SHIP IP User Guide

Logic Link (AXI-MM and AXI-ST)

Revision: 0.1

17 Aug 2021

Table of Contents

Table of Figures

Table of Tables

Revision History

Revision	Date	Author	Summary of Changes
0.1	8/20/21	John	Initial draft

Table 1 Revision History

1. Overview

The Logic Link (LLINK) IP is used to provide an interface between AMBA AXI Protocols (AXI-MM) and AMBA AXI-ST interfaces 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.

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.

1.5 Summary

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

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.

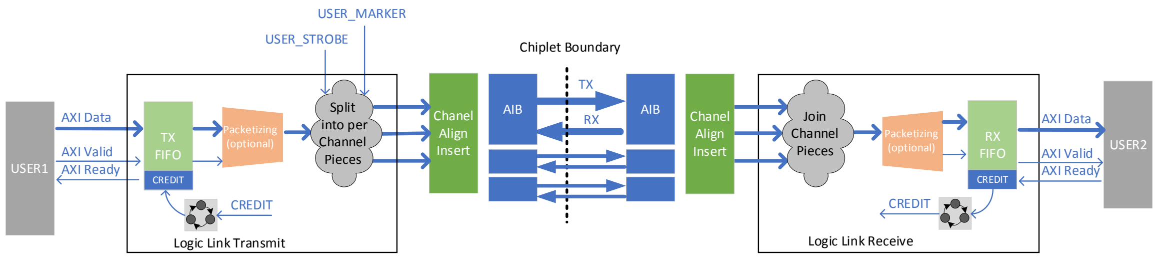


Figure 1 Logic Link Overview

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. 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 marge the Logic Link data (e.g. AXI) onto the AIB data channel with its various overhead.

The 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.

Question: Do we have a good rule of thumb for latency for 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 is called llink_gen.py. This is a python script, so it requires a python executable to run. Python comes in two major flavor, Python2 and Python3, and the script works for both. 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
```

will pick up the configuration file specified and by default put the resulting RTL in a directory called ./axi_st_d64.

3.1 Script Output

Generally, the script generates a two sets of RTL, one is a master and one is a slave. The name of the module and is defined by the MODULE_NAME configuration in the config file. The outputs are are listed below. Note that the .f files use the environment variable PROJ_DIR which is expected to point to the level above the llink directory.

For Each LL	Description
<MODULE_NAME>_info.txt	Text based information file. This contain detailed information on the channel configuration, usage, AIB overhead and specific locations of key signals in the AIB channel. This should be the first stop to looking at the RTL. This file, like the rest of the design, is written from the Master's perspective. So the Master's TX is necessarily the Slave's RX and 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	User side interface module
<MODULE_NAME>_master_concat.sv	PHY / CA interface module
<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 master module
<MODULE_NAME>_slave_name.sv	User side interface module
<MODULE_NAME>_slave_name.sv	PHY / CA interface module

Asymmetric Variant

For the Asymmetric Gearboxing case, instead of a single set of Master / Slave there are multiple Masters and Slaves generated in the same directory. This produces a set 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 example, 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.
--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.

Table 3 Command Line Options

3.3 Example invocations

// Simple python invocation

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

// Explicitly 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

// Invocation 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 and	Values	Description
MODULE	Any	Name of the generated RTL. By convention, we name the .cfg the same as the module name.
Channel Description		
NUM_CHAN	1-24	Number of AIB Channels
CHAN_TYPE	Gen1Only, Gen2Only, Gen2	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.
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.

Table 4 Module Name and Channel Description Configuration

4.2 Strobe

These configure the Strobe functionality as controlled from the Logic Link. Note that it is possible downstream blocks like the Channel Alignment may override the values coming out of the Logic Link. Note there are generally one Strobe bit per channel.

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 DBI data stream. If False, the Strobe is recoverable and can be used as 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 DBI data stream. If False, the Strobe is recoverable and can be used as 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. Note, downstream logic like CA may override the strobe if configured to do so.
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. Note, downstream logic like CA may override the strobe if configured to do so.
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.
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.
TX_STROBE_GEN1_LOC	0-79	Master Transmit Gen1 Strobe Location (and Slave Receive 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.
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.

Table 5 Strobe Configuration

4.3 Marker

These configure the Marker functionality as controlled from the Logic Link. Note that 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

[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.

		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 DBI data stream. If False, the Marker is recoverable and can be used as 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 DBI data stream. If False, the Marker is recoverable and can be used as data after synchronization.
TX_USER_MARKER	True, False	Master Transmit USER Marker Enable. If True, then the USER can drive the Marker via a port on the Logic Link. If False, the Marker is 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 via a port on the Logic Link. If False, the Marker is 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. 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
  output user_tvalid valid
  input  user_tready ready
}
```

The Logic Link name will be used internally for grouping key signals.

Each Logic Link can be configured with different Transmit and Receive FIFO Depths as shown above.

The signals associated with the Logic Link are indicated here too. Each sginal 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.

