are then sent to the transmit side using the standard AIB data interface. So it is common for some AXI interfaces to have a large amount of data going from one chiplet to another, but only have a single bit of credit data going in the reverse direction.

Due to the round trip nature of the credit replenishment, the depth of the RX FIFO should be sized so according to expected data traffic patterns and round-trip latency numbers. So an AXI interface that wants to make full use of the AIB bandwidth should have an RX_FIFO equal to a little more than the round trip latency, while an AXI interface that does not necessarily need the full bandwith could make do with a less deep RX_FIFO. The system will operate correctly with any RX_FIFO depth including a depth of one, but max performance can only be guaranteed with RX_FIFO depth being greater than the round trip latency.

2.1 Credit Analogy

The above credit scheme and round trip latencies are relatively standard networking credit flow control functionality. The following analogy is offered for users unfamiliar or uncomfortable with this functionality. Those familiar with these concepts can skip to the next section.

A reasonable analogy is to imagine a shipping company using airplanes to transfer cargo. The planes take an hour to reach their destination, and then are unloaded by cargo crews at the far terminal. Normally, the crew can unload the plane onto nearby trucks in about 10 minutes, but if the truck is late then the cargo crew may take longer. Once they are unloaded, a message is sent back to the original facility informing the destination runway is available for another flight.

Using this analogy, each beat of data on the AIB bus is a single planeload of cargo containing AXI data. Each individual landing runway is an entry in the RX FIFO, and permissions to land at that airport are the TX

Eximius Design, Copyright 2021

Credits. The trucks driving the cargo away is the USER side interface and its potential backpressure which can slow down the system.

So, due to an abundance of safety, the local airport control tower will not let the plane take off unless we are certain there is a runway for it to land on. If the tower gives clearance to takeoff, but for some reason the runway is not available by the time arrives, the plan will run out of fuel and crash, which is similar to the AXI data being lost because of an unforeseen slowdown in the egress from the RX FIFO and without a place for the data to safely land, it will be lost. So, the tower prevents planes from launching until there is TX Credits ensuring there is space for the plane to land.

Once a plane lands at the remote airport, it will sit there until the local crews unload the cargo and pull it off the runway. This is similar to the data sitting in the RX FIFO until the USER interface pulls it out of the FIFO an onto the remote AXI interface. Once this happens, the indication that the runway is free is returned to the original airport. This is the credit flow back to the transmitter. This message takes some time to return to the original airport.

Now, if the company only uses one remote runway, this reduces the cost overhead of maintaining multiple runways, which is similar to smaller area by using a 1 deep RX FIFO. But, in this case each plane must wait for the entire round trip of the previous plane before it can take off. This round trip is the plane taking off from the local runway, landing at the remote site, unloading of cargo, and returning of the message saying the runway is available. In the above example, this takes 60+60+10=130 minutes, which means the max throughput possible is 1 cargo unit per 130 minutes but could be slower if the remote cargo crew is delayed for any reason.

If the company uses two remote runways, then this increases the cost but doubles the max throughput to approximately 1 cargo unit per 65 minutes. Following this pattern, if the company uses 100 remote runways, then you may assume the throughput would be 1 cargo / 1.3 minutes, but since the cargo crew needs 10 minutes to unload, the actual throughput will be 1 cargo every 10 minutes. So at some point, extra runways will not increase the max throughput. But note that having more runways can speed up recovery from prolonged delays. For example, if cargo crew comes down sick for a day and stop unloading the planes, the planes will continue to land and there could be as many as 100 runways worth of cargo ready when the cargo crew returns to work.

In this example, the theoretical optimal number of runways to achieve the max throughput with is 13 runways. This number arrives from the number of planes in the air, and number of return messages on the ground to keep the ground crew busy. In practice, you add a little more to accommodate various unforeseen small slowdowns like unexpected headwinds. This is usually a small overhead, something like 5%. For this example, we'll say a total of 14 runways to include the extra.

In the above example, we've been targeting the max throughput, i.e. one piece of cargo every 10 minutes. But it possible that the client of the shipping company doesn't need that throughput. They may only be able to produce one piece of cargo every 20 minutes. In this case, the 14 runways are not needed, but do not hurt to have. However, if you wanted to optimize the route and reduce cost (size), then the general formula is the same, i.e. Rate of Cargo / RoundTripTime <= NumberOfRunways. So for a client producing a unit of cargo every 5 minutes, it would be (1 cargo/ 20 min) * 130 min <= 6.5 or rounding up maybe 7 runways.

For the AIB/AXI system, the above analogy holds. The ideal number of RX FIFO depth is equal to the round trip of the system plus a "little" to accommodate a modest overhead. The actual round trip time comes from the various blocks in the system, such as the LLINK(s), CA(s) and AIB PHYs, as well as implementation details like inter-channel skew.

2.2 Round Trip Delays RX FIFO Sizing

The actual latency of the system is highly dependent on the configuration of the system, and this latency has a relationship to the RX FIFO depth. We'll talk about those here. Note that LLINK and CA cycles are in terms of wr_clk, which is expressed in terms of 1x, 2x or 4x the FWD clock used by the AIB channel.

2.2.1 **LLINK**

The base LLINK is entirely combinatorial and has no cycles of latency for TX and RX. Adding flow control, which is optional for AXI-ST but required for AXI-MM, adds 1 wr_clk cycle TX and 1 wr_clk cycle RX latency.

An additional and optional timing FF can be enabled for the LLINK interfaces with the CA/PHY. This can add another 1 wr_clk cycle TX and 1 wr_clk RX latency.

Packetization confuses this discussion, so it is sufficient to say that the packetization does not affect the latency in terms of FIFO depth. Packetization inherently adds latency, for example it could take 2 beats of AIB data to send a single AXI transaction which is obviously more latency than 1 beat. However, because these 2 beats of AIB data only consume a single entry in the RX_FIFO, the effects of packetization on the RX_FIFO depth calculating round trip are negated. That is, for every cycle of additional latency added by packetization, we necessarily use one less entry of RX_FIFO, so the result is no effect.

2.2.2 Channel Alignment

The CA has a single cycle of wr_clk on TX. On the RX, the CA adds a cycle of wr_clk as overhead, but this is also where the inter channel deskew happens. All RX channels are delayed by the worst case actual inter channel skew. That is if two channels have varying inter channel skew, the CA will align them so that both have the worst case skew.

The inter channel deskew is considered modest, and is estimated to be less than 5 cycles FWD clock.

2.2.3 AIB

The AIB latency is dependent on the Rate. The latency shown below is for the MAC to MAC latency, i.e. the MAC interface on the TX through the TX portion of one AIB PHY, through the RX portion in another AIB PHY and to the MAC interface on the RX. All values are expressed in FWD clocks.

Full Rate: Min 6, Max 6

Half Rate: Min 16, Max 20

Quarter Rate: Min 22, Max 28

Asymmetric modes have the slowest Rate’s latency.

2.2.4 Recommended RX_FIFO Depth Values

For a standard, AXI-MM (or AXI-ST with flow control), no additional timing constraints and no expected abhorrent behavior, then a FIFO depth for Full Rate to achieve maximum bandwidth could reasonably be: 32 deep. This is derived from this equation:

- LLINK TX + RX = 2
- CA TX + RX = 2
- Inter Channel Skew = 5
- AIB = 6
- Total one way latency = 15
- Multiple by two for round trip = 30
- Add a small overhead = 32

Similarly for Half Rate, this would be 36 deep (in terms of wr_clk)

- LLINK TX + RX = 2
- CA TX + RX = 2
- Inter Channel Skew = 5 / 2 for half rate wr_clk
- AIB = 20 / 2 for half rate wr_clk
- Total one way latency = 17
- Multiple by two for round trip = 34
- Add a small overhead = 36

And for Quarter Rate, this would be 28 deep:

- LLINK TX + RX = 2
- CA TX + RX = 2
- Inter Channel Skew = 5 / 4 for quarter rate wr_clk

Eximius Design, Copyright 2021

- AIB = 28 / 4 for quarter rate wr_clk
- Total one way latency = 13
- Multiple by two for round trip = 26
- Add a small overhead = 28

2.2.5 Recommended TX_FIFO Depth Values

One. The short answer is 1 deep is the recommended value.

There are a limited set of systems that may benefit from a deeper TX_FIFO, allowing the AXI transaction to be pulled off of the shared AXI bus, but generally the FIFOing is better spent in the AXI fabric, than in the peripheral.

2.2.6 Over advertising Credits

Over advertising credits is a dangerous networking trick. In theory, if the remote side will practically never assert flow control, the user could try to get away with a smaller FIFO depth and a larger advertisement of initial TX Credits. This could allow max throughput in the system with a smaller RX FIFO depth, if flow control does get asserted, the data will be lost so it is not recommended.

Going back to the analogy above, this is similar to having 6 planes in the air, all destined for 1 runway. As long as the ground crew work at their ideal pace of 10 minutes to unload a plane, they should empty the plane's cargo before the next plane lands. But if the crew is delayed, one or more planes may have no place to land and will crash, destroying the cargo.

This is generally not a recommended flow. But if it is to be supported, a non-zero value can be driven into the LLINK's init_credit port for each Logic Link, which will override the default credit value which is equal to the remote side's RX FIFO Depth.

3. Script Overview

The main script can be downloaded from https://github.com/chipsalliance/aib-protocols. It is located in llink/script and the main script is called llink_gen.py. This is a python script, so it requires a python executable to run. Python comes in two major flavors, Python2 and Python3, and the script is intended works for both flavors. Most linux based OS come with python and it can be downloaded for Windows. This website may help if you need to download python for your OS: https://www.python.org/downloads

You invoke the script by pointing to the configuration file. For example:

python ../llink/script/llink_gen.py --cfg cfg/axi_st_d64.cfg

This command as-is will pick up the configuration file specified and by default will put the resulting RTL in a local directory based on the MODULE variable name in the config file (in this example the directory will be called ./axi_st_d64).

3.1 Script Output

Generally, the script generates an INFO file containing details about the configuration and bit placement on the AIB lines, and two sets of corresponding RTL, one master and one slave, implementing the logic link. The name of the files and Verilog module names is specified by the MODULE variable in the config file. The outputs of the script are listed below.

Note that the .f files use the environment variable PROJ_DIR which is expected to point to the aib-protocols directory level (i.e. one level above the llink directory).

For Each LL	Description
<MODULE_NAME>_info.txt	Text based information file. This contains detailed information on the channel configuration, usage, AIB

Eximius Design, Copyright 2021

	overhead and specific locations of key signals in the AIB channel. This should be the first stop to looking at the RTL and debugging any issues. This file, like much of the rest of the design, is written from the Master's perspective. So TX signals refer to the master's TX, and is necessarily the Slave's RX. Similarly RX signals refer to the Master's RX is necessarily the Slave's TX.
<MODULE_NAME>_master.f	Verilog list file for the master. Relative file paths are prepended with the environment variable ${PROJ_DIR}
<MODULE_NAME>_master_top.sv	Top level master module
<MODULE_NAME>_master_name.sv	master side submodule interfacing to the USER side.
<MODULE_NAME>_master_concat.sv	master side submodule interfacing to the PHY/CA side.
<MODULE_NAME>_slave.f	Verilog list file for the slave. Relative file paths are prepended with the environment variable ${PROJ_DIR}
<MODULE_NAME>_slave_top.sv	Top level slave module
<MODULE_NAME>_slave_name.sv	slave side submodule interfacing to the USER side.
<MODULE_NAME>_slave_name.sv	slave side submodule interfacing to the PHY/CA side.

3.1.1 Asymmetric Variant

The normal LLINK script output is described above and the script will generate a single set of Master / Slave IP. However, for LLINK configurations that support Asymmetric Gearboxing, the script will generate multiple sets of Masters that are compatible with a set of Slaves, allowing the USER to chose from a compatible set of RTL. The names will follow the above examples, but _full, _half or _quarter (Gen2 only) will be postpended to <MODULE_NAME> indicating the configurated rate.

3.2 Script Options

The script has several command line options. Only the configuration file is a require input, the rest are optional.

For Each LL	Values	Description
--cfg	CONFIG.cfg	Point to the targeted configuration file.
--odir	directory	This controls the output directory to place the generated Verilog RTL and .f files. If excluded, the default is ./<MODULE_NAME> which is defined in the config file.
--cfg_debug	none	Enable Configuration Debug. Internal Debug.

Eximius Design, Copyright 2021

--signal_debug	none	Enable Signal Debug. Internal Debug.
--packet_debug	none	Enable Packetization Debug. Internal Debug.
--sysv_indexing	True, False	If True, indices will be indicated using the System Verilog form of [lsb +: width]. This can be useful to immediately identify the width of fields and compare two fields to confirm they are the same width. This is the default. If False the traditional indices are used of the form [msb:lsb]. For example a 24 bit field starting at bit location 8 will look like [8+:24] in System Verilog Indexing or [31:8] in traditional mode. Both modes are synthesizable, so this a purely human aesthetic choice. Current default is True, but this can be changed in the script.

Table 3 Command Line Options

3.3 Example invocations

This is a simple python invocation

python ../llink/script/llink_gen.py --cfg cfg/axi_st_d64.cfg

This version explicitly is using python3 and specifying the output directory to be ../dut/my_out_dir.

python3 ../llink/script/llink_gen.py --cfg cfg/axi_st_d64.cfg --odir../dut/my_out_dir

This version includes piping the debug information into a file for post run analysis

python ../llink/script/llink_gen.py --cfg cfg/axi_mm_a32_d64.cfg --packet_debug > debug_output.txt

4. Common LLINK Parameters

Much of the LLINK configuration is not protocol specific and are used by all protocols. Those are explained below. Note that some fields are not strictly required. For example, if the TX_STROBE_ENABLE is False, there is no need to specify the details of the Strobe placement, User Strobe fields, etc. But there is no harm to explicitly call out all fields, which is the recommendation for clarity.

4.1 Channel Configuration

These options configure the basic channel information including number of channels, Rates, etc.

Module Name	Values	Description
MODULE	Any	Name of the generated RTL. By convention, we name the .cfg the same as the module name. This is the value used for <MODULE_NAME>.
Channel Description		

Eximius Design, Copyright 2021

NUM_CHAN	1-24	Number of AIB Channels
CHAN_TYPE	Gen1Only, Gen2Only, Gen2, Tiered	Indicates the type of AIB Channel. Note that the LLINK is agnostic if the Gen1 is going to an AIB 1.0 or AIB 2.0 PHY. Gen1Only – Configure for Gen1. Gen2Only – Configure for Gen2 Gen2 – Configure for Gen2 with option for dynamically switching to Gen1. Tiered – Used for Tiered Logic Links (see Advanced Topics)
TX_RATE	Full, Half, Quarter	Master Transmit Rate (and Slave Receive Rate). Note that Quarter Rate is not available in Gen1. Note too that this field is a don't care if SUPPORT_ASYMMETRIC is set to True
RX_RATE	Full, Half, Quarter	Master Receive Rate (and Slave Transmit Rate) Note that Quarter Rate is not available in Gen1. Note too that this field is a don't care if SUPPORT_ASYMMETRIC is set to True
TX_DBI_PRESENT	True, False	Master Transmit DBI Enable (and Slave Receive DBI Enable). If True, then the Logic Link will be configured to drive 0s in the positions used for DBI (e.g. bit 38, 39, 78, 79, etc). If False, the Logic Link may use these bits as data.
RX_DBI_PRESENT	True, False	Master Receive DBI Enable (and Slave Transmit DBI Enable). If True, then the Logic Link will be configured to drive 0s in the positions used for DBI (e.g. bit 38, 39, 78, 79, etc). If False, the Logic Link may use these bits as data.
TX_REG_PHY	True, False	If set to True, both the Master and Slave transmit paths will have a FF stage between the Logic Link output and the AIB PHY input (or CA input). This will mitigate timing paths, but obviously adds a cycle of latency.
RX_REG_PHY	True, False	If set to True, both the Master and Slave receive paths will have a FF stage between the output of the AIB PHY (or CA) and the input of the Logic Link. This will mitigate timing paths, but obviously adds a cycle of latency.

