

XAUI BFM User Guide

1.0 Overview.....	2
2.0 TX Path Functional Description.....	3
3.0 RX Path Functional Description.....	5
4.0 Deep Loopback.....	6
5.0 Top Level Ports.....	7

List of Tables

2.1	TX Path Source File Descriptions	3
3.1	RX Path Source File Description	5
3.2	RX Path Protocol Checks	6
5.1	TX Path Packet interface to testbench or loopback	7
5.2	RX Path Packet interface from testbench or loopback	8
5.3	TX Path line interface to DUT (serial, 10bit TBI, 20bit)	8
5.4	RX Path line interface from DUT (serial, 10bit TBI, 20bit)	9
5.5	Control Inputs for TX Path	10
5.6	Control Inputs for RX Path	13
5.7	Control Inputs for RX and TX Paths	13

1.0 Overview

The XAUI BFM is written in Verilog and is designed to be portable to any verification environment that is compatible with Verilog. The interfaces to the BFM are intended to also maximize portability. The testbench interface to the XAUI BFM is a simple clocked data interface and configuration signals. The DUT interface is 4 data lanes that are either serial, 10 bits wide (TBI) or twenty bits wide. This allows the XAUI BFM to be used in applications including the SERDES model as well as those that do not.

The XAUI BFM consists of a transmit path (TX Path) that drives the line interface of the DUT and a receive path (RX Path) that monitors the line interface from the DUT. In cases where the DUT has more than one XAUI interface, the XAUI BFM may be instantiated multiple times.

The TX path receives packet payload data from the testbench over a clocked data interface and drives these packets into the DUT over the line interface. Additionally, the TX path performs idle mapping to properly hold the bus in an idle state between packets. During both idle periods as well as during packet transmission, errors can be inserted. The presence, frequency, and type of errors are controlled through testbench parameters. Finally, the TX Path has a DUT state machine monitor for the DUT sync state machine, the DUT alignment state machine, the DUT receive state machine, and the DUT XGMII. In this way, the exact response of the DUT to the presence of errors is checked in a cycle by cycle manner. Functional coverage of these state machines is collected to ensure the testing is exhaustive.

The RX path monitors the line interface from the DUT and passes packet data on to the testbench and terminates the idle columns. Extensive checking is performed to ensure no violations of the protocol occur. Both the packet payload encapsulation and the idle columns in the inter-packet gap are checked.

Any errors detected by the RX and TX paths are reported through a single error reporting task that can be customized to best suit the target verification environment.

A deep BFM loopback can be used if line side traffic generators and checkers are not desired. The data that would have gone to the testbench from the RX path is queued and provided to the BFM TX path. This loopback provides the desired convenience of bypassing the line side traffic generation and checking without any loss of link protocol checking.

Finally, XGXS support is provided to allow integration with a legacy XGMII testbench interface.

2.0 TX Path Functional Description

The functionality of the TX path is partitioned into several parts. The Verilog source file name corresponds to the module name (see Table 1.1 TX Path Source File Description). The master control of the XAUI link is performed by *xau_i_txsm*. The monitoring and checking of the DUT alignment, receive, and XGMII state machines is done by *check_4889*. The alternate XGXS interface is provided by *xgxs_tx*. Then, per lane the 8b10b serdes *encoder* is used along with the DUT sync state machine checker *check_487*.

TX Path Source File	Description
<i>xau_i_txsm.v</i>	Master control of BFM TX path
<i>check_4889.v</i>	Checker for DUT alignment, receive, and XGMII
<i>xau_i_4889.v</i>	Used by <i>check_4889</i> to model state machines.
<i>xau_itxdefs.h</i>	File of `defines for 8b10b control codes.
<i>xgxs_tx.v</i>	Gasket to alternate transmit XGMII interface
<i>xau_ibfmxlane.v</i>	Wrapper around per-lane modules
<i>check_487.v</i>	Checker for DUT sync state machine
<i>xau_i_487.v</i>	Used by <i>check_487</i> to model state machine
<i>encoder.v</i>	8b10b encoder (see Serdes8b10b documentation)
<i>stim.v</i>	Queue to provide constant delay to model DUT ingress path.
<i>stimv2.v</i>	Queue to provide variable delay to model DUT ingress path.