The presence of the optional_ width will make the signal a bus of the width specified. If no width is specified, then the signal is a 1 bit scaler. The optional_lsb will provide an least significant bit offset for the bus, but the overall width is still specified by optional_width. So this example will result in a signal called user_signal1[9:2].

```
  input user_signal1 8 2
```

Note that in place of the optional width, a keyword of "valid" or "ready" can be used to indicate the Valid / Ready of the AXI Channel. These are single bit scalers.

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.

Below is a list of the IO

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.
delay_x_value[7:0]	input	Delay used in Auto Synchronization. Tie to 0 if unused.
delay_xz_value[7:0]	input	Delay used in Auto Synchronization. Tie to 0 if unused.
delay_yz_value[7:0]	input	Delay used in Auto Synchronization. Tie to 0 if unused.
tx_st_debug_status	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 7 Common Logic Link Ports

The below are configuration specific ports.

Port Name	Direction	Description
tx_st_debug_status	output	Debug signal only present if there is a Transmit Logic Link. [31:24] – Current TX Credits [23:18] – rsvd [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_st_debug_status	output	Debug signal only present if there is a Receive Logic Link. [31:18] – rsvd [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]	input/output	Configuration specific.

user_tvalid 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 installation.
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));
```

5. AXI-ST

5.1 *Simple, Fixed Allocation*

This is the no frills, simply 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. There is also a TLAST bit.

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 command.

```
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. Generally, this should be set to 1, but there are a limited set of systems that may benefit from a deeper FIFO, allowing the AXI transaction to be pulled off of the shared AXI bus, but it could be stalled in the TX_FIFO if flow control is asserted.

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.

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

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.

TBD

5.2.2 Example

TBD

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. 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 Gen2/Gen1 Rates allow for the same clock speed on either side. 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, other Strobe/Marker settings like USER inserted and Persistent / Recoverable Strobe and Markers will maintain the same configuration when switching from Gen2 to Gen1. Note too that Gen2 may have DBI enabled, but since Gen1 does not support DBI, this option will be ignore 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 Logic Link configuration allows for a Gen2 AXI-ST. The Gen1 AXI-ST must be a subset of the Gen2 configuration, that is no new signals can be defined for Gen1.

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
  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 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 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 behave.