Table 4 Module Name and Channel Description Configuration

4.2 Strobe

These configure the Strobe functionality as controlled from the Logic Link. Note that these configures what the Logic Link places on the Strobe location, but it is possible downstream blocks like the Channel Alignment may override the values coming out of the Logic Link. There is generally one Strobe bit per channel, and the same value is used for all channels.

Eximius Design, Copyright 2021

Parameter	Values	Description
TX_ENABLE_STROBE	True, False	Master Transmit Strobe Enable (and Slave Receive Strobe Enable). If False, all Strobe related functionality is removed from the Logic Link. If True, this enables the various TX_*_STROBE functionality.
RX_ENABLE_STROBE	True, False	Master Receive Strobe Enable (and Slave Transmit Strobe Enable). If False, all Strobe related functionality is removed from the Logic Link. If True, this enables the various RX_*_STROBE functionality.
TX_PERSISTENT_STROBE	True, False	Master Transmit Persistent Strobe (and Slave Receive Persistent Strobe). If True, then the Strobe will permanently occupy a space in the AIB data stream and the AXI data will be routed around it. If False, the Strobe is recoverable and can be used as AXI data after synchronization.
RX_PERSISTENT_STROBE	True, False	Master Receive Persistent Strobe (and Slave Transmit Persistent Strobe). If True, then the Strobe will permanently occupy a space in the AIB data stream and the AXI data will be routed around it. If False, the Strobe is recoverable and can be used as AXI data after synchronization.
TX_USER_STROBE	True, False	Master Transmit USER Strobe Enable. If True, then the USER can drive the Strobe via a port on the Logic Link. If False, the Strobe is driven to a 1 by the Logic Link, and it is expected that downstream logic (e.g. the CA block) will replace this with the real strobe.
RX_USER_STROBE	True, False	Slave Transmit USER Strobe Enable. If True, then the USER can drive the Strobe via a port on the Logic Link. If False, the Strobe is driven to a 1 by the Logic Link, and it is expected that downstream logic (e.g. the CA block) will replace this with the real strobe.
TX_STROBE_GEN2_LOC	0-319	Master Transmit Gen2 Strobe Location (and Slave Receive Gen2 Strobe Location). Only valid for CHAN_TYPE = Gen2Only or Gen2 configuration. Indicates the location where the Strobe bit will be in the DBI data stream. This value is used for all channels. Note if SUPPORT_ASYMMETRIC is true, the value is constrained to bits 0-79.
RX_STROBE_GEN2_LOC	0-319	Master Receive Gen2 Strobe Location (and Slave Transmit Gen2 Strobe Location). Only valid for CHAN_TYPE = Gen2Only or Gen2 configuration. Indicates the location where the Strobe bit will be in the DBI data stream. This value is used for all channels. Note if SUPPORT_ASYMMETRIC is true, the value is constrained to bits 0-79.
TX_STROBE_GEN1_LOC	0-79	Master Transmit Gen1 Strobe Location (and Slave Receive Gen1 Strobe Location). Only valid for CHAN_TYPE =

Eximius Design, Copyright 2021

		Gen1Only or Gen2 configuration. Indicates the location where the Strobe bit will be in the DBI data stream. This value is used for all channels. Note if SUPPORT_ASYMMETRIC is true, the value is constrained to bits 0-39.
RX_STROBE_GEN1_LOC	0-79	Master Receive Gen1 Strobe Location (and Slave Transmit Gen1 Strobe Location). Only valid for CHAN_TYPE = Gen1Only or Gen2 configuration. Indicates the location where the Strobe bit will be in the DBI data stream. This value is used for all channels. Note if SUPPORT_ASYMMETRIC is true, the value is constrained to bits 0-39.

Table 5 Strobe Configuration

4.3 Marker

These configure the Marker functionality as controlled from the Logic Link. Note that these configures what the Logic Link places on the Marker location(s), but it is possible downstream blocks like the AIB PHY Adaptor may override the values coming out of the Logic Link.

Note there is one Marker bit for every Full Rate chunk of data. So a Full Rate would have at most 1 marker bit[1], Half Rate would have 2 marker bits and Quarter Rate would have 4 marker bits. The USER Marker field reflects this width.

For Each LL	Values	Description
TX_ENABLE_MARKER	True, False	Master Transmit Marker Enable (and Slave Receive Marker Enable). If False, all Marker related functionality is removed from the Logic Link. If True, this enables the various TX_*_MARKER functionality.
RX_ENABLE_MARKER	True, False	Master Receive Marker Enable (and Slave Transmit Marker Enable). If False, all Marker related functionality is removed from the Logic Link. If True, this enables the various RX_*_MARKER functionality.
TX_PERSISTENT_MARKER	True, False	Master Transmit Persistent Marker (and Slave Receive Persistent Marker). If True, then the Marker will permanently occupy a space in the AIB data stream and the AXI data will be routed around it. If False, the Marker is recoverable and can be used as AXI data after synchronization.
RX_PERSISTENT_MARKER	True, False	Master Receive Persistent Marker (and Slave Transmit Persistent MARKER). If True, then the Marker will permanently occupy a space in the AIB data stream and the AXI data will be routed around it. If False, the Marker

[1] Symmetric interfaces like Full to Full (F2F) don't strictly require a marker bit, but for Asymmetric modes a Full transmitter needs the marker bit.

Eximius Design, Copyright 2021

		is recoverable and can be used as AXI data after synchronization.
TX_USER_MARKER	True, False	Master Transmit USER Marker Enable. If True, then the USER can drive the Marker(s) via a port on the Logic Link. If False, the Marker bits are driven to a 0. Note, downstream logic like CA may override the Marker if configured to do so.
RX_USER_MARKER	True, False	Slave Transmit USER Marker Enable. If True, then the USER can drive the Marker(s) via a port on the Logic Link. If False, the Marker bits are driven to a 0. Note, downstream logic like CA may override the Marker if configured to do so.
TX_MARKER_GEN2_LOC	0-79	Master Transmit Gen2 Marker Location (and Slave Receive Gen2 Marker Location). Only valid for CHAN_TYPE = Gen2Only or Gen2 configuration. Indicates the location where the Marker bit will be in the DBI data stream. This value is used for all channels.
RX_MARKER_GEN2_LOC	0-79	Master Receive Gen2 Marker Location (and Slave Transmit Gen2 Marker Location). Only valid for CHAN_TYPE = Gen2Only or Gen2 configuration. Indicates the location where the Marker bit will be in the DBI data stream. This value is used for all channels.
TX_MARKER_GEN1_LOC	0-39	Master Transmit Gen1 Marker Location (and Slave Receive Gen1 Marker Location). Only valid for CHAN_TYPE = Gen1Only or Gen2 configuration. Indicates the location where the Marker bit will be in the DBI data stream. This value is used for all channels.
RX_MARKER_GEN1_LOC	0-39	Master Receive Gen1 Marker Location (and Slave Transmit Gen1 Marker Location). Only valid for CHAN_TYPE = Gen1Only or Gen2 configuration. Indicates the location where the Marker bit will be in the DBI data stream. This value is used for all channels.

Table 6 Marker Configuration

4.4 Logic Link

The following is a simple example of a Logic Link data structure. Each Logic Link has a similar structure and starts with the keyword “llink” followed the name of the logic link and on following lines the open brace and close brace.

```
llink ST
{
  TX_FIFO_DEPTH        1
  RX_FIFO_DEPTH        32

  output user_tkeep  8
  output user_tdata  64
```


Eximius Design, Copyright 2021

```
  output user_tvalid valid
  input  user_tready ready
}
```

The Logic Link name (“ST” in the above example) will be used internally for grouping key signals.

Each Logic Link can be configured with different Transmit and Receive FIFO Depths as shown above. See Section 2 for more info on RX FIFO Depths.

The signals associated with the Logic Link are indicated here too. Each signal line has the format:

```
  [output|input] signalname  optional_width optional_lsb
```

Signals that are listed as `output` are transmitted from the master to the slave, while `input` signals are transmitted from the slave to the master. The signal names shown are exactly what the names will be on the master and slave USER sides.

If there is a value specified for `optional_width`, then the signal will be defined as a bus of the width specified. If no width is specified, then the signal is a 1 bit scaler. The `optional_lsb` field provides the ability to specify the least significant bit offset for the bus, but the overall width is still specified by `optional_width`. So this example will result in a bus called **user_signal1[9:2]**:

```
  input user_signal1 8 2
```

Note that in place of the `optional_width` parameter, a keyword of “valid” or “ready” can be used to indicate the Valid / Ready of the AXI Channel. These are single bit scalers and can have special meaning in the Logic Link.

4.5 Verilog Instantiation

The actual ports of the Logic Link will be defined by the configuration. For example, the AXI signal names are drawn from the .cfg file, as are the number of transmit/receive AIB channels and their configurations.

This is a list of the common IO on all Logic Links and are not affected by the configuration.

Port Name	Direction	Description
clk_wr	input	Primary clock for the system. Same as used on the AIB PHY interface.
rst_wr_n	input	This is an active low asynchronous reset. The reset should be synchronously deasserted with respect to clk_wr.
tx_online	input	Transmit Online. This feature is used to hold off USER transmit until the AIB link is up/running. Refer to AutoSynchonization section for more details.
rx_online	input	Transmit Online. This feature is used to ignore incoming AIB data until the AIB link is up/running. Refer to AutoSynchonization section for more details.
m_gen2_mode	input	Only used in Dynamic Gen2/Gen1 settings. This is the same configuration signal used in AIB PHY. If it is high, we are in Gen2 mode. If we are low we are in Gen1 mode. If unused, tie to 1.

Eximius Design, Copyright 2021

delay_x_value[7:0]	input	Delay used in AutoSynchronization. Tie to 0 if unused.
delay_y_value[7:0]	input	Delay used in AutoSynchronization. Tie to 0 if unused.
delay_z_value[7:0]	input	Delay used in AutoSynchronization. Tie to 0 if unused.

Table 7 Common Logic Link Ports

The below are configuration specific ports. The name of the Logic Link is referred to as <LLINK> below.

Port Name	Direction	Description
tx_<LLINK>_debug_status	output	Debug signal only present if there is a Transmit Logic Link. [31:24] – Current TX Credits [23:20] – rsvd [19] – Delayed RX_ONLNE (delayed by delay_x_value) [18] – Delayed TX_ONLNE (delayed by delay_y_value + delay_y_value + USER_STROBE/MARKER) [17] – Sticky indication of an underflow in the TX FIFO. Requires reset to clear. [16] – Sticky indication of an overflow in the TX FIFO. Requires reset to clear. [15:8] – Configured TX FIFO Depth [7:0] – Current TX FIFO Entries
rx_<LLINK>_debug_status	output	Debug signal only present if there is a Receive Logic Link. [31:20] – rsvd [19] – Delayed RX_ONLNE (delayed by delay_x_value) [18] – Delayed TX_ONLNE (delayed by delay_y_value + delay_z_value + USER_STROBE/MARKER) [17] – Sticky indication of an underflow in the RX FIFO. Requires reset to clear. [16] – Sticky indication of an overflow in the RX FIFO. Requires reset to clear. [15:8] – Configured RX FIFO Depth [7:0] – Current RX FIFO Entries
user_signals eg: user_tkeep[7:0] user_tdata[63:0] user_tvalid	input/output	Configuration specific.

Eximius Design, Copyright 2021

user_tready		
tx_phyN[W-1:0]	output	Transmit AIB channel data. N refers to the Channel number. The size of this bus is W which is affected by Gen and Rate. This channel can be connected to the AIB PHY data_in (or data_in_f) or CA tx_din as the system requires.
rx_phyN[W-1:0]	input	Receive AIB channel data. N refers to the Channel number. The size of this bus is W which is affected by Gen and Rate. This channel can be connected to the AIB PHY data_out (or data_out_f) or CA rx_dout as the system requires.
init_LLINK_credit[7:0]	input	Initial Transmit Credits. Only present if there is a Transmit Logic Link. If tied to 0, the TX Credits will be initialized according to the size of the configuration RX_FIFO Depth which is the recommended flow. Any other value will override the initial TX Credits, which could lead to incorrect behavior.
tx_stb_userbit	input	USER inserted Strobe bit. Only present on Master if TX_USER_STROBE is set to true. Only present on Slave if RX_USER_STROBE is set to true. The current value of the USER Strobe will be inserted into the Logic Link Strobe, but this can be overridden by downstream blocks like the CA.
tx_mrk_userbit[N-1:0]	input	USER inserted Marker bit. N bits wide where N is 1, 2 or 4 for Full, Half or Quarter Rate. Only present on Master if TX_USER_MARKER is set to true. Only present on Slave if RX_USER_MARKER is set to true. The current value of the USER Marker will be inserted into the Logic Link Marker locations, but this can be overridden by downstream blocks like the AIB PHY.

Table 8 Configuration Specific Logic Link Ports

4.6 Full Example

Below is the full example of the above “common” configuration. This is used in the ${PROJ_DIR}/axi4-st/cfg/axi_st_d64.cfg

```
MODULE axi_st_d64

// PHY and AIB Configuration
NUM_CHAN                1
CHAN_TYPE               Gen2Only //Gen1Only, Gen2Only, Gen2, AIBO
TX_RATE                 Full    // Full, Half, Quarter
RX_RATE                 Full    // Full, Half, Quarter
TX_DBI_PRESENT          True
```

```
RX_DBI_PRESENT          True

// Channel Alignment Strobe Configuration
TX_ENABLE_STROBE        True    // If False, all strobe functionality is removed.
RX_ENABLE_STROBE        True    // If False, all strobe functionality is removed.
TX_PERSISTENT_STROBE    False   // If True strobes are persistent (always there).
RX_PERSISTENT_STROBE    False   // If True strobes are persistent (always there).
TX_USER_STROBE          True    // If True, then we input user generated signal
RX_USER_STROBE          True    // If True, we output recovered signal
TX_STROBE_GEN2_LOC      76      // Location of Strobe when in Gen2 Mode
RX_STROBE_GEN2_LOC      76      // Location of Strobe when in Gen2 Mode

// Word Marker Configuration
TX_ENABLE_MARKER        True    // If False, all Marker functionality is removed.
RX_ENABLE_MARKER        True    // If False, all Marker functionality is removed.
TX_PERSISTENT_MARKER    False   // If True Markers are persistent (always there).
RX_PERSISTENT_MARKER    False   // If True Markers are persistent (always there).
TX_USER_MARKER          True    // If True, then we input user generated signal
RX_USER_MARKER          True    // If True, we output recovered signal
TX_MARKER_GEN2_LOC      4       // Location of Marker when in Gen2 Mode
RX_MARKER_GEN2_LOC      4       // Location of Marker when in Gen2 Mode

llink ST
{
  TX_FIFO_DEPTH        1
  RX_FIFO_DEPTH        40

  output user_tkeep  8
  output user_tdata  64
  output user_tvalid valid
  input  user_tready ready
}
```

Below is the full example of the above axi_st_d64 master and slave instantiations.

```
  axi_st_d64_master_top axi_st_master_top_i
     (// Outputs
      .tx_phy0                      (tx_phy_master_0[79:0]),
      .user_tready                  (user1_tready),
      .tx_st_debug_status           (tx_st_debug_status[31:0]),
```

```
   // Inputs
   .clk_wr                          (clk_wr),
   .rst_wr_n                        (rst_wr_n),
   .tx_online                       (master_tx_online),
   .rx_online                       (master_rx_online),
   .init_st_credit                  (8'd0),
   .rx_phy0                         (rx_phy_master_0[79:0]),
   .user_tkeep                      (user1_tkeep[7:0]),
   .user_tdata                      (user1_tdata[63:0]),
   .user_tlast                      (user1_tlast),
   .user_tvalid                     (user1_tvalid),
   .m_gen2_mode                     (1'b1),
   .tx_mrk_userbit                  (1'b1),
   .tx_stb_userbit                  (1'b1),
   .delay_x_value                   (8'h5),
   .delay_xz_value                  (8'h20),
   .delay_yz_value                  (8'h40));


axi_st_d64_slave_top axi_st_slave_top_i
  (// Outputs
   .tx_phy0                         (tx_phy_slave_0[79:0]),
   .user_tkeep                      (user2_tkeep[7:0]),
   .user_tdata                      (user2_tdata[63:0]),
   .user_tlast                      (user2_tlast),
   .user_tvalid                     (user2_tvalid),
   .rx_st_debug_status              (rx_st_debug_status[31:0]),
   // Inputs
   .clk_wr                          (clk_wr),
   .rst_wr_n                               (rst_wr_n),
   .tx_online                       (slave_tx_online),
   .rx_online                       (slave_rx_online),
   .rx_phy0                         (rx_phy_slave_0[79:0]),
   .user_tready                     (user2_tready),
   .m_gen2_mode                     (1'b1),
   .tx_mrk_userbit                  (1'b1),
   .tx_stb_userbit                  (1'b1),
   .delay_x_value                   (8'h5),
   .delay_xz_value                  (8'h20),
   .delay_yz_value                  (8'h40));
```


Eximius Design, Copyright 2021

5. AXI-ST

5.1 Simple, No Frills

This is the no frills, simple AXI-ST over AIB interface.