Table 2.1 TX Path Source File Descriptions

The higher level testbench will interface to *xau_i_txsm*. One feature the higher level testbench will typically use is the different modes of operation controlled with the *state_garbage*, *state_sync*, and *state_normal* inputs. When *state_garbage* is asserted, garbage is driven to the DUT to mimic periods of bus instability at startup or cable pull. The *garbage_type* and *garbage_vector0-3* inputs allow more precise control of the bad line state. The BFM does a check before each code is selected to ensure that no sync characters could be detected by the DUT during *state_garbage*.

In *state_sync* the sync character is sent to the DUT with sufficient frequency to allow sync to occur on a lane by lane basis. The BFM input *sync_type* specifies if a legal spacing of alignment characters will be driven to the DUT during *state_sync*. If an illegal spacing of alignment characters is selected, then the DUT may take a long time to achieve alignment after sync is achieved. Note: once *state_normal* is asserted, alignment will occur quickly because valid spacings of the alignment character will be provided.

In *state_normal* the BFM will nominally transmit a packet when indicated by the testbench and fill the gap between packets with idles. There are many input parameters that control how the BFM will select each idle character. The input parameter *illegal* will override other input parameters and cause a legal sequence of idles to be generated. In this case, if *rf_weight* is greater than zero, then remote-fault indications follow each alignment character. If *lf_weight* is greater than zero, then local-fault indications follow each alignment character. Alignment characters will occur with the frequency indicated in the IEEE spec. The remaining idle mapping is done randomly between the |R| and |K| columns in the ratios indicated by *r_weight* and *a_weight*.

In *state_normal* with *illegal* deasserted the idle mapping is done differently with the intention of more robustly exercising the DUT without causing packet errors. First, a check is done to see if an alignment character should be sent. *xau_i_txsm* will not violate the alignment character minimum spacing requirements. If the minimum alignment character spacing has passed a random number is compared to *a_weight* to determine if an alignment character is sent. If no alignment character was selected, *r_weight*, *rf_weight*, *lf_weight*, and *qrsv_weight* are checked in that order to see which idle to randomly select.

Packet transmission follows the specified sequence of codes with a minimum gap specified by the input parameter *min_ipg*. If the *in_error* input is high during packet transmission the testbench is indicating this packet must be terminated with an error and an error column will be sent. Otherwise, the packet is terminated normally.

One method of error insertion is controlled with the input parameter *epn_constant_error*. When nonzero, this parameter has the percent likelihood that each column will contain an error. If an error is to be injected onto a lane, then it is randomly selected from disparity error, code error, or unexpected k-code. These errors can occur mid-packet and therefore can cause packet loss.

The other method of error insertion only affects idles and therefore at an infrequent rate that will not cause packet loss. *error_probability_num* indicates the frequency of the random errors and typically would be set to a very low frequency. Disparity errors and code errors are chosen by *error_probability_num*.

In this way, *xau_i_txsm* selects the column to be transmitted. Each code of the column is passed along to each lane's 8b10b encoder in *encoder* (see *serdes_8b10b* documentation).

The lane specific modules are encapsulated in *xauibfmxlane* and instantiated 4 times. These modules are *encoder* and *check_487*. *check_487* uses *stimv.v* to delay the outputs of the *xau_i_txsm* so they coincide in time with the processing of the DUT sync state machine. To calculate this needed delay, the testbench inserted lane skew is added to the

latency added by the DUT to the sync state machine. *check_487* provides this delayed stimulus to *xau_i_487* which implements a testbench version of the IEEE sync state machine. *check_487* probes into the DUT to observe its sync state machine and compares the two. Depending on the DUT implementation there may be periods of uncertainty where checking cannot be reliably performed, but nominally, checking is performed. Any mismatch is an error.

Similarly, *check_4889* uses *xau_i_4889*, *stimv*, and *stimv2* to model and check the alignment state machine, receive state machine, and XGMII bus behavior.

xgxs_tx provides an XGMII transmit interface to interface to legacy testbench environments. When connected via the XGXS interface, the *xau_i_txsm* module would be bypassed and *xgxs_tx* performs the idle mapping.