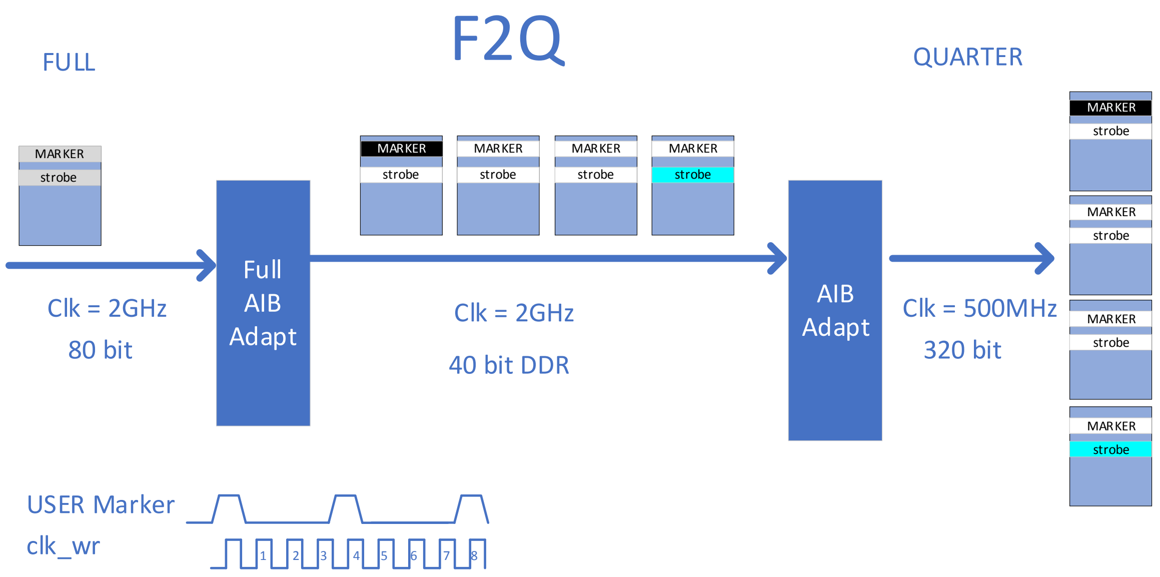


Figure 2 Asymmetric F2Q

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

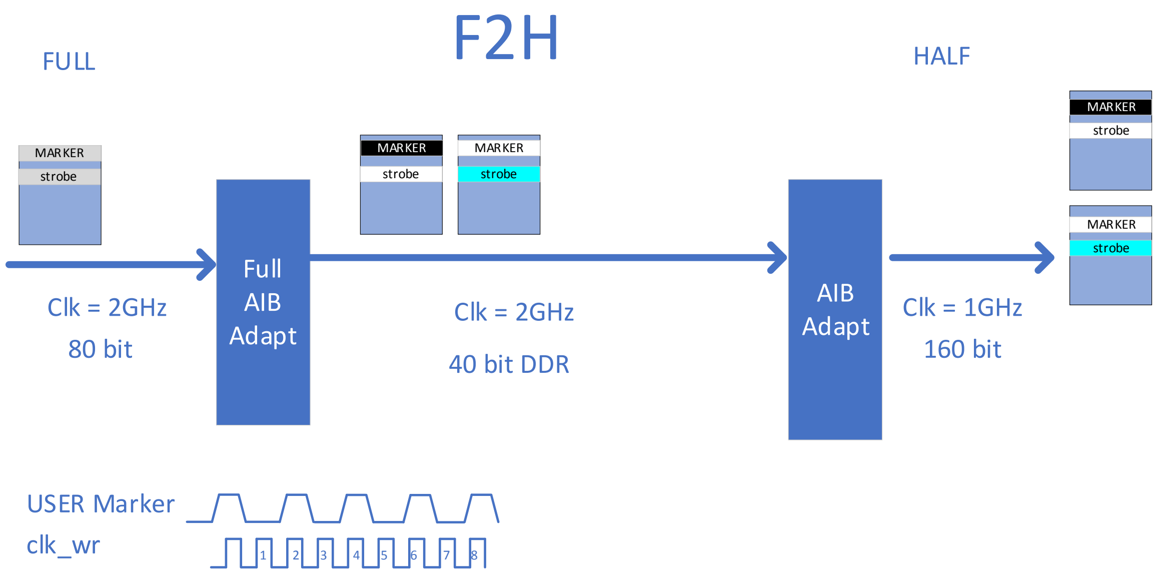


Figure 3 Asymmetric F2H

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.

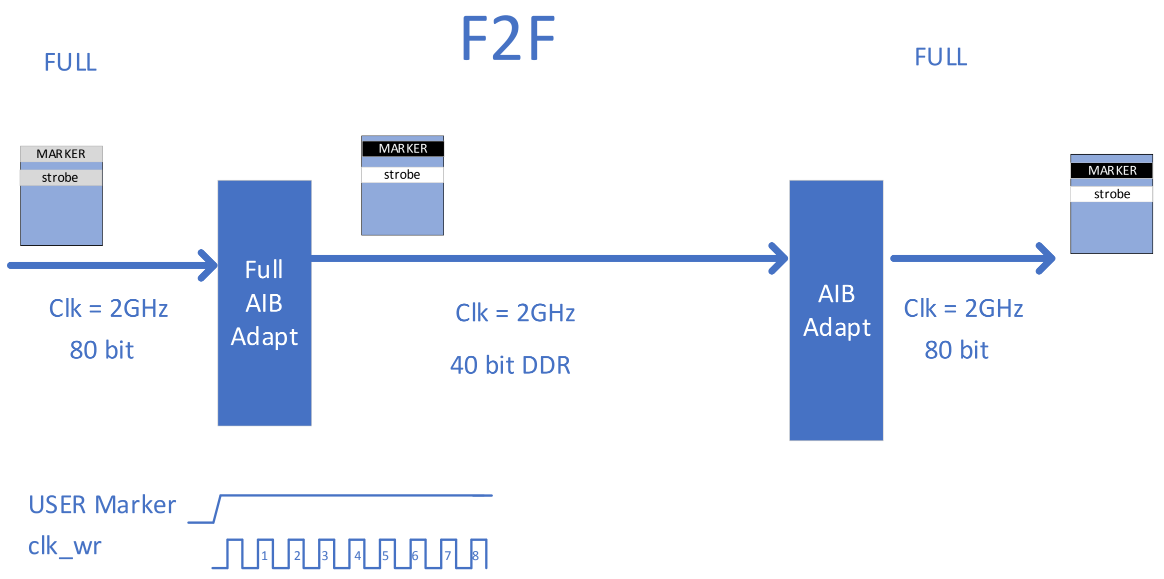


Figure 4 Asymmetric F2F

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.