5.1.1 Theory

This is a Logic Link unimaginatively called “ST” that implements a simple AXI-ST interface with 64 bits of TDATA and 8 bis of the TKEEP.

Note that TVALID is marked as a valid bit and TREADY is marked as a ready bit. This information is used to generate the correct flow control in the RTL.

```
llink ST
{
  TX_FIFO_DEPTH        1
  RX_FIFO_DEPTH        40

  output user_tkeep  8
  output user_tdata  64
  output user_tvalid valid
  input  user_tready ready
}
```

The TX_FIFO_DEPTH is the depth of the TX FIFO.

The RX_FIFO_DEPTH is the depth of the RX FIFO. As described in the Logic Link Overview above, the depth of the RX_FIFO is also the initial Transmit Credit. For maximum throughput, the RX_FIFO_DEPTH should be greater than the round trip delay between the two chiplets. See Section 2 for more information.

5.1.2 Example

The configuration for this file be seen in ${PROJ_DIR}/axi4-st/cfg/axi_st_d64.cfg

This implements a simple, no frills AXI-ST over a single Full Rate Gen2 channel. This has a simple, Verilog, Logic Link only testbench here: ${PROJ_DIR}/axi4-st /dv/axi_st_d64

The test is setup for Xcellium execution, but the run script can be modified for any simulator.

The test goes through 8 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 - Simple, non overlapping minimum sized transfers

Phase 2 - Overlapping minimum sized transfers

Phase 3 - Simple, non overlapping medium sized transfers

Phase 4 - Overlapping medium transfers

Phase 5 - Simple, non overlapping large sized transfers

Phase 6 - Overlapping large transfers

Phase 7 - Random size, packets

Eximius Design, Copyright 2021

Phase 8 - Overlapping large transfers with no flowcontrol from downstream

The master side is designated by user1_* signals and the slave side is designated by user2_* signals.

5.2 Optional Ready / Valid

5.2.1 Theory

In theory, the AXI-ST TREADY signal is not required in the interface. Without it, there is no flow control but there is also no signaling going from the slave side back to the master side, which may be an implementation boon.

When the TREADY is removed from the LLINK design, the flow control FIFOs are also removed which effectively means a combinatorial path from the USER side to the AIB PHY side. This may result in new timing paths, which can be mitigated by setting the TX_REG_PHY or RX_REG_PHY to True.

When the TREADY is removed, TVALID is treated like any other data signal. There is no PUSHBIT (and for that matter, no CREDIT signal), rather the TVALID is simply another bit on AXI-ST bus.

Note that the LLINK also provides a mechanism to support no TREADY and no TVALID which will allow for the maximum data available on the AIB lines. Note that TVALID is required to be legal AXI-ST[2] so the receive side logic may need to drive a local TVALID high to appease downstream IP.

5.2.2 Example

The configuration for this file be seen in ${PROJ_DIR}/axi4-st/cfg/axi_st_d64_nordy.cfg.

This is purposely intended to be very similar to axi_st_d64.cfg for comparison.

This has a simple, Verilog, Logic Link only testbench here: ${PROJ_DIR}/axi4-st /dv/axi_st_d64_nordy_tb

The test goes through 8 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 - Simple, non overlapping minimum sized transfers

Phase 2 - Overlapping minimum sized transfers

Phase 3 - Simple, non overlapping medium sized transfers

Phase 4 - Overlapping medium transfers

Phase 5 - Simple, non overlapping large sized transfers

Phase 6 - Overlapping large transfers

Phase 7 - Random size, packets

Phase 8 - Overlapping large transfers with no flowcontrol from downstream

The master side is designated by user1_* signals and the slave side is designated by user2_* signals. Note in this sim, we have a couple of wires called user1_tready and user2_tready that we have tied high in the TB. This was required for the local TB transactor, but the TREADY signal is not actually used in LLINK and is not transmitted across the AIB.

[2] Amusingly TDATA and TREADY are optional in the AXI specification, but TVALID is not considered optional. See AMBA 4 AXI-4 Stream Protocol Specifications section 3.

Eximius Design, Copyright 2021

5.3 Dynamic Gen2/Gen1 Switching

5.3.1 **Theory**

The Logic Link supports running an AXI-ST in both a Gen2 and Gen1 mode, allowing the same chiplet to operate in both Gen2 and Gen1 modes, albeit in a reduced data width. This requires the system be configured in one of two ways:

Option	Gen2 Rate	Gen1 Rate	Data Width
A	Half Rate	Full Rate	Gen1 data width is generally ¼ that of Gen2
B	Quarter Rate	Half Rate	Gen1 data width is generally ¼ that of Gen2

Note that the supported Gen2/Gen1 Rates imply for the same clock speed in Gen1 or Gen2. This allows the Logic Link on both chiplets to operate using the same clock.

The Gen2 vs Gen1 rate is controlled by the logic link port m_gen2_mode, which has the same functionality and polarity as the AIB PHY signal of the same name, specifically driving the signal high will enable Gen2 behavior and low will enable Gen1 behavior.

The Logic Link also allows for different locations for the Strobe and Marker in Gen2 and Gen1. However, the other Strobe / Marker configurations, such as USER inserted signals and Persistent signal, will be the same configuration in both Gen2 and Ge1. Note too that Gen2 may have DBI enabled, but since Gen1 does not support DBI, this option will be ignored for the Gen1 portion.

The AIB PHY has a requirement that m_gen2_mode should remain constant while the AIB is out of reset, and the Logic Link behavior is the same, that is the m_gen2_mode should remain constant while the Logic Link is out of reset.

The Dynamic Gen2 / Gen1 mode requires that the Gen1 signals necessarily be a complete subset of the Gen2 signals, that is no new signals can be defined for Gen1 that were not already present in Gen2 mode.

5.3.2 **Example**

The Logic link for a Gen2/Gen1 encoding looks like the following:

```
LLINK ST
{
  TX_FIFO_DEPTH        1
  RX_FIFO_DEPTH        64

  output user_tkeep  32
  output user_tdata  256
  output user_tlast
  output user_tvalid valid
  input  user_tready ready

  GEN2_AS_GEN1

  output user_tkeep  8
  output user_tdata  64
```


Eximius Design, Copyright 2021

```
  output user_tvalid valid
  input  user_tready ready
}
```

The first half describes the Gen2 operation, and the second half (after the GEN2_AS_GEN1) describes the Gen1 operation. Note that the total data width available on the Gen2 AIB channels are approximately 1/4th that of the Gen1 AIB channel. However, the specific details could allow different options.

Note that in this example, the Gen2 AXI-ST has TLAST, while Gen1 does not, which is an allowed configuration if the USER desires. The reverse (Gen1 having TLAST but Gen2 not having it) is not allowed because the Gen1 should be a subset of the Gen2 signals.

The full configuration file can be found here:

${PROJ_DIR}/axi4-st/cfg/axi_st_d256_gen1_gen2.cfg

Note that the Strobe is in bit position 76 for Gen2 and 35 for Gen1, while the Marker is in bit position 4 for Gen2 and 39 for Gen1.

This has a simple, Verilog, Logic Link only testbench here:

${PROJ_DIR}/axi4-st/dv/axi_st_d256_gen1_gen2

The sim can be invoked by executing the run command in that directory:

./run_st_d256_gen1_gen2

Waves will be generated. The test is setup for Xcellium execution, but the run script can be modified for any simulator.

The test goes through 11 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 – Gen2 Simple, non overlapping minimum sized transfers

Phase 2 - Gen2 Overlapping minimum sized transfers

Phase 3 - Gen2 Simple, non overlapping medium sized transfers

Phase 4 - Gen2 Overlapping medium transfers

Phase 5 - Gen2 Simple, non overlapping large sized transfers

Phase 6 - Gen2 Overlapping large transfers

Phase 7 - Gen2 Random size, packets

Phase 8 - Gen2 Overlapping large transfers with no flowcontrol from downstream

Phase 9 - Gen1 Simple, non overlapping minimum sized transfers

Phase 10 - Gen1 Random size, packets

Phase 11 - Gen1 Overlapping large transfers with no flowcontrol from downstream

5.4 Asymmetric Gearboxing

5.4.1 **Theory**

Asymmetric Gearboxing allows the AXI-ST Logic Links to be launched from an AIB running at one rate and received by an AIB running at a different rate. This allows a common chiplet to serve data to a variety of different chiplets running at different clock speeds, but maintain the overall bandwidth.

For example, a high-end A2D chiplet might be generating a 128 bit data AXI-ST stream running at Full Rate. This chiplet could be talking to an FPGA requiring Quarter Rate or to a mid-range DSP chiplet running at Half Rate.

5.4.1.1 AIB Refresher

The AIB uses the gearboxing to convert from Full, Hall or Quarter rate on the Transmitter to Full, Half or Quarter rate on the Receiver.

Below is Full Rate to Quarter Rate (abbreviated F2Q). Note the USER inserted Marker on the transmit side. This effectively determines how the receiver will package up the receive data.

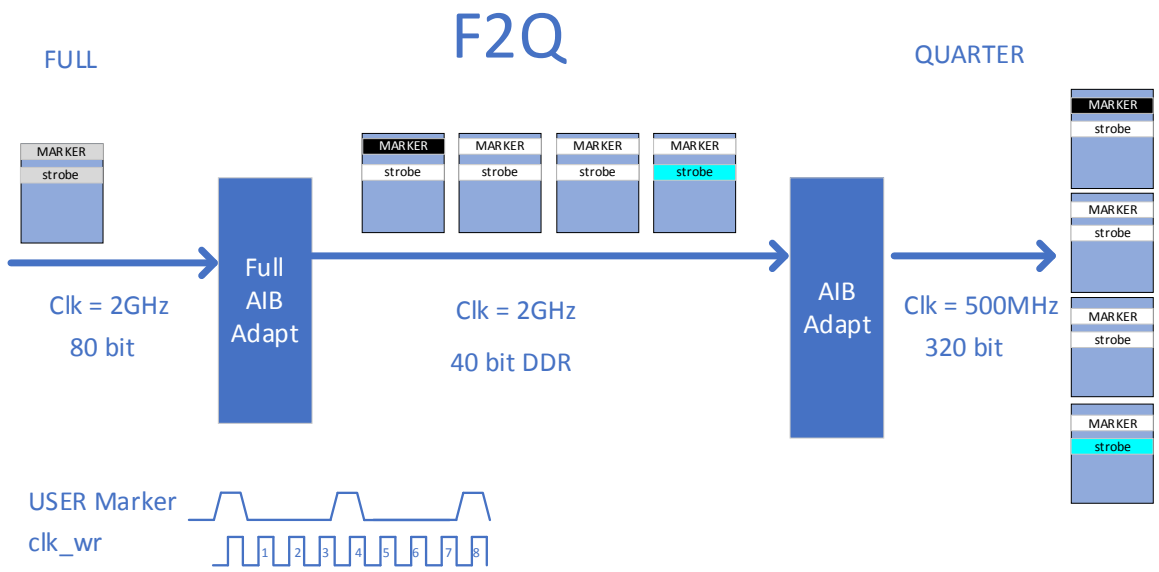


Figure 2 Asymmetric F2Q

Below Full Rate to Half Rate. Note the USER Marker pattern changed from F2Q.

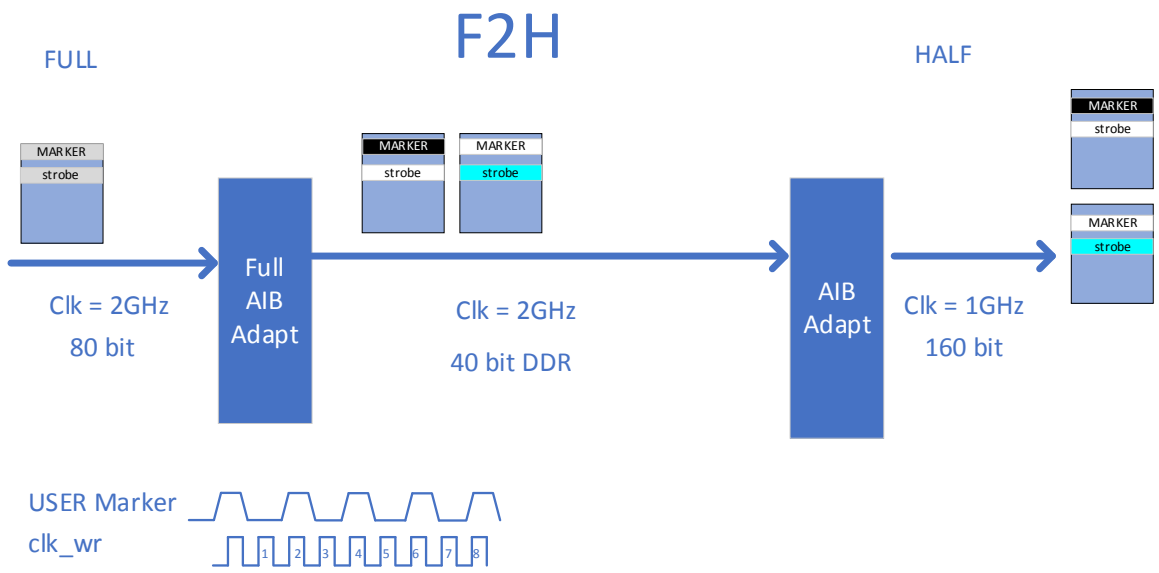


Figure 3 Asymmetric F2H

Eximius Design, Copyright 2021

Below is Full Rate to Full Rate. Note this is a little different than Symmetric LLINK simply because the flexibility to interoperate in other modes produces a different design than a fixed rate block. So even though it is the same rate on both sides, we call this Asymmetric.

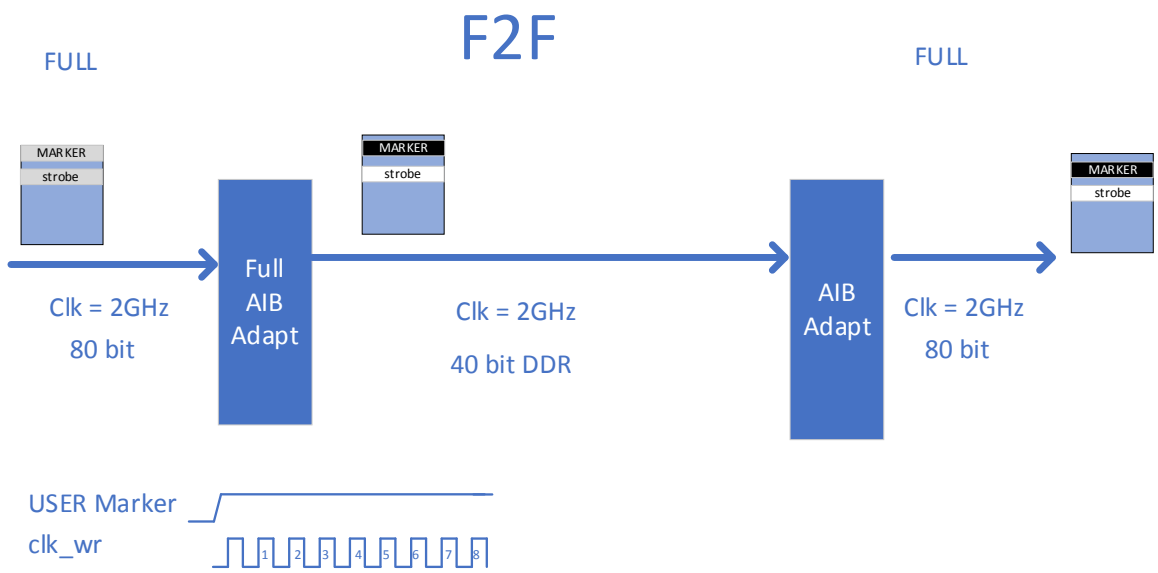
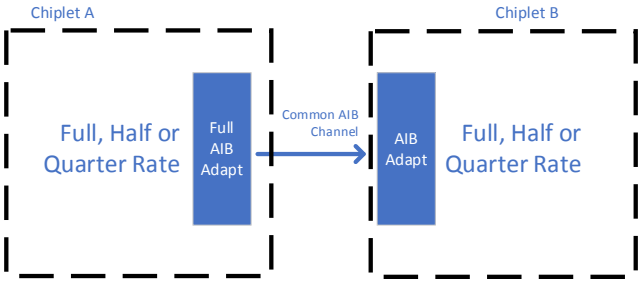


Figure 4 Asymmetric F2F

Basically, AIB allows for interoperability via the Asymmetric Gearboxing. That is Chiplet A can be configured in a specific Rate, and it can talk to Chiplet B which could be configured in a different Rate.