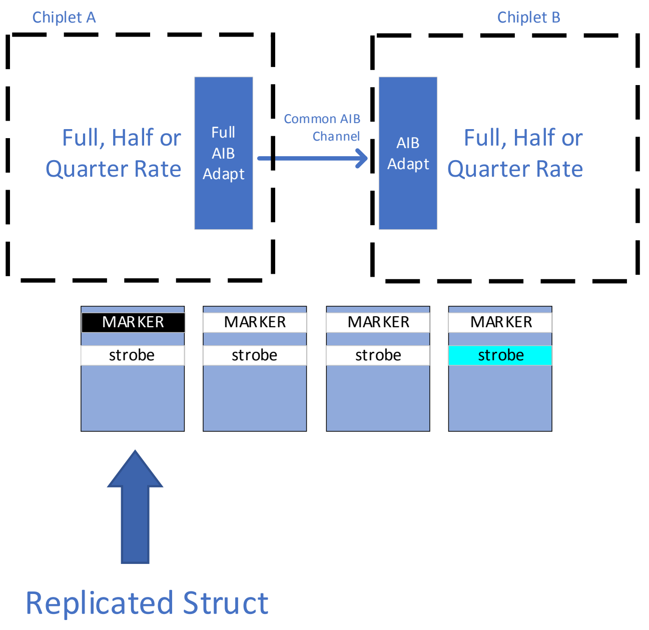
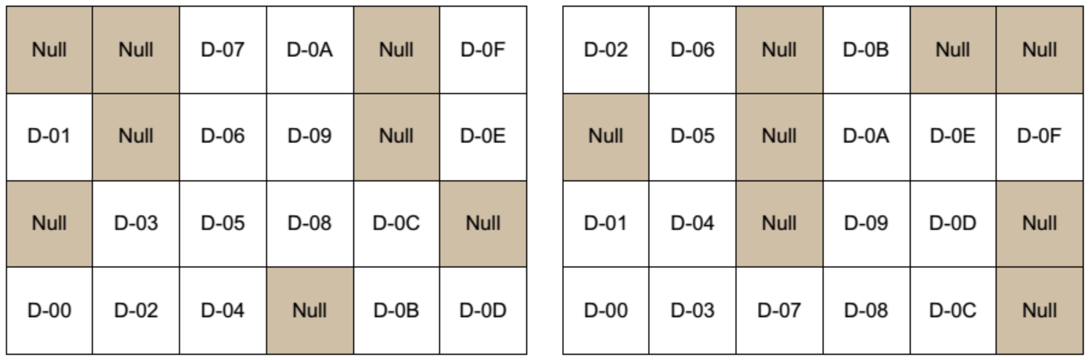


Figure 5 Asymmetric Replicated Struct

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).

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 and combining them into a single beat on a double wide bus with size of 2X. For this reason, TLAST is not supported as the Logic Link will not generate legal AXI-ST.