To realize this, the Logic Link uses a Replicated Struct, which can be defined as the minimum quanta that can be used in Full, Half or Quarter Rate.

As a rule, a Transmitter will send 1x, 2x or 4x Replicated Structs in Full, Half or Quarter mode. The Receiver will independently receive 1x, 2x or 4x Replicated Structs depending on its configured Rate.



Eximius Design, Copyright 2021

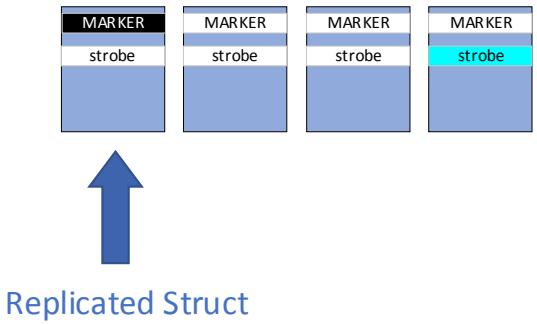


Figure 5 Asymmetric Replicated Struct

5.4.1.2 AXI-ST Refresher

AXI-ST is a stream of bytes.

Unlike AXI-MM, the byte positions are not fixed and can be shifted in different byte positions, but the order of the bytes must be maintained. For example, the two streams to the right are identical.

Null	Null	D-07	D-0A	Null	D-0F
D-01	Null	D-06	D-09	Null	D-0E
Null	D-03	D-05	D-08	D-0C	Null
D-00	D-02	D-04	Null	D-0B	D-0D

D-02	D-06	Null	D-0B	Null	Null
Null	D-05	Null	D-0A	D-0E	D-0F
D-01	D-04	Null	D-09	D-0D	Null
D-00	D-03	D-07	D-08	D-0C	Null

Figure 6 AXI-ST stream

This is from the AXI-ST spec (page 15, Figure 1-1). Not shown, but effectively implied is the NULL bytes are indicated by individual bits in TKEEP[3:0] being zero.

These are the TKEEP for the above examples starting with the first beat of data on the left:

TKEEP [3:0] Left = 0x5, 0x3, 0xf, 0xe, 0x3, 0xd

TKEEP [3:0] Right = 0xb, 0xf, 0x1, 0xf, 0x7, 0x4

Changing AXI-ST TDATA width from a smaller bus to a wider bus (e.g. 16 byte to 32 byte) is called Upsizing. Gong from a wider bus to a smaller one is call Downsizing.

AXI-ST has certain rules for Upsizing/Downsizing. Some are obvious, such as TDATA/TKEEP/TUSER bits should maintain their relative position after the up/downsizing (i.e. the TDATA that what was marked as valid by TKEEP should continue to be marked as valid after the upsizing/downsizing).

Eximius Design, Copyright 2021

One less obvious rule is that if TLAST is set, it effectively prevents upsizing, or at least prevents taking two beats of TDATA of size X both with TLAST set. For this reason, TLAST is not supported as the Logic Link will not generate legal AXI-ST[3].

Note that the structure shown here is very similar to the Replicated Struct going from Half to Quarter.

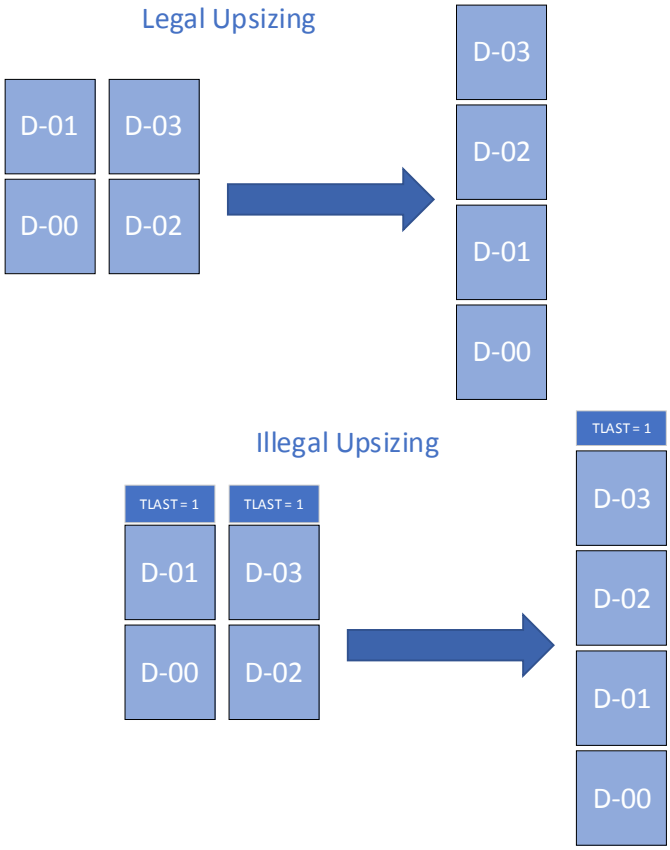


Figure 7 AXI-ST Upsizing Example

5.4.1.3 Replicated Struct Example

Lets define a replicated struct for the Full case. For example:

```
llink ST
{
        TKEEP     8
        TDATA    64
        TVALID
        TREADY
}
```

[3] Technically, the LLINK allows the multiple TLASTs on an upsizing, but it is up to the receiver to handle and decode this oddity. For strictly legal AXI-ST, do not use TLAST on upsized interfaces.

Eximius Design, Copyright 2021

The generated Master RTL will have user interfaces that look like the modules below.

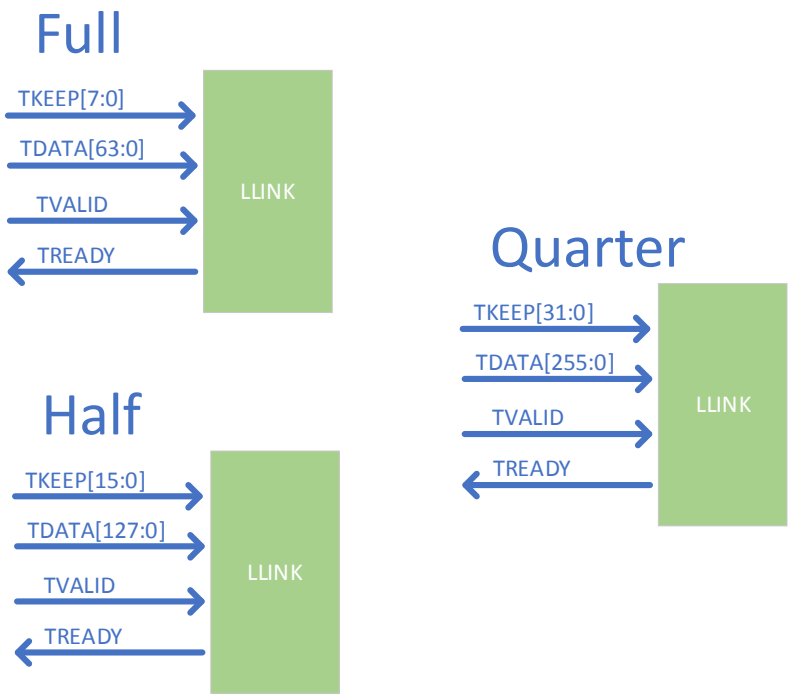


Figure 8 Asymmetric Master Examples

The generated Slave Logic Links generally have a mirror version of the Master but note that the Slaves have an extra signal called ENABLE. This is an artifact of the asymmetric interface, and it acts as a kind of VALID for the entire replicated struct.

We'll discuss the ENABLE signal next.

Eximius Design, Copyright 2021

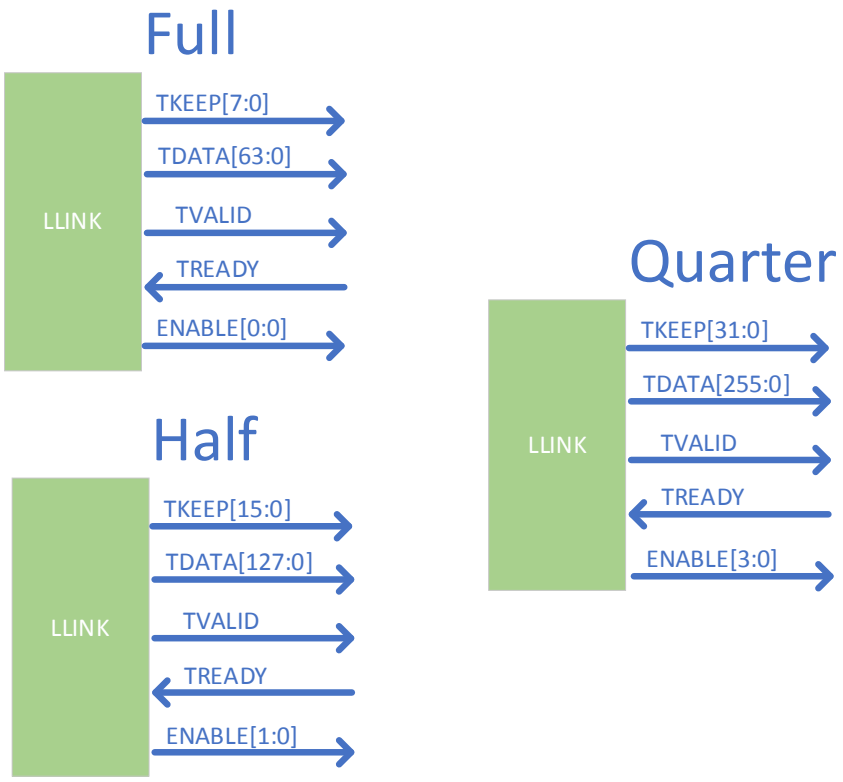


Figure 9 Asymmetric Slave Examples

Lets take a simple example, a F2H. Recall the Master Full Rate has 64 bits of TDATA and the Slave Half Rate has 128 bits of TDATA.

The Full side will receive one beat of AXI-ST data which is 64 bits of TDATA and 8 bits of TKEEP. It sends this to the Half side.

On the Receive Half side, the 64 bits of data will show up in either the upper or lower 64 bits of the 128 bit TDATA output.

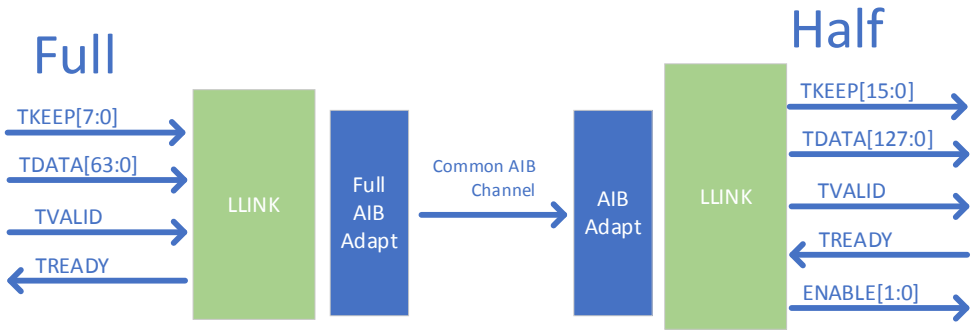


Figure 10 Asymmetric Example

Eximius Design, Copyright 2021

In an overly simplistic view, the location of the Full beat of transmit data in the Half beat of receive data is determined by the Marker bit. If the Marker bit was low, this indicates the data will be on the lower part of the Half data.

However, generally we cannot rely on this mechanism for AXI-ST using flow control. The AIB channel does not “stall” when the AXI-ST asserts flow control. So there could be many cycles after the AXI-ST data is fed into the Transmit Full LLINK before it can egress on the AIB, which will result in a pseudorandom location in the Half rate receive data.

Note that AXI-ST LLINK without flow control (i.e. no TREADY) should have a predictable location of the data based off of the USER inserted Marker bit.

AXI-ST LLINK with flow control that has not had to throttle due to lack of credits should also be predictive, but due to implementation, there is a one cycle delay from the USER inserted Marker bit due to the TX FIFO.

Examples are shown to the below showing Full Rate Transmit with contiguous bursts and no stalling due to flow control and how they could be received by the Half Rate Receive in one of two alignments, including the ENABLE values.

Note the initial beat location is unknown, but the pattern is more predictive after that (assuming no flow control on AXI-ST).

The Grey boxes represent garbage data. The first few beats of data will generally have zeros in these locations, but later beats will have stale garbage data, necessarily duplicates of previous Full data. The ENABLE effective differentiates the valid data from the garbage data.

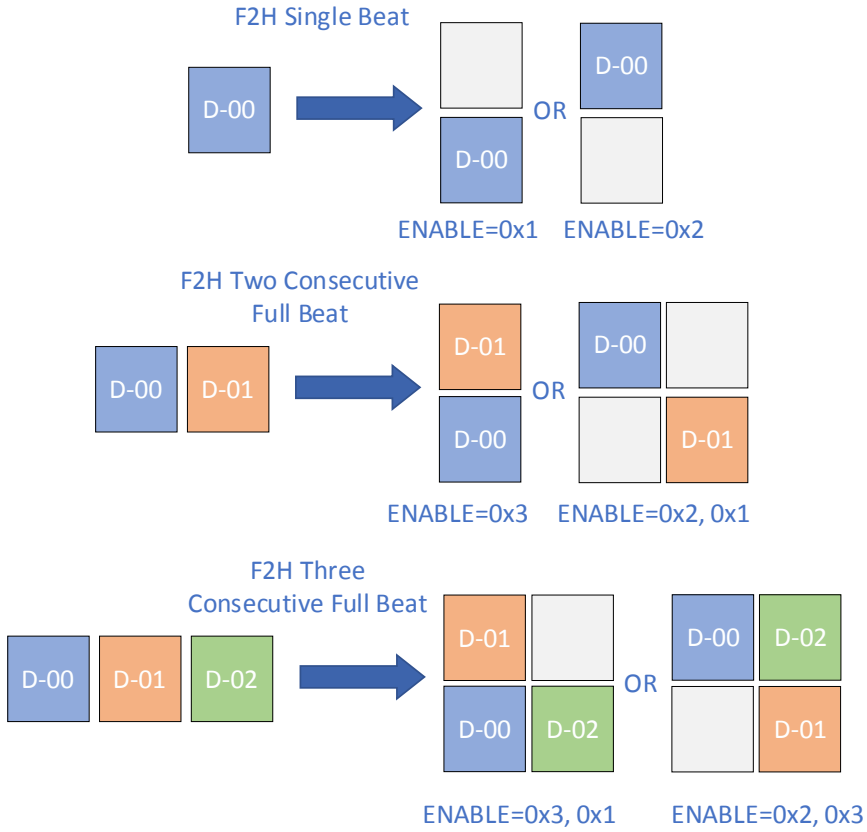


Figure 11 Asymmetric Receive Examples

Eximius Design, Copyright 2021

The actual use of the ENABLE is dependent on the application specific data being sent on the AXI-ST interface. If a TKEEP is present, then the most straightforward thing to do is to effectively AND the Replicated Struct's worth of TKEEP with the ENABLE. In our previous example, we had an 8 bit TKEEP, so we can simplistically AND each 8 bit chunk of TKEEP with the corresponding bit of ENABLE.

An example of this is shown below with a Quarter Rate Receive interface

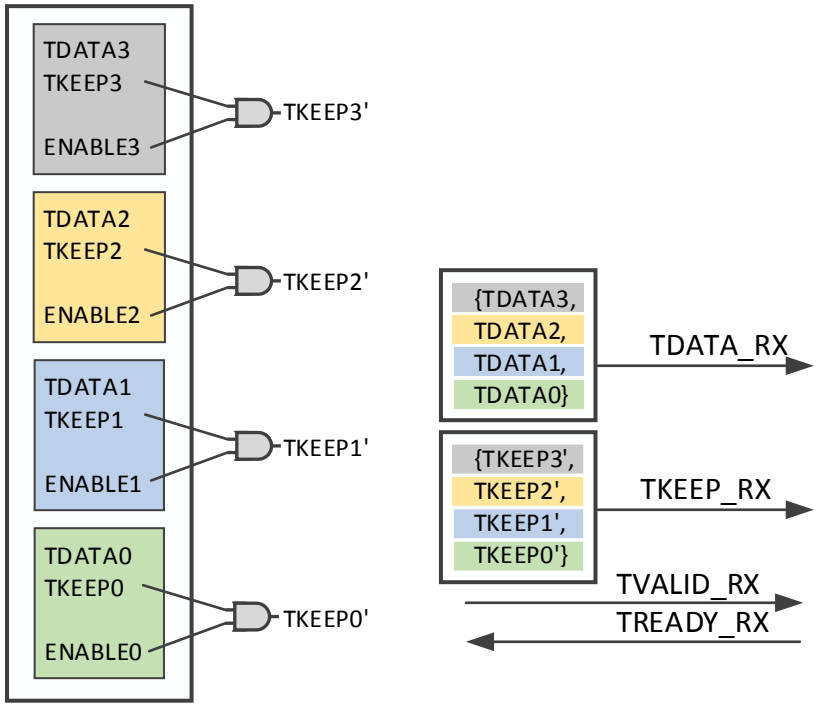


Figure 12 Asymmetric Example

However, not all AXI-ST will have TKEEP. In this situation, the USER needs to make intelligent decisions on what to do based off of the interface. Some potential examples are listed below:

1. The Receive AXI-ST has a TKEEP, but the Transmit side does not. In this scenario ENABLE is simply replicated to form a TKEEP signal on the receive size, but the TKEEP is assumed to be all 1s on the Transmit side and is not transmitted.

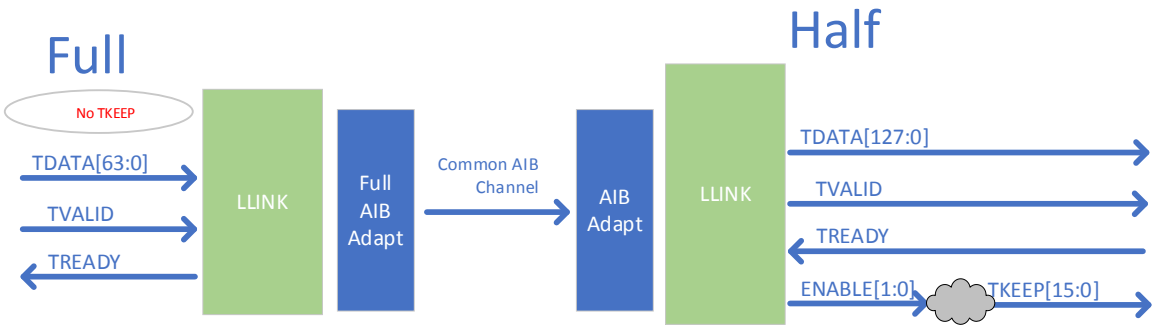


Figure 13 Asymmetric Example

Eximius Design, Copyright 2021

2. It is possible the TDATA has an invalid value (e.g. all 0s). The ENABLE can be used to zero out this data. This can be seen to the right.

Because these methods are application specific, this needs to be handled outside of the LLINK in a way appropriate to the application data.

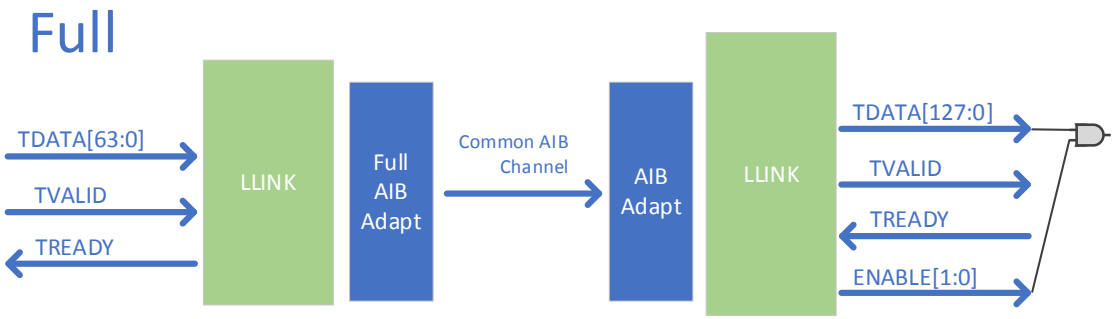


Figure 14 Asymmetric Example

5.4.1.4 Strobes and Markers

Another non-obvious side effect of the asymmetric interoperability is the Strobe and Markers. As a rule, all Strobes and Markers must be generated by the USER and fed into the LLINK. In the case of operating in a single channel environment, the system does not have to use a Strobe. However, in all Asymmetric cases, there will always be Marker insertion, even in the Full to Full configuration since the markers are how we interoperate. See Asymmetric Marker Insertion figure below for a list of expected markers.

For a multi-channel design, a strobe is needed to align the channel data. The Full Replicated Struct can (in multi-channel cases) have this strobe defined too. However, this means when we receive multiple Replicated Struct on the remote side, we'll end up with multiple Strobes on the receive side.

Similarly, even though Full Rate does not require a Marker for receive, we a require to allocate space for a Marker bit in the Replicated Struct to allow for interoperating with different Rates.

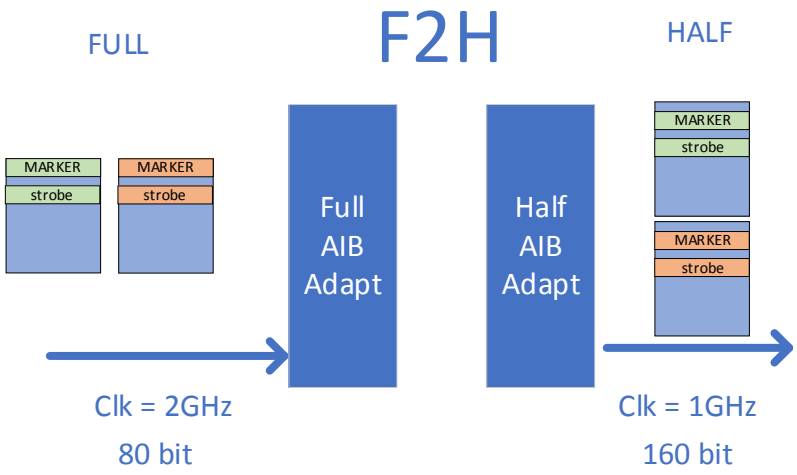


Figure 15 Asymmetric Strobe and Marker Example

Eximius Design, Copyright 2021

Since the Marker and Strobe are not affected by AXI-ST flow control, their position should be deterministic with respect to the USER Marker.

Note that the system relies on the USER Marker to determine the rate of the Receive Side.

It is possible to make this Marker Non-Persistent (or Recoverable) and this will free up space on the AIB line for more data. However, the Transmit LLINK still requires the USER Marker to be fed into the LLINK, even if it is not being actively transmitted to the remote side. The Marker is used to determine crediting schemes between the local and remote AXI-ST interface.

For Marker generation, a module has been provided to help generate correct markers. It is in $PROJ_DIR/common/dv/marker_gen.sv. It takes in a 4 bit value for the LocalRate and RemoteRate, which is 4'h1, 4'h2 or 4'h4 for Full, Half or Quarter. It will then generate a 4 bit marker, with the expectation that bit[0] is used in a Full implementation, bits[1:0] for a Half implementation and bits [3:0] for a Quarter implementation.

For Strobe generation, a module has been provided to help generate correct markers. It is in $PROJ_DIR/common/dv/strobe_gen.sv. It takes in the OR reduction of the 4 bit marker output of marker_gen and uses an interval. That interval should be set according to the following formula:

Local interval = (remote CA.rx_stb_intv * RemoteRate) / LocalRate

So a F2Q case where the remote Quarter Rate side is expecting an interval of 8 should have a 32 interval on the transmit Full Rate side. Note that because there is a possibility of a divide by 4, it is wise to chose rx_stb_intv that are multiples of 4. Generally, the rx_stb_intv is recommended to be 3x the CA FIFO Depth to ensure no aliasing.

The above marker strobe output of these modules can be connected to the LLINK USER Marker and USER Strobes for transmission to the remote side.

5.4.1.5 Flow Control

Recall that the purpose of the LLINK Transmit Credits is to ensure we do not overflow the remote Receive FIFO. If we do not have space to receive in the Remote FIFO, we should assert flow control back to the master by deasserting READY.

The LLINK implements the RX FIFO based on the Full Rate Replicated Struct RX_FIFO_DEPTH. The Full Rate Receive FIFO will have the full RX_FIFO_DEPTH. The Half Rate Receive FIFO will have double the width, but half of the depth of the RX_FIFO_DEPTH. The Quarter Rate FIFO will similarly be four times as wide, but one fourth the depth.

On the Transmit side, we start with the full RX_FIFO_DEPTH initial credits, and decrement by 1, 2 or 4 depending the combination of the Transmit and Remote side Rate. In this case the TX credit refers to a Replicated Struct in the Receive FIFO.

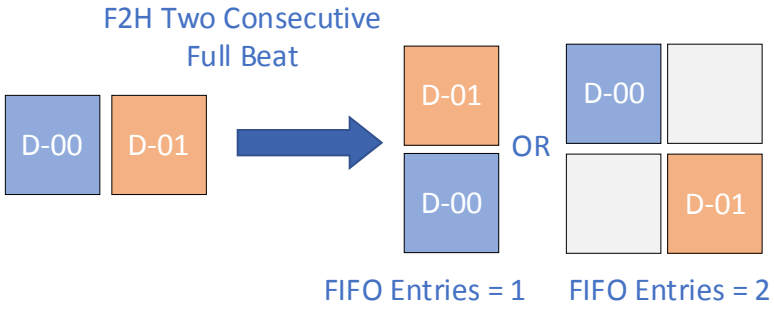


Figure 16 Asymmetric Credit Example

The above is an example of the trickiness of the Asymmetric credit flow. This example shows how 2 beats of a Full Transmit can occupy 1 or 2 entries on the RX FIFO of a Half Rate Receive.

Eximius Design, Copyright 2021

5.4.1.6 *Multi-Channel Issue*

There is an oddity that is created by the AIB Asymmetric interface across multiple channels. To see this, first observe the Half to Half case as shown below. Note the TDATA size is expressed as bytes (so 39:30 is 10 bytes or 80 bits). It seems obvious, but the Green and Orange data are the lower and upper bits of the TDATA and are generally restricted to Channel 0 and Channel 1, respectively.

This will produce good AXI-ST on the remote side as shown without any additional logic.

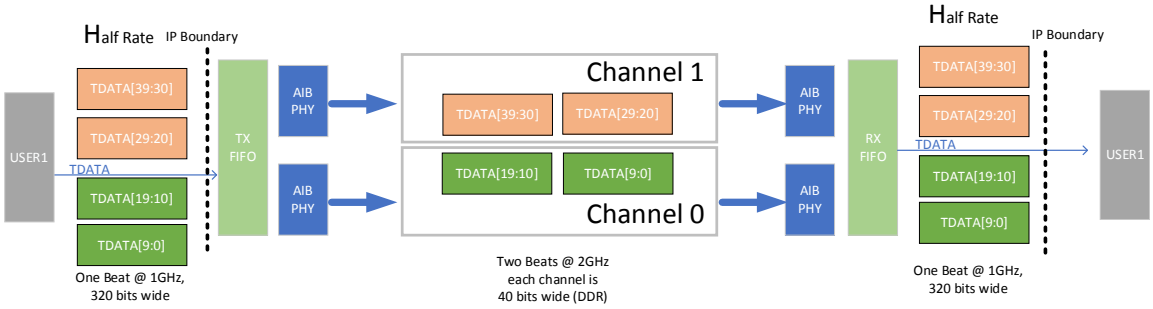


Figure 17 Asymmetric Multi Channel 1

However, in the Full to Half case below, the Green and Orange data are intermixed due to how the Full side data is presented. As a result, the data is received differently on the Half Rate Receive than in the H2H case.

As a general rule, this occurs any time an upsizing occurs on multiple channels. So F2H, F2Q or H2Q across multiple channels will require the swizzling. Note that the swizzling is left up to the USER since this requires configuration of the remote side transmit, which was counter to the requested design.

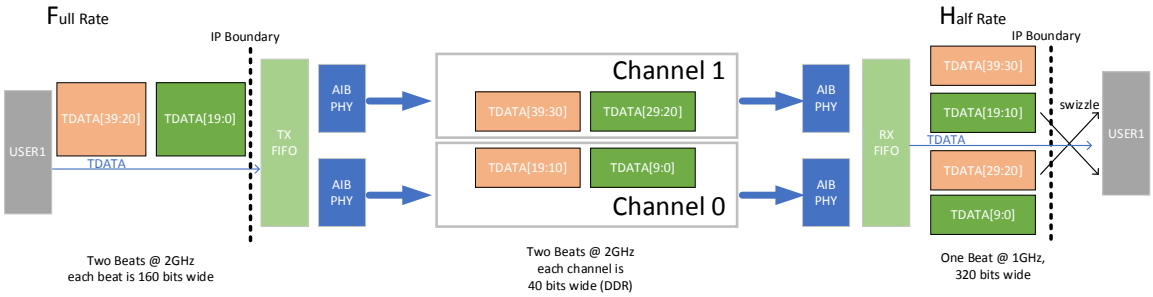


Figure 18 Asymmetric Multi Channel 2

This is an example configuration for a 128 bit data AXI-ST occupying two channels.

By convention, the module name refers to the Full Rate sizing.

The script will generate a set of Masters (Full, Half, Quarter) and a set of Slaves (Full, Half Quarter). The USER should instantiate the pieces that make sense for the implementation.

```
MODULE axi_st_d128_asym


// PHY and AIB Configuration
NUM_CHAN                2
```

```
CHAN_TYPE               Gen2Only
TX_DBI_PRESENT          True
RX_DBI_PRESENT          True

// Channel Alignment Strobe Configuration
TX_ENABLE_STROBE        True
RX_ENABLE_STROBE        True
TX_PERSISTENT_STROBE    True
RX_PERSISTENT_STROBE    True
TX_USER_STROBE          True
RX_USER_STROBE          True
TX_STROBE_GEN2_LOC      1
RX_STROBE_GEN2_LOC      1

// Word Marker Configuration
TX_ENABLE_MARKER        True
RX_ENABLE_MARKER        True
TX_PERSISTENT_MARKER    True
RX_PERSISTENT_MARKER    True
TX_USER_MARKER          True
RX_USER_MARKER          True
TX_MARKER_GEN2_LOC      77
RX_MARKER_GEN2_LOC      77
SUPPORT_ASYMMETRIC      True
LLINK ST
{
  TX_FIFO_DEPTH        1
  RX_FIFO_DEPTH        64
  output user_tkeep  16
  output user_tdata  128
  output user_tuser
  output user_tvalid valid
  input  user_tready ready
}
```

The LLINK at the bottom the Replicated Struct, which is necessarily the AXI-ST as it would appear on the Full Rate.