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

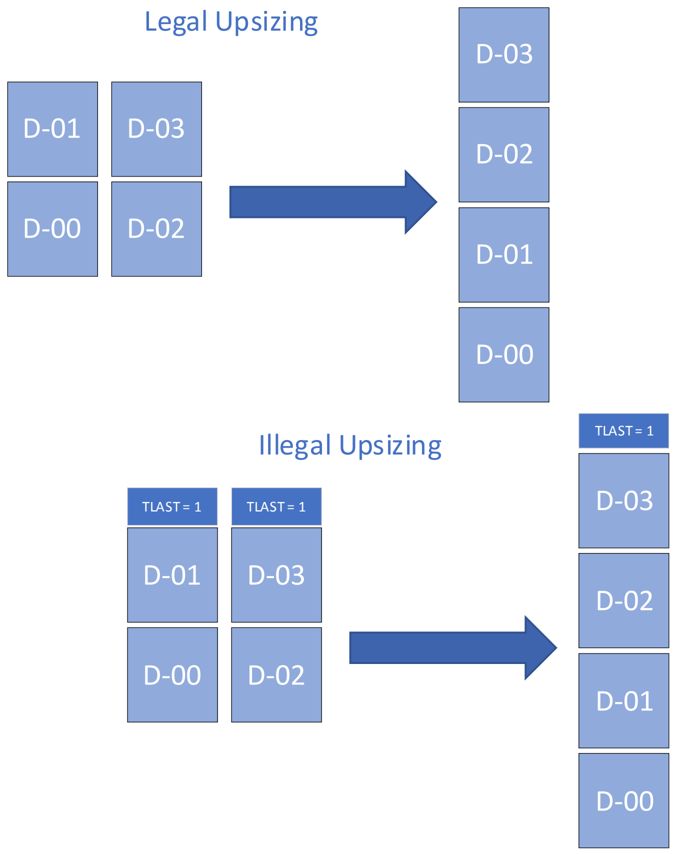


Figure 7 AXI-ST Upsizing Example

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

}
```

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

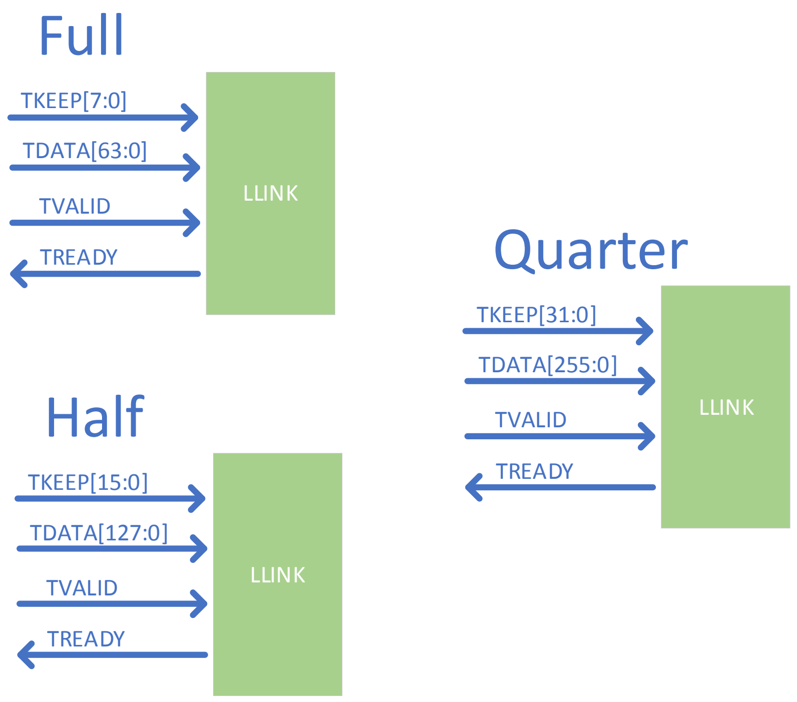


Figure 8 Asymmetric Master Examples

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.

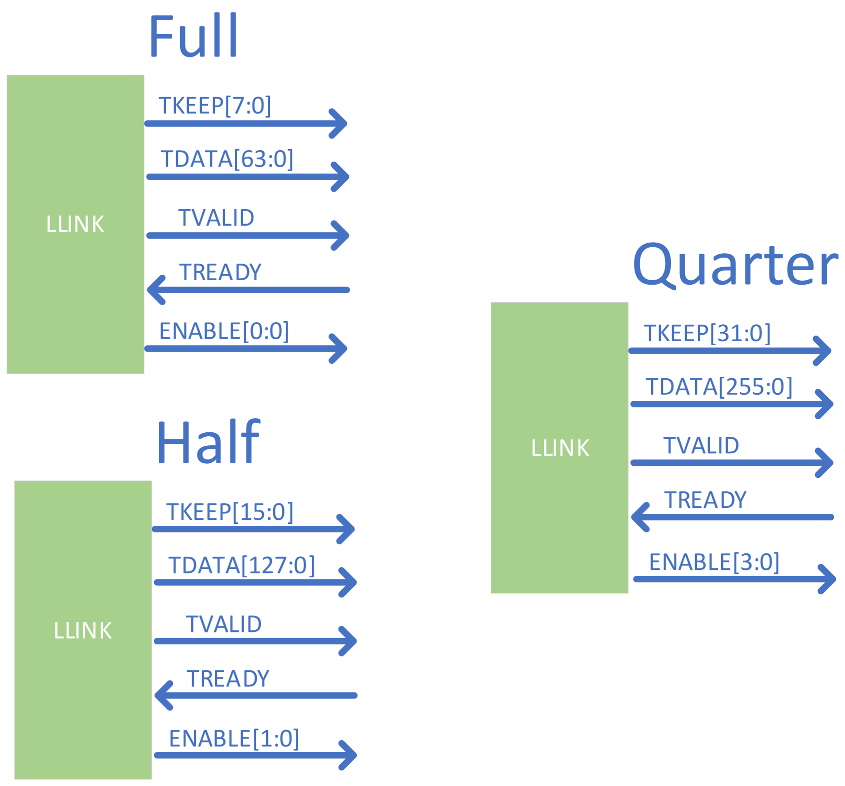


Figure 9 Asymmetric Slave Examples

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.

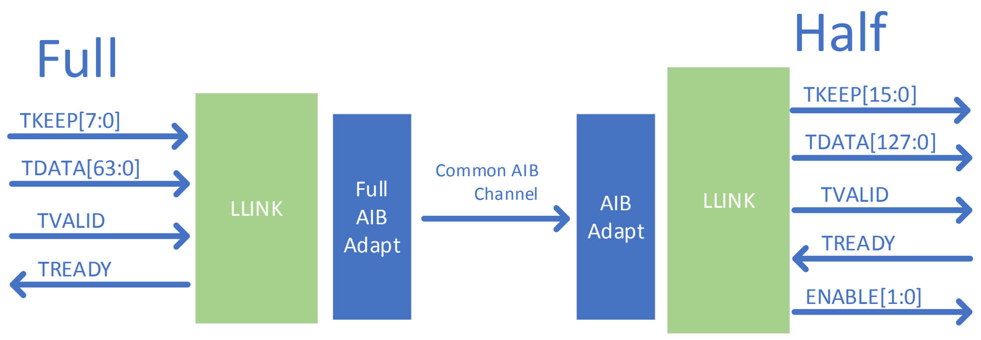


Figure 10 Asymmetric Example

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 current 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 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.

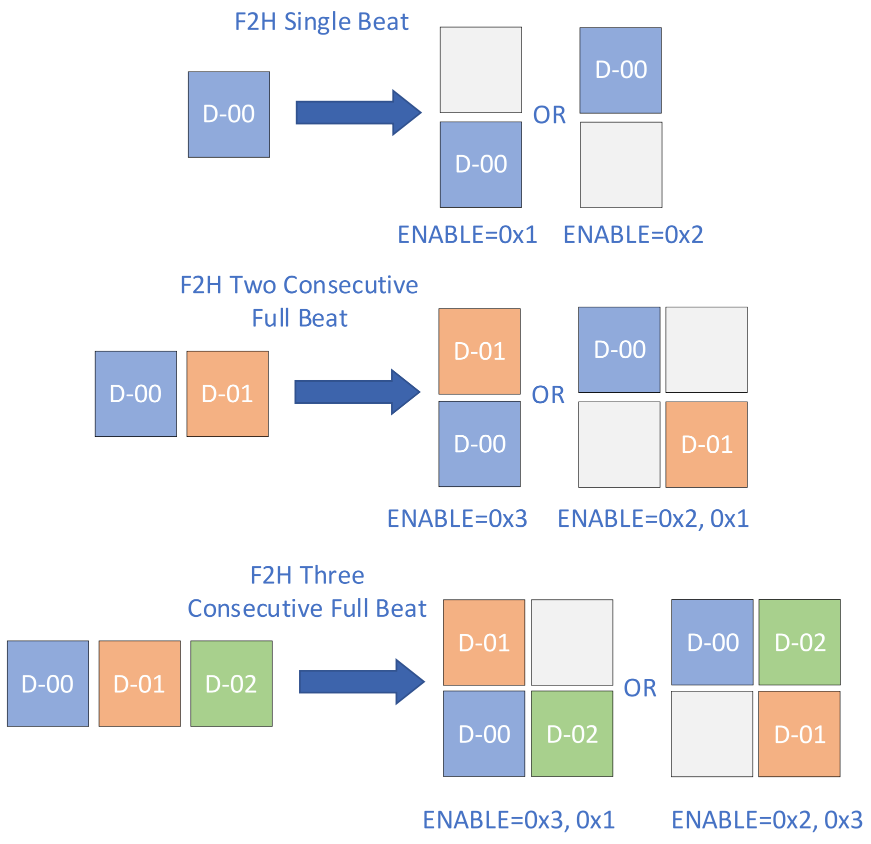


Figure 11 Asymmetric Receive Examples