Eximius Design, Copyright 2021

The USER markers are inserted into the LLINK on the transmit side via a LLINK port called tx_mrk_userbit. This field will be 1, 2 or 4 bits wide depending on the transmit rate (Full, Half or Quarter). The bits effectively travel with the corresponding replicated struct and determine how the receive side will interpret the data. The following are general rules for the markers.

- For symmetric communication (F2F, H2H, Q2Q) the msbit of the markers should be set to 1 and all other bits (if any) should be set to zero.
- For downsizing (e.g. H2F), you'll need to drive the multiple bits to reflect the bits the receiver is expecting. So for H2F, the 2 bit tx_mrk_userbit should both be driven to 1s, meaning each part of the Half Rate is a Full Rate word on the receive side. For Q2H, the 4 bit tx_mrk_userbit should be driven to 0xa since each part of the Quarter Rate will be a Half Rate on the receive side.
- For upsizing (e.g. F2H) the tx_mrk_userbit usually needs to toggle. So for F2H, the 1 bit tx_mrk_userbit should be driven to a 0,1,0,1 pattern indicating every two Full Rate beats is one Half Rate on the receive side. For H2Q, the 2 bit tx_mrk_userbit should have a 0,2,0,2 pattern.

Below are the 9 combinations of Full/Half/Quarter to Full/Half/Quarter and the corresponding USER Marker bit values.

Note that for bidirectional traffic (e.g. AXI-ST with flowcontrol) there is a marker needed on the slave side that is the mirror to the master. So if the master side is F2H, then the slave is necessarily H2F and needs the appropriate marker bits for Half to Full.

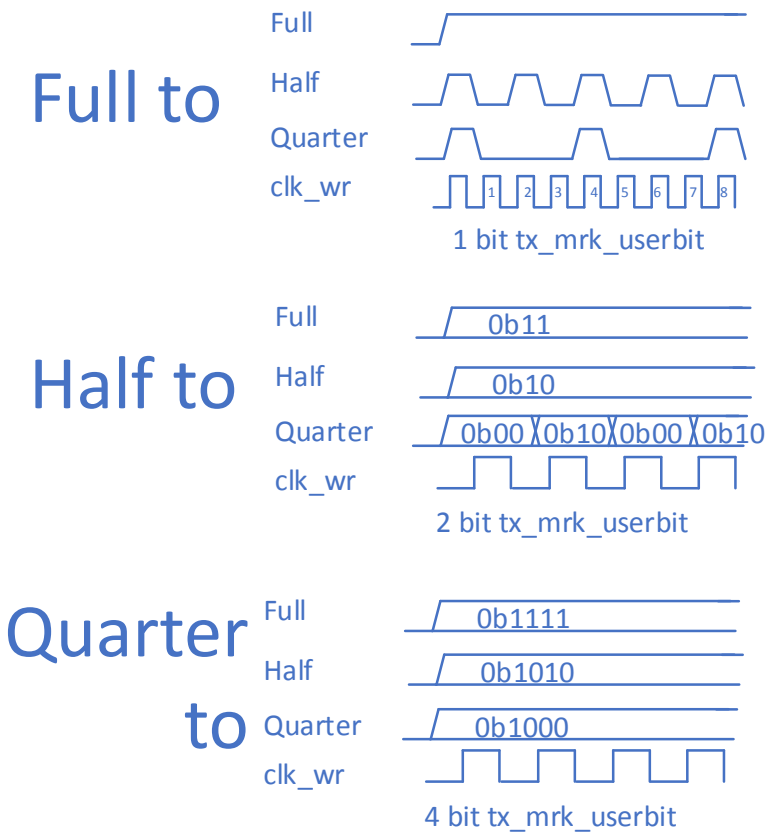


Figure 19 Asymmetric Marker Insertion

Eximius Design, Copyright 2021

5.4.1.7 Info Files

Info files are generated as part of the LLINK script and are used for debug / visualization of the signals. Generally one Info file is made showing the Master's TX and RX signaling, and the Slave RX and TX is inferred as mirror copies of those signals.

The same is true for Asymmetric LLINK, but since Master/Slave can have different Rates, it can be confusing. The rule is to look at the Info file for the side you are debugging. So for a F2H, if debugging the Master, look at the Full Rate Info file. If debugging the Slave, look at the Half Rate Info file, which will detail a Half Rate Master, and infer the Slave signals accordingly.

5.4.2 Example

An example of a Asymmetric mode can be seen here:

${PROJ_DIR}/axi4-st/cfg/axi_st_d256_multichannel.cfg

This uses four 7 Gen1 channels and implements an AXI-ST Full to Half Interface (F2H). This also includes example instantiation of the Channel Alignment block. It uses a simple model for the AIB.

This has a simple, Verilog, Logic Link only testbench here:

${PROJ_DIR}/axi4-st/dv/axi_st_d256_multichannel

The sim can be invoked by executing the run command in that directory:

./run_axi_st_d256_multichannel_f2h

Waves will be generated. The test is setup for Xcellium execution, but the run script can be modified for any simulator.

The test goes through 8 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 – Simple, non overlapping minimum sized transfers

Phase 2 - Overlapping minimum sized transfers

Phase 3 - Simple, non overlapping medium sized transfers

Phase 4 - Overlapping medium transfers

Phase 5 - Simple, non overlapping large sized transfers

Phase 6 - Overlapping large transfers

Phase 7 - Random size, packets

Phase 8 - Overlapping large transfers with no flowcontrol from downstream

5.4.3 Implementation Details

5.4.3.1 Asymmetric with Valid / Ready

The above example of the Asymmetric AXI-ST uses Valid and with Ready and uses the user_enable[] signals to indicate the valid beats of data in upsizing as described above.

The same basic logic holds for Asymmetric AXI-ST using Valid but with no Ready. Note in this case, there is no flow control, and therefore there is no FIFOing in the system, which means there is no delay from the USER Marker input into the LLINK and the rest of the AXI-ST data.

However, for the case where we have Asymmetric AXI-ST using no Valid and no Ready, there is no user_enable[] signal. This is because without the Valid, there is no way to indicate which beats of data are valid and which are not, literally all beats of data are valid. So in this case, the user_enable[] signal is omitted from the receive side.

Eximius Design, Copyright 2021

5.4.3.2 User Interface in Asymmetric Mode

As described above, the Asymmetric flow uses Replicated Structs, which breaks the data into its minimum quanta. So, if a Half Rate side wanted to send a total of 160 bits of TDATA and 20 bits of TKEEP, this would actually be sent across the AIB as two replicated structs of 80 bits of TDATA and 10 bits of TKEEP. The LLINK presents the TDATA and TKEEP as contiguous busses to the USER, so the Replicated Struct functionality is generally hidden from the USER. This can be seen in this diagram:

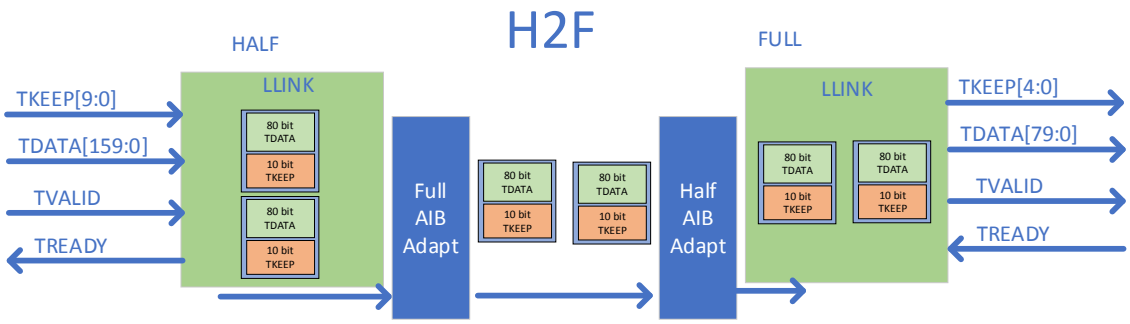


Figure 20 Asymmetric Marker Insertion

So even though the LLINK is configured for the minimum replicated struct, the data busses presented to the USER are dependent on the local AIB rate and effectively emulate a full width AXI-ST interface.

6. AXI-MM

6.1 Fixed Allocation

6.1.1 Theory

This is the simplest of AXI-MM and the version that utilizes the most bandwidth. Fixed allocation refers to the fact that each AXI signal has a corresponding fixed location on the AIB lines, thus ensuring no reduction of AXI bandwidth over AIB.

Conceptually this looks like multiple AXI-ST streams for the 5 AXI Channels in a single AXI-MM interface.

6.1.2 Example

The full configuration file can be found here:

${PROJ_DIR}/axi4-mm/cfg/axi_mm_a32_d128.cfg

This uses four Full Gen2 channels.

This has a simple, Verilog, Logic Link only testbench here:

${PROJ_DIR}/axi4-mm/dv/axi_mm_a32_d128

The sim can be invoked by executing the run command in that directory:

./run_axi_mm_a32_d128

Waves will be generated. The test is setup for Xcellium execution, but the run script can be modified for any simulator.

The test goes through 8 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 – Simple, non overlapping minimum sized transfers

Eximius Design, Copyright 2021

Phase 2 - Overlapping minimum sized transfers

Phase 3 - Simple, non overlapping medium sized transfers

Phase 4 - Overlapping medium transfers

Phase 5 - Simple, non overlapping large sized transfers

Phase 6 - Overlapping large transfers

Phase 7 - Random size, packets

Phase 8 - Overlapping large transfers with no flowcontrol from downstream

6.2 Packetizing

6.2.1 Theory

The packetization option allows for the AXI signals to be time duplexed over the AIB channel. This reduces the total number of wires needed on the AIB, but does result in a reduction of the overall AXI bandwidth and adds a modest increases of latency in the AXI transactions.

However, the AXI bandwidth loss and latency increase can minimized if the AXI traffic pattern can be reasonably predicted. For example, transactions with a large AxLEN will have many W or R beats of data, while having relatively fewer AW, AR or B beats. In this case, sizing the packets to fit an entire W or R beat, while time duplexing the AW, AR and B beats as needed, will produce a maximum throughput for a minimum reduction of BW and low latency. The script takes the effort of finding the most efficient packing scheme.

From the USER perspective, the AXI interface is exactly the same as with Fixed Allocation, with the only noticeable difference is potentially longer latency to cross the AIB.

Packetization is enabled by the config fields. Note these fields are used across the entire configuration and are not restricted to a single Logic Link.

Configuration Option	Values	Description
TX_ENABLE_PACKETIZATION	True, False	Master Transmit Packetization Enable (and Slave Receive Packetization Enable). If False, we use fixed allocation. If true, we use packetization.
RX_ENABLE_PACKETIZATION	True, False	Master Receive Packetization Enable (and Slave Transmit Packetization Enable). If False, we use fixed allocation. If true, we use packetization.
TX_PACKET_MAX_SIZE	Integer Value	Specifies the maximum size of each packet in the Master to Slave direction. Setting to 0 (which is the default and is recommended) will automatically use all available AIB data on the defined channels for each packet.
RX_PACKET_MAX_SIZE	Integer Value	Specifies the maximum size of each packet in the Slave to Master direction. Setting to 0 (which is the default and is recommended) will automatically use

Eximius Design, Copyright 2021

		all available AIB data on the defined channels for each packet.
PACKETIZATION_PACKING_EN	True, False	If set to true, packets may have pieces from multiple AXI channels, potentially resulting in more efficient packetization. If set to False, the default, the algorithm is restricted to only one AXI channel per packet.

Table 9 Packetization Options

6.2.1.1 Implementation Details

Packetization can at first seem complex to understand, but fortunately the script handles most of the work. Referring to the packetizing information in the INFO.txt file may help to visualize the packing. The easiest option to implement packetizing is to set the TX|RX_ENABLE_PACKETIZING to True and the script will take care of the rest. This will calculate the number of available data bits available for use in packets and implement the packetizing accordingly.

For example, in a single Gen2 Full rate channel with no strobes, no markers and no DBI, we'll have exactly 80 bits data available to be used for AXI data. All 80 bits will be used for the packet size. This means that a W channel with 128 bit WDATA field will likely be split into two packets and reassembled on the far side. Similarly, AR channel with 32 bit ARADDR would likely fit in a single packet.

If however, the DBI field was enabled, then 4 of the 80 bits would be unavailable for data, and the packets would be correspondingly smaller, which could cause additional fragmentation of the AXI channel into smaller packets. By default the script will calculate this on the fly and fit the packet to the maximum data available on the AIB channel(s).

6.2.1.1.1 Examples of Packetization

The details of the packetization depends heavily on the AXI Channel details. For this example, let's assume we are using the Full Gen2 channel without markers, strobes or DBI, meaning 80 bits of data available for AXI. We'll constrain the conversation to the AXI master, i.e. the AR, AW and W busses only.

Each packet is comprised of three pieces:

1. Header – Used to by the receive side to reassemble the data. The size is necessarily the log2 of the number of packets (rounding up).
2. Packet Data – This varies by packet, but could contain pieces from one or more AXI channels
3. Return Credits – These are the credits for other AXI channels (other than the ones being packetized).

But the actual format of the data depends on the channels and the configuration. For example, let's assume the AR and AW channel each have a total of 50 bits and the W has 140 bits of data. Using packetization, we'd have four packets:

1. The only AR packet – 50 bits of AR data, 2 bits of credit (R and B), 2 bits of Header, 26 bits unused.
2. The only AW packet – 50 bits of AW data, 2 bits of credit (R and B), 2 bits of Header, 26 bits unused.
3. The first W packet – 64 bits of W data, 2 bits of credit (R and B), 2 bits of Header, 12 bits unused.
4. The second W packet – 76 bits of W data, 2 bits of credit (R and B), 2 bits of Header, 0 bits unused.

The header in the above example is a 2 bit header, sufficient to differentiate the four different packet types. The actual data is dependent on the LLINK configuration. See the INFO file for more information.

Eximius Design, Copyright 2021

As another example, a single Gen2 Quarter rate channel with no strobes and recovered markers and no DBI will have 320 bits of data available. That same 128 bit data width W bus will likely fit in a single packet, and likewise the AR and AW will fit into two additional packets. That would look like these formats:

1. The only AR packet – 50 bits of AR data, 2 bits of credit (R and B), 2 bits of Header, 266 bits unused.
2. The only AW packet – 50 bits of AW data, 2 bits of credit (R and B), 2 bits of Header, 266 bits unused.
3. The only W packet – 140 bits of W data, 2 bits of credit (R and B), 2 bits of Header, 176 bits unused.

Again, the header above needs 2 bits to identify the 3 packet types. But obviously, there is significant unused space in the above sequence. This is because the “advanced” packing option is off by default, so each AXI channel is forced to be in a single packet.

However, if PACKETIZATION_PACKING_EN is changed from its default False to True, the algorithm attempts to pack the disparate AXI channels into a single packet to improve throughput. Using the Gen2 Quarter rate example, this would result in:

1. One packet – 50 bits of AR, 50 bits of AW, 140 bits of W, 2 bits of credit (R and B), 0 bits of Header, 78 bits unused.

The header above needs 0 bits to identify the only packet type, so there is no header (i.e. all packets are the same type). Note that in this case, the packetization looks practically identical to fixed allocation since every AXI channel bit has an exclusive bit location in the AIB fields.

Additionally, the user can attempt to manually constrain the size of the packet by setting TX|RX_PACKET_MAX_SIZE. In the above Full Gen2 example, the packet size is set to 80. If the user wanted to, they could reduce the packet size to say 40 bits wide. This would result in more packets (approximately two AR packets, two AW packets and four W packets, and a corresponding increase in the Header size. More specifically

1. The first AR packet – 15 bits of AR data, 2 bits of credit (R and B), 3 bits of Header, 20 bits unused.
2. The second AR packet – 35 bits of AR data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.
3. The first AW packet – 15 bits of AW data, 2 bits of credit (R and B), 3 bits of Header, 20 bits unused.
4. The second AW packet – 35 bits of AW data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.
5. The first W packet – 35 bits of W data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.
6. The second W packet – 35 bits of W data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.
7. The third W packet – 35 bits of W data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.
8. The fourth W packet – 35 bits of W data, 2 bits of credit (R and B), 3 bits of Header, 0 bits unused.

The header above needs 3 bits to identify the 8 packet types.

The value of artificially constraining is not particularly interesting in packetizing alone, but it can be used to great effect in Tiered Logic Links.

6.2.1.1.2 Effects of MAX_PACKET_SIZE

The following may explain the packing combinations. Below are packetizing A-E. Each represents the same AXI-MM interface, with the colored boxes representing the AIB Transmit from the AXI interface. For example purposes, lets assume the AR and AW are 50 bits wide, while the W is 100 bits wide. The R and B are single bit credits used for the R and B flow control. The below examples shows what happens when you constrain the MAX_PACKET_SIZE to specific values.

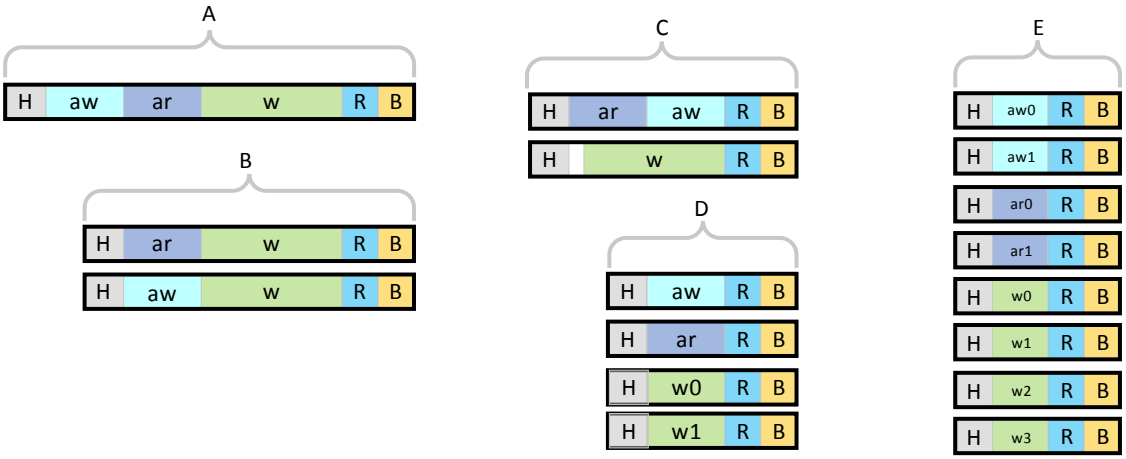


Figure 21 Packetization Visualization

Option A has a MAX_PACKET_SIZE of 202 bits, which results in a single packet. In this corner case, the Header (H) is removed from the flow as there is only one packet and no need to differentiate different packets. This is the highest AXI bandwidth option.

Option B has a MAX_PACKET_SIZE of 153 bits, which allows space for the W to be sent in every packet, and only the AW and AR have to arbitrate. Assuming the AW and AR burst lengths are above the minim value, this should provide max AXI bandwidth. H is 1 bit wide.

Option C has a MAX_PACKET_SIZE of 103 bits, which allows for the space for the W to be sent in every packet and then the AW/AR in another packet. In this case, the max W bandwidth is shared with the AW/AR bandwidth. For minimum burst lengths, this effectively halves the AXI bandwidth because we can only send one AW and then one W. But for larger burst lengths (e.g. burst length of 16) will allow for 16/17th AXI bandwidth. H is 1 bit wide.

Option D has a MAX_PACKET_SIZE of 54 bits, which allows for AR and AW to be sent in individual packets, but the W is broken into two pieces. Even with very large burst lengths, the effective W bandwidth is half of the max AXI interface, simply because we must send two packets to form one beat of W. H is 2 bit wide.

Option E has a MAX_PACKET_SIZE of 30 bits. This means AR and AW are split into two packets and W is split into four packets. The AXI interface uses few AIB wires at the cost of overall AXI-MM bandwidth. H is 3 bit wide.

The above shows the effect of MAX_PACKET_SIZE. Typical applications can simply specify the AIB channels available, and the script will set the MAX_PACKET_SIZE to the maximum value. Users who are concerned about achieving the maximum bandwidth should understand the ramifications of MAX_PACKET_SIZE and adjust the AIB configuration (channels / rate) or MAX_PACKET_SIZE accordingly.

6.2.1.1.3 Algorithm

The packetization algorithm is simplistic, but it can be explained in the following pseudocode.

0. HeaderSize = 0; DataSize = MAX_PACKET_SIZE
1. Break up the AXI-MM channels into DataSize chunks.
2. Generate all combinations of Chunks to form all Potential Packets.
    a. For each Chunk:
        i. Create Potential Packet with One Chunk
        ii. Check if Potential Packet Sequence is Legal

Eximius Design, Copyright 2021

            iii. If there is still space available and PACKETIZING_PACKING_EN, add in second Chunk and create new Potential Packet
        b. Result should be a list of Potential Packets that fit within DataSize
3. Generate Packet List
    a. Sort Potential Packets to choose largest packets first
    b. For each sorted Potential Packet
        i. Are there unique Chunks not already in Packet List? If so, skip that Potential Packet. Otherwise, add Potential Packet to the Packet List.
    c. Results should be a list of Potential Packets that transfer the required data in the fewest number of packets.
4. Check Results.
    a. HeaderSize = log2 (Number of Packets in List)
    b. DataSize = MAX_PACKET_SIZE - HeaderSize
    c. Did HeaderSize change from previous value? If so, goto step 2 and recalculate with updated DataSize. Do not progress until we've settled on HeaderSize
5. We now have final Packet List which should be optimized for maximum data in minimum packets.

Note the HeaderSize is both an input into the Algorithm (i.e. reducing available space in DataSize) and is an output of the Algorithm (based off of the number of packets). As a result, we have the iterative process as shown above. For example, if we have a total of 100 bits of AXIMM data to send and DataSize is 50, I could theoretically do that in 2 packets, but this would require a 1 bit Header, which means DataSize maxes out at 49, which means we'll need at least 3 packets, which requires 2 bit Header, etc. With reasonable packet sizes, this process should converge, but with irrational packet sizes (e.g. MAX_PACKET_SIZE < 10), the process could grow exponentially so a hard limit of a maximum of 100 individual packets is implemented to prevent the process from running away.

6.2.2 **Example**

The full configuration file can be found here:

${PROJ_DIR}/axi4-mm/cfg/axi_mm_a32_d128_packet.cfg

This uses one Full Gen2 channel.

This has a simple, Verilog, Logic Link only testbench here:

${PROJ_DIR}/axi4-mm/dv/axi_mm_a32_d128_packet

The sim can be invoked by executing the run command in that directory:

./run_axi_mm_a32_d128_packet

Waves will be generated. The test is setup for Xcellium execution, but the run script can be modified for any simulator.

The test goes through 8 different phases, which can be seen by the TestPhase variable at the top:

Phase 1 – Simple, non overlapping minimum sized transfers

Phase 2 - Overlapping minimum sized transfers

Phase 3 - Simple, non overlapping medium sized transfers

Phase 4 - Overlapping medium transfers

Phase 5 - Simple, non overlapping large sized transfers

Phase 6 - Overlapping large transfers

Phase 7 - Random size, packets

Phase 8 - Overlapping large transfers with no flowcontrol from downstream

Eximius Design, Copyright 2021

Note that the only difference between axi_mm_a32_d128 and axi_mm_a32_d128_packet is the packetization and a reduction in the number of AIB channels. This makes it reasonable to compare both implementations.

7. Advanced Topics

7.1 Auto synchronization

Auto synchronization is a mechanism to have both ends of the AIB Protocols synchronize to the Marker Word Alignment and Channel Alignment Strobes without external control or influence using signals from the AIB PHY and some predetermined delays suitable to ensure the Word Alignment and Channel Alignment have been completed. This is also the path to ensure that automatic recovery of Strobes and / Markers to maximize the use of the AIB bandwidth.

Note that this mechanism supports recoverable strobes and markers when generated from the USER and recoverable strobes when generated from the CA. It does not support recoverable markers when generated by the AIB Adapter.

7.1.1 **Theory**

Autosynchronization is achieved by using some key signals from the AIB and adding delays to those signals to ensure that both sides of the interface are able to receive before the AXI data is transmitted. The auto synchronization sequence is keyed off of the assertion of the TX_ONLINE and RX_ONLINE signals on the LLINK (and possibly CA) modules.

The ONLINE signals are driven differently depending on the presence of a Channel Alignment (CA) block in the system, which generally is only present if multiple AIB channels are used.

	With CA	Without CA
llink.tx_online	ms_tx_transfer_en & sl_tx_transfer_en	ms_tx_transfer_en & sl_tx_transfer_en
ca.tx_online	ms_tx_transfer_en & sl_tx_transfer_en	n/a
ca.rx_online	ms_tx_transfer_en & sl_tx_transfer_en	n/a
llink.rx_online	ca.rx_align_done	ms_tx_transfer_en & sl_tx_transfer_en

These delays are listed here:

- delay_x_value – This is defined as the time after the AIB asserts ms_tx_transfer_en and sl_tx_transfer_en before the word alignment is guaranteed to be completed and downstream logic can begin to interpret the receive data. Before this point, the AIB DLL is calibrating and word alignment is not guaranteed, so the receive data is considered random.
- delay_y_value – This is defined as the time after the transmission of the Strobes before the remote side is ready to receive AXI data. This is dominated by the Z value, which effectively could slow down the remote side's readiness.
- delay_z_value – This is defined as the max time after one side of the AIB interface (follower or leader) asserts the local view of ms_tx_transfer_en and sl_tx_transfer_en before the remote side also sees ms_tx_transfer_en and sl_tx_transfer_en. The majority of this delay is due to the AIB

Eximius Design, Copyright 2021

shift delay of transferring the signals across the AIB and the relative speed of the shift register clock.

Internally to the logic, the online signals are delayed by the X,Y and Z delays to ensure valid synchronization.

The following diagram highlights the phases the transmit and receive paths go through, and what the delays are intended to cover.

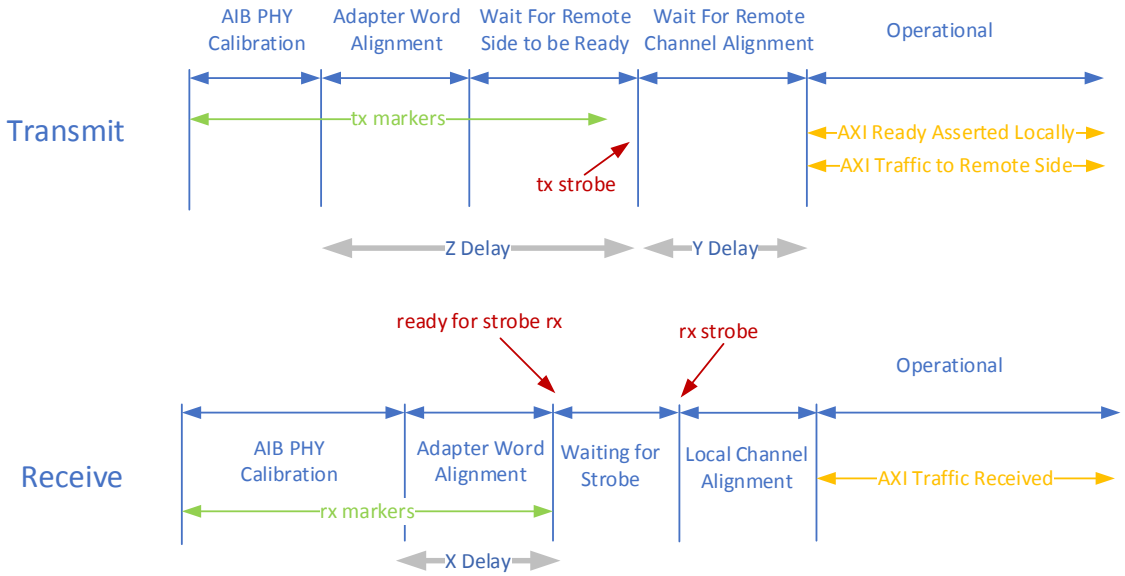


Figure 22 Auto Synchronization Sequence

As can be seen above, the markers and strobes are only required for a relatively short time before the AXI traffic flows. For this reason, they can be recovered and reused as additional bits of data. Note that the term Recoverable indicates the marker / strobes can be reused as AXI data, while Persistent indicates the opposite, namely that markers / strobes are always present and cannot be reused as AXI data.

Note, with the above flow it is possible for the local Transmit path to be inactive, while the Receive path is active, that is one side of the LLINK interface is open to receiving but cannot transmit back. In this situation, data will be received and held in the local FIFO until the Transmit path is ready to return credits.

Note that the above sequence uses AXI flow control signals (e.g. AWREADY or TREADY) to hold off the interface until the AIB system is ready for traffic. For AXI-ST without READY operations, any data transmitted will be lost until both sides are fully up and ready. If the user delays the initial transmission of the AXI-ST data by delay_x_value and delay_y_value, then the first beat of AXI-ST data should be successfully received on the remote side.

7.1.2 Recommended minimum values.

Below are the recommended values for the LLINK and CA. Other values may work, but these should provide a stable setup without significant analysis.

	Local Rate			Used in Block
	Full	Half	Quarter	

Eximius Design, Copyright 2021

delay_x_value[15:0]	8000	4000	2000	LLINK, CA
delay_y_value[15:0]	9000	4500	2250	LLINK
delay_z_value[15:0]	32	16	8	LLINK, CA
All values are decimal and are expressed in terms of the local m_wr_clk.				

7.1.3 Auto synchronization without Flow Control

In AXI-MM and in AXI-ST with TREADY, the LLINK forces the READY signals of the AXI interface low until the strobes / markers have been recovered, ensuring that the data sent across is not “corrupted” by the marker or strobe insertion. However, in AXI-ST without TREADY, there is no flow control and the USER should hold off transmission of meaningful data until the strobe/marker has been recovered or the data sent across will have some bits corrupted (anywhere from one to five bits per channel, depending on configuration). Generally speaking, the USER should wait for at least the X+Y+Z time above after the ms_tx_transfer_en and sl_tx_transfer_en are high. However, a safer value would be to wait approximately 20 us (roughly 2x X+Y+Z) to ensure everything has quiesced before sending the data across.

Alternatively, bits [19] and [18] of the tx_*_debug_status or rx_*_debug_status contains the internally delayed versions of TX_ONLINE and RX_ONLINE, respectively. Waiting until both of these signals are high (plus a few cycles for good measure[4]) will ensure that the strobes / markers have been recovered and the full data width should be available.

7.1.4 Auto synchronization with USER Inputs

USER inserted recoverable strobes are a little unorthodox, but are supported. Without USER input strobes, the LLINK would wait delay_z_value cycles before sending the strobe. But if the USER input strobe is asserted, the LLINK will wait delay_z_values and then wait until the second cycle of USER inserted strobe. This ensures we wait the minimum Z time, but also obey the USER inserted strobe time. However, if the period between USER inserted strobes is long, this can cause the transmit path to be correspondingly delayed. For this reason, USER inserted strobe periods should not be excessively long (e.g. thousands of cycles), but should max out at something more reasonable like 256 cycles or less.

A similar situation arises with the USER inserted marker, where the strobe transmission will be synchronized to the marker. However, since the marker period is 4 or less, the delay caused by waiting for the marker is generally negligible. However, it is important that if the USER marker option is selected, that the USER continue to drive the marker, even in the Full Rate case where the marker should be effectively always high. If the USER inserted marker or USER inserted strobe are permanently tied low, this can lockup the Auto Synchronization process.

[4] A few cycles is maybe 4 or 8 cycles (each cycle is 0.5 to 2ns depending on rate) and is intended to give some buffer to ensure the strobe/marker is fully recovered.

Eximius Design, Copyright 2021

7.2 Tiered Logic Link

It should be noted this feature is advanced and is only intended for users already familiar with LLINK functionality who want to squeeze additional bandwidth out of the AIB interface(s). Casual users should generally never need this advanced feature.