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

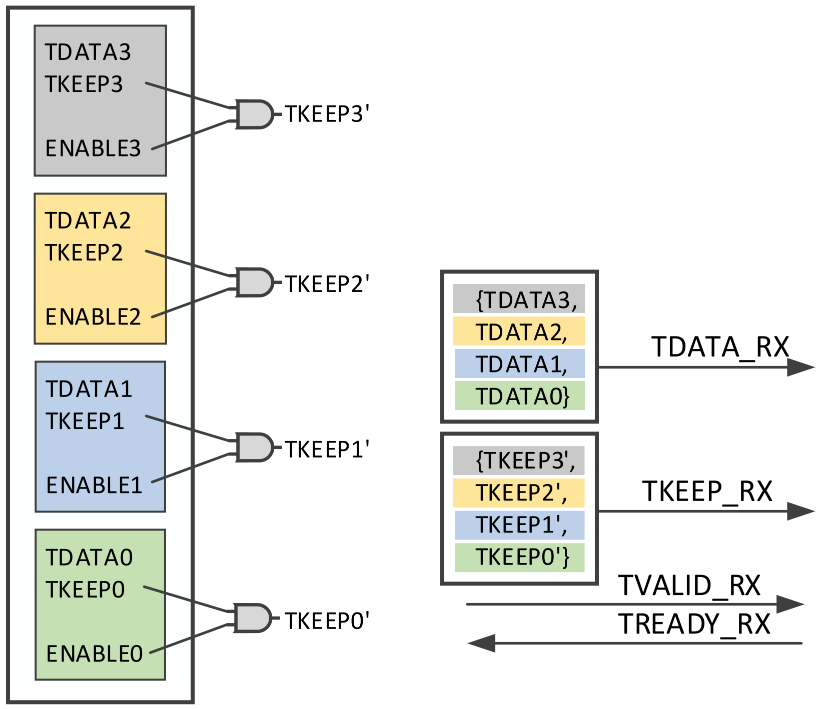


Figure 12 Asymmetric Example

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.

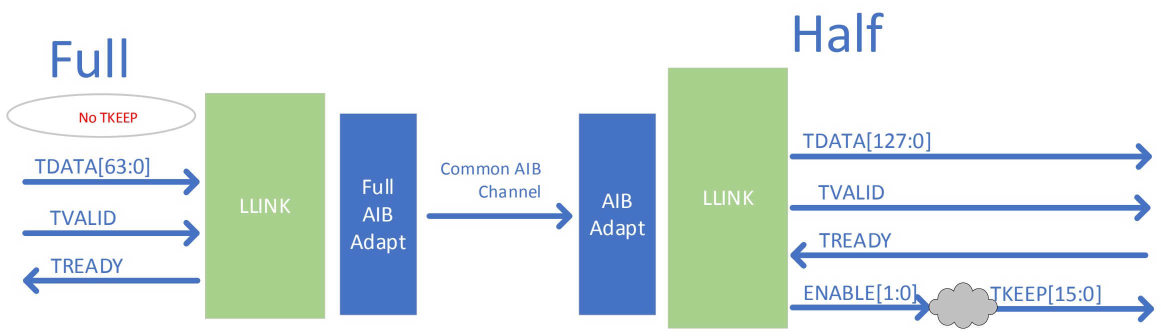


Figure 13 Asymmetric Example

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.

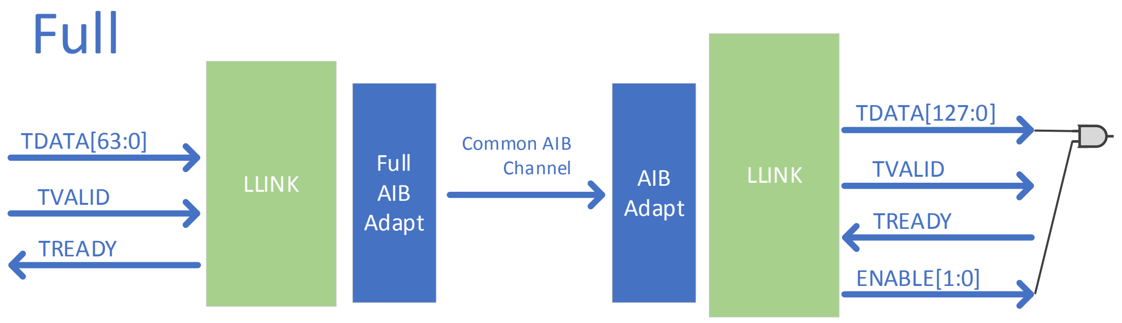


Figure 14 Asymmetric Example

5.4.1.4 *Strobes and Markers*

Another non-obvious side effect of the asymmetric interoperability is the Strobe and Markers.

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.

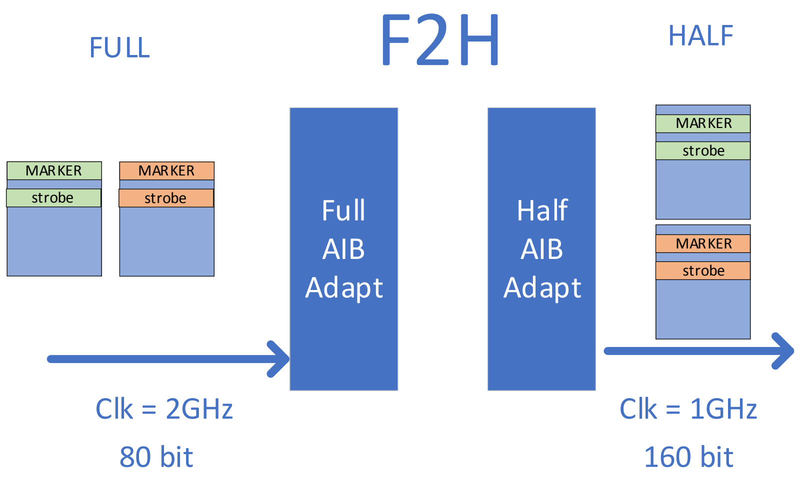


Figure 15 Asymmetric Strobe and Marker Example

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.

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.

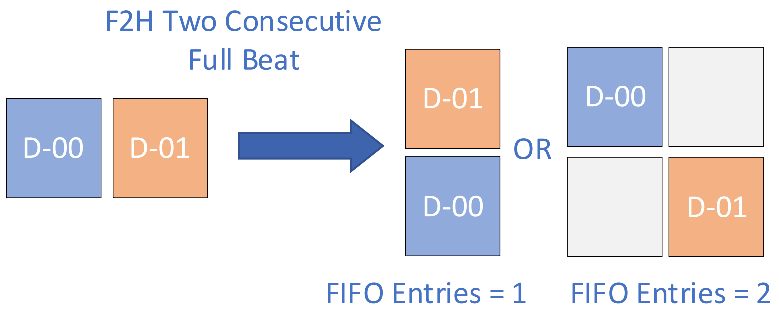


Figure 16 Asymmetric Credit Example