The LLINK script supports a mechanism called Tiered Logic Link, which provides a mechanism for the USER to assert more control over how various AXI interfaces are to be packed onto the AIB. This can be useful in a couple of specific situations:

- Stitching together multiple AXI interfaces, including AXI-ST and AXI-MM, onto a single set of AIB channels. This can be useful for packing AIB channels (i.e. using unused bits from AIB channel in an AXI interface primarily on another AIB channel) or constraining the AIB bandwidth of specific AXI channels (e.g. four AXI-MM interfaces evenly split the bandwidth across a set of AIB channels would guarantee each AXI channel has the same bandwidth).
- Similarly, providing a mechanism for lower level control of the AXI across non-uniform AIB channels (e.g. splitting a single AXI LLINK interface so it uses 10 bits of one channel, 20 bits of a second channel, and 30 bits of a third).

For reference, this is the normal, non-tired Logic Link. Note that the core of the LLINK IP is in green and the AIB channel specific pieces (channel count, strobe location, etc) are in the grey cloud.

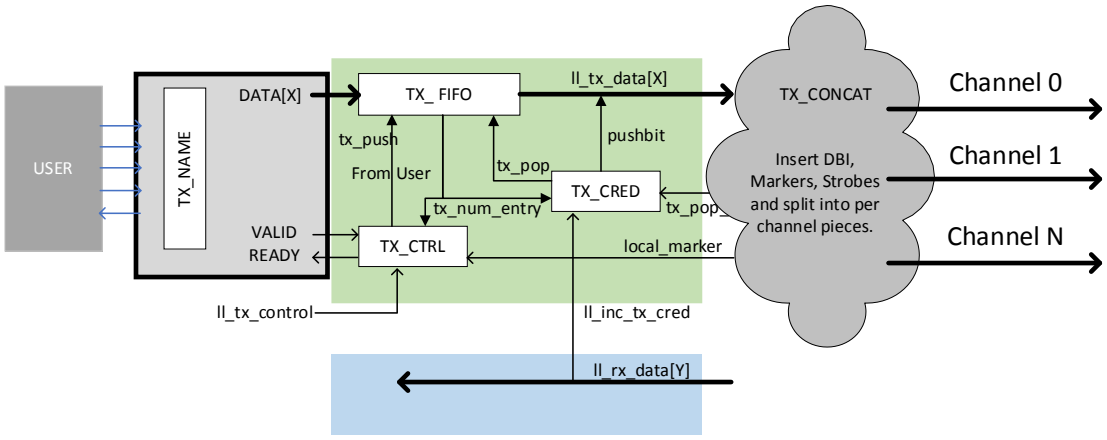


Figure 23 Non-Tiered LLINK

Compare the above with the below simplistic view of the Tiered Logic Link, which is effectively two LLINK. The left side is Tier1 which is the side connected to the USER interface. Tier1 has no information related to Marker, Strobe, DBI and even AIB Channel specific settings and produces a single input and output interface (tx_phy0 and rx_phy0) that is sized to the targeted LLINK data path.

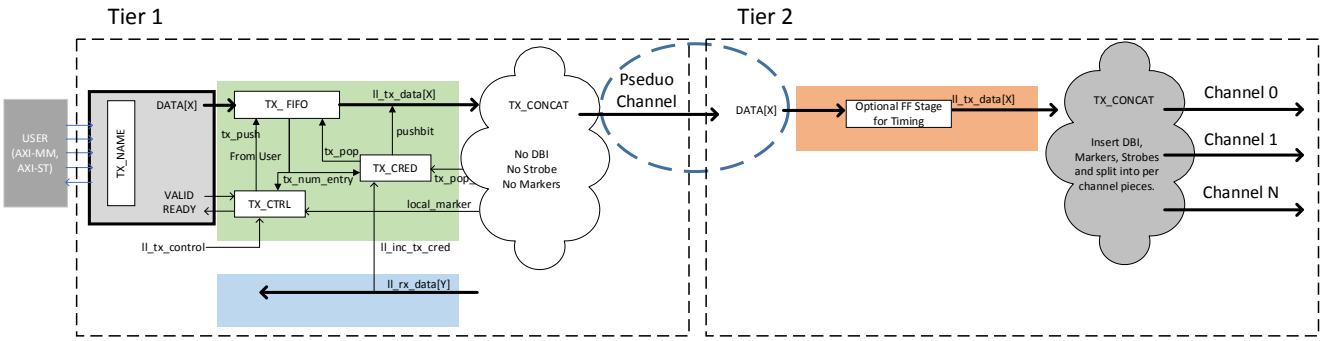


Figure 24 Tiered LLINK

Eximius Design, Copyright 2021

The right side is Tier2 Logic Link which interfaces to the AIB Channel. The Tier2 has all the channel specific configurations including Channel Type, Markers, Strobes, DBIs, etc. The input of the Tier2 is configured to be a two Logic Links, one for all transmit vectors and one for all receive vectors, both with no-VALID and no-READY, which makes them effectively passthrough.

The actual interconnect between Tier1 and Tier2, designated by the blue circle above, is left for the user. This should be nothing more than connecting a few wires.

Simplistically, a standard, simple AXI-MM or AXI-ST LLINK can be split into two pieces, Tier1 and Tier2. The value of this will become more apparent walking through these examples.

There is one implementation caveat for the Tiered implementation. As discussed in Auto Synchronization, the USER inserted markers and strobe can cause a delay in the time to recover markers and strobes. This delay only effects the Tier2 side because the Tier1 has no strobe/marker configuration. As a result, Tier1 can go online and begin transmitting data before Tier2 has finished using the strobes / markers and recovered them for data.

To work around this, connect the connect the Tier1's tx_online signals to Tier2's debug_status[19] which is the delayed version of Tier2's tx_online. This will ensure that Tier1 does not begin operations until Tier2 has recovered its markers / strobes, i.e. after the Z delay, Y delay and any USER inserted Strobe / Marker delays. In this configuration Tier1's delay_z_value and delay_y_value can be tied to 0 for a modest performance improvement (<5us) but it is not necessary.

Tier1's and Tier2's rx_online should both be connected to the CA.align_done as per normal.

7.2.1 Example #1 – Multiple AXI-MM Interfaces

For this example, we want to define four identical AXI-MM Interfaces, each with the same configuration of AXI-MM channels (i.e. same AW, AR, W, R, B) and with packetization and then feed these into a single AIB channel operating in Quarter Rate. As a result, each of the four AXI-MM Interfaces will utilize one fourth of the AIB bandwidth, while still supporting per channel options such as DBI or strobe. The following are step by step examples on how to generate this sequence.

The first step is to calculate the desired MAX_PACKET_SIZE. In a non-tiered design, this is calculated by the script and is a product of the AIB Configuration. We can calculate the same value ourselves using these general formulas.

Note, PERST_ indicates persistent strobes or markers. Recoverable strobes or markers are, as stated, recoverable, and can be reused as data so do not reduce the available data on the AIB.

```
AIB_MAX_CHAN_WIDTH = 40,80,80,160,320 for Gen1Full, Gen1Half, Gen2Full,
Gen2Half, Gen2Quarter

MARKERS_PER_CHANNEL = 1,2,1,2,4 for Gen1Full, Gen1Half, Gen2Full, Gen2Half,
Gen2Quarter (Full versions can also be 0, depending on user usage)

DBI_PER_CHANNEL = 0,0,4,8,16 for Gen1Full, Gen1Half, Gen2Full, Gen2Half,
Gen2Quarter

USEABLE_CHAN_WIDTH = AIB_CHANNEL_WIDTH – PERST_STROBE*1 –
PERST_MARKERS*MARKERS_PER_CHANNEL – DBI_ENABLE*DBI_PER_CHANNEL

MAX_PACKET_SIZE = ROUNDDOWN((USEABLE_CHAN_WIDTH * NUMBER_CHANNELS) /
NUMBER_AXI_INTERFACES)
```

Alternatively, this can be found via "experimentation" by entering a guess value int the Tier2 and observing the resulting INFO file, and from that see how much space is available for use.

The Tier2 logic links is then two sets of llink, one for TX and one for RX. Each set will have one entry for each AXI Channel. The size of each port is necessarily the MAX_PACKET_SIZE. So for example, this is the result of a single, Gen2 AIB channel with no strobe, with persistent markers and DBI. The resulting MAX_PACKET_SIZE is 75 bits.

Eximius Design, Copyright 2021

```
llink tx
{
  output ch0_tx_data  75
  output ch1_tx_data  75
  output ch2_tx_data  75
  output ch3_tx_data  75
}

llink rx
{
  input ch0_rx_data   75
  input ch1_rx_data   75
  input ch2_rx_data   75
  input ch3_rx_data   75
}
```

The Tier1 is configured to be an AXI-MM with the TX_PACKET_MAX_SIZE and RX_PACKET_MAX_SIZE set to 75. Tier1 also has CHAN_TYPE set to “Tiered”. When built, this will build a single AXI-MM LLINK with packet size set to 75. The Tier1 tx_phy0 will be exactly 75 bits wide, and the rx_phy will be similarly sized.

If we then take four instances of Tier1 and one instance of Tier2, we can connect the signals for the TX and RX busses. We’ll put these 5 instances in top level file for the master and similar file for the slave. Connecting these signals can be done by hand or done trivially with one of several Verilog autoconnect tools like Emac’s AUTOs. Putting everything into a single top level module will simplify the design but is not strictly required, but if the logic is encased in a top level module, the module looks identical to a standard LLINK with 4x AXIMM interfaces and the standard interfaces to the CA / AIB PHY.

As a result, we end up with a design similar to what is in the following figure. Note for simplicity the figure only shows the transmit path, but there is a receive path that is nearly identical but in the reverse direction.

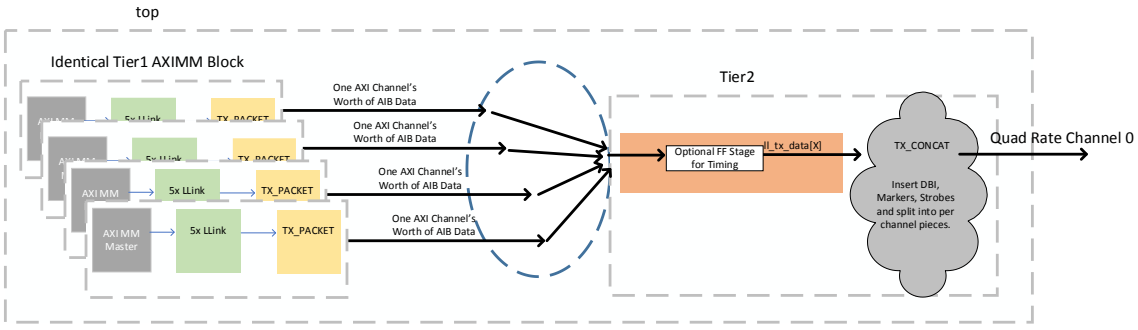


Figure 25 Tiered AXI-MM LLINK

7.2.2 Example #2 – Mixing / Matching AXI-ST and AXI-MM

To support mixing AXI-ST and AXI-MM across a single or multiple AIB channels, the process is very similar to Example #1. Similar to the above example, we use the same basic equations to determine the MAX_PACKET_SIZE for the AXI-MM.

Eximius Design, Copyright 2021

So, if we have a Gen2 Half Rate single channel interface, with persistent markers and DBI and no strobe, we'll have 154 bits available for data. If the AXI-ST uses 128 bits of data and a valid, we'll use 129 bits of that for the AXI-ST, leaving only 25 bits for the AXI-MM MAX_PACKET_SIZE.

So, we'll feed this into a Tier2 reserving 129 bits for the AXIST and 25 bits for the AXI-MM, as shown below

```
llink tx
{
  output axist_tx_data  129
  output aximm_tx_data  25
}


llink rx
{
  input axist_rx_data   1
  input aximm_rx_data   153
}
```

Note the RX side does not use much data for the RX side of the AXI-ST, so most of the 154 bits available can be used for the AXIMM.

When Tier1 AXIMM is built, the TX_MAX_PACKET_SIZE should be 25, and the RX_MAX_PACKET_SIZE can be 153.

7.2.3 Example #3 – Splitting a single AXI Interface Across Multiple AIB Channels

The previous examples have shown how to combine multiple AXI interfaces across a single AIB channel. The normal LLINK functionality can split an AXI Interface across multiple AIB channels. However, the LLINK will be configured to be the sole user of the specified AIB channels, which may result in some AIB data channels being unused. So to achieve the maximum usage of the AIB, we may want to have multiple LLINK across multiple AIB channels, but not necessarily constrained to a single LLINK per set of AIB channels.

Supporting this is relatively trivial via the Tiered LogicLink. As an example, lets assume we have a single AXI LLINK we want to use across three Gen2 Half Rate AIB channels called A B and C. Normally, each channel would provide 160 bits of user data per channel, but for this example, lets assume there are 10 bits of AIB overhead (DBI, markers, etc), so we can only get 150 bits of user data per channel. Furthermore, lets assume some other entity (e.g. other LLINKs) is making use of parts of Channels A and C, so only 110 bits on Channel A and 50 bits of Channel C are available for us.

To summarize:

- AXI Data is 290 bit that is to be split across Chan A, B, C
- Chan A – has 110 user data bits available.
- Chan B – has 150 user data bits available.
- Chan C – has 50 user data bits available.

The AXI Data should be build as a Tier 1 with CHAN_TYPE = Tiered. This will produce the raw 290 bit output.

Channel A, B and C should all be Tier2 configured for Gen2 Half Rate with the AIB overhead.

Then the Tier1 AXI Data output can be split amongst the 3 channels. A reasonable split of the 290 bits can be 110, 150, 30 bits for Chan A, B and C. This is shown in the following diagram.

Eximius Design, Copyright 2021

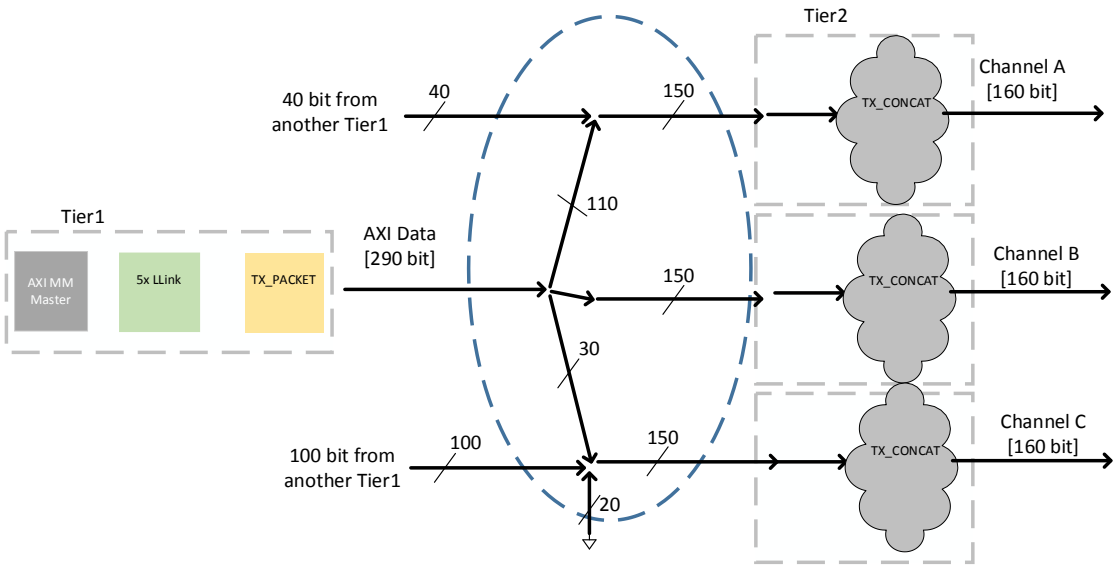


Figure 26 Tiered LLINK across Multiple Channels

Note that the AXI Data is 290 bits wide, but between Chan A, B and C, we have 310 bits available. This means 20 bits should be tied off to zero or reused in some other capacity.

Using the above scheme, many LLINK can be packed across multiple AIB Channels to achieve the maximum benefit from the AIB Channels. However, for this to work, there are a few potentially obvious caveats. Namely:

A. All channels must be operating at the same rate, i.e. Full, Half or Quarter.
B. All channels should be Channel Aligned together.

Eximius Design, Copyright 2021