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.

5.4.1.5 *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.

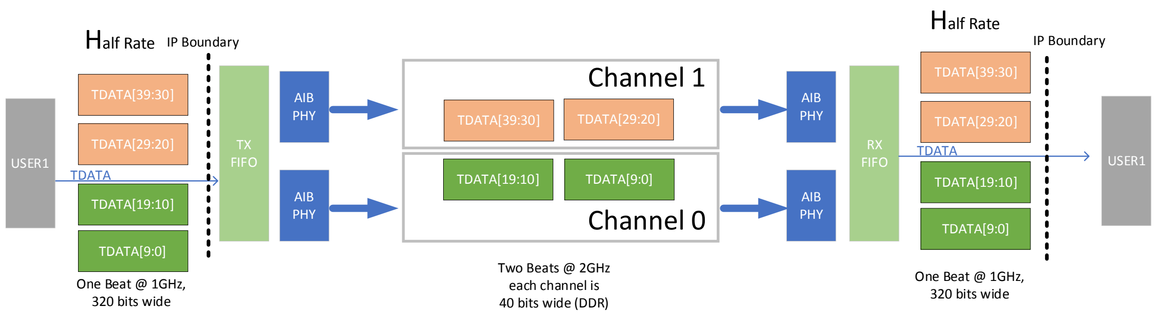


Figure 17 Asymmetric Multi Channel 1

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.

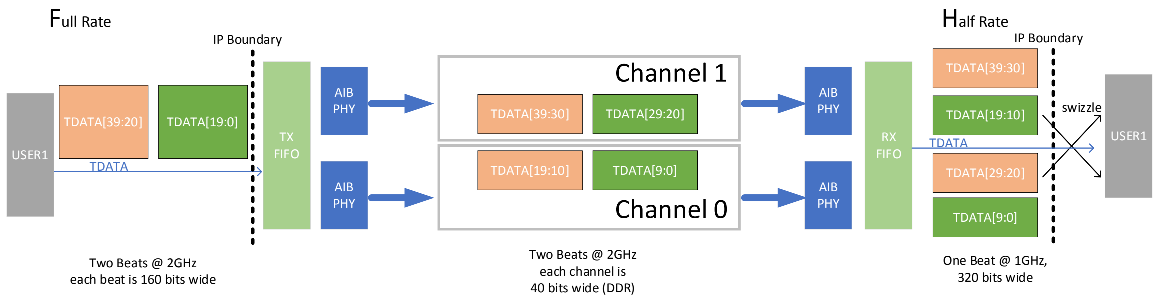


Figure 18 Asymmetric Multi Channel 2

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.

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.

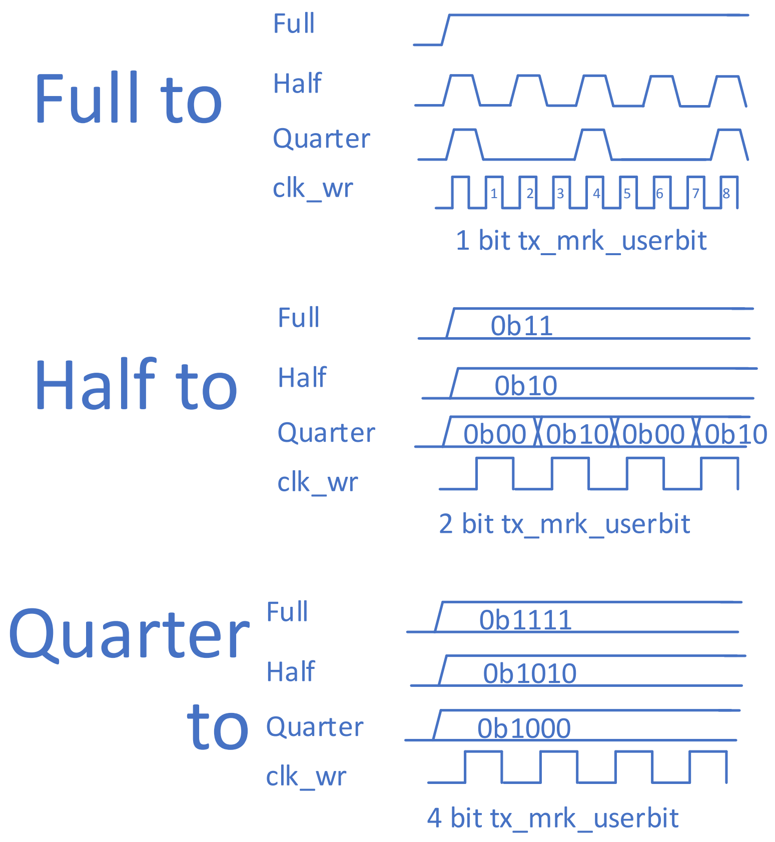


Figure 19 Asymmetric Marker Insertion

5.4.1.6 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

6. AXI-MM

6.1 Fixed Allocation

6.1.1 Theory

This is the simplest of AXI-MM and the most full 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.

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

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 duplex 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 increases latency in the AXI transactions.

However, the effect 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.

From the USER perspective, the AXI interface behaves exact as it does 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 for 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 recommended) will automatically use all available AIB data on the line 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 recommended) will automatically use all available AIB data on the line for each packet.
PACKETIZATION_PACKING_EN	True, False	If set to true, advance “packing” options are enabled which produce more efficient packetization.

Table 9 Packetization Options

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

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

TBD

7.2 Tiered Logic Link

TBD