3.0 RX Path Functional Description

The XAUI RX monitors the 4 lane output of the testbench to extract packets and check protocol violations. The Verilog source file name corresponds to the module name (see Table 1.2 RX Path Source File Description).

RX Path Source File	Description
<i>xau_i_rxsm.v</i>	Extract payload, check protocol, forward packets to testbench
<i>xau_i_rxalign.v</i>	Align 4 lanes of characters and pass along columns.
<i>xauibfmrxlane.v</i>	Wrapper around per-lane modules
<i>xau_i_fifo.v</i>	Per-lane FIFO to assist in lane alignment.
<i>xau_i_rxcodes.v</i>	Map ordered sets
<i>decoder.v</i>	8b10b decoder (see Serdes8b10b documentation)
<i>xau_i_rxdefs1.h</i>	`defines of bit positions of vector from <i>xau_i_rxalign</i>
<i>xau_i_rxdefs2.h</i>	`defines of bit positions of vector from <i>xau_i_rxcodes</i>
<i>xgxs_rx.v</i>	Alternate XGMII receive interface for legacy testbenches

Table 3.1 RX Path Source File Description

The dataflow starts with the 8b10b *decoder* (see *serdes_8b10b* documentation) which passes the character aligned 8b10b codes to *xau_i_rxcodes*. This module builds a vector indicating the ordered set which then enters a FIFO prior to lane alignment: *xau_i_fifo*. All these per-lane modules are encapsulated in *xauibfmrxlane* and instantiated 4 times.

The output of the 4 lanes is aligned in *xau_i_rxalign* where protocol checking occurs and a vector indicating the column is passed along. *xau_i_rxsm* which performs further protocol checking and packet extraction. Alternatively, *xgxs_rx* can be used to provide an XGMII interface for compatibility to legacy testbenches.

A list of the check that are performed on the RX Path can be found in Table 1.3 RX Path Protocol Checks.

RX Path Protocol Check	Source File
All disparity and code errors	<i>decoder.v</i>
Unknown 8b10b characters detected as invalid	<i>xau_i_rxcodes.v</i>
All alignment errors including excessive skew	<i>xau_i_rxalign.v</i>
Illegal columns flagged as errors	<i>xau_i_rxalign.v</i>
Minimum A spacing violation check	<i>xau_i_rxsm.v</i>
Maximum A spacing violation check	<i>xau_i_rxsm.v</i>
Check that all columns during IPG are correct	<i>xau_i_rxsm.v</i>
Check that all RF and LF follow an A	<i>xau_i_rxsm.v</i>
Invalid preamble in S column detected	<i>xau_i_rxsm.v</i>
Invalid preamble in second preamble column detected.	<i>xau_i_rxsm.v</i>
Invalid column mid-packet detected.	<i>xau_i_rxsm.v</i>
Termination of packet correct check	<i>xau_i_rxsm.v</i>
Missing A after eop when max A spacing exceeded.	<i>xau_i_rxsm.v</i>
Unexpected A after eop when prior eop had A	<i>xau_i_rxsm.v</i>
Incorrect column immediately following T detected	<i>xau_i_rxsm.v</i>
Second idle after T must be R or RF check	<i>xau_i_rxsm.v</i>
If LF or RF indicated after EOP, 3 rd idle must be R check	<i>xau_i_rxsm.v</i>
Loss of Sync mid packet detected as an error	<i>xau_i_rxsm.v</i>

Table 3.2 RX Path Protocol Checks

4.0 Deep Loopback

The module *xau_i_loopback* can be used in place of the external testbench packet generator and checker to pass received packet traffic back to the DUT through the TX Path.

The RX Path will have the same protocol checking enabled in both loopback and non-loopback applications. The TX Path error injection and DUT state machine checking will also be available in both loopback and non-loopback applications.

The loopback module has a small FIFO to give the TX Path time to transmit the start column and preamble while payload is queued from the RX Path.

5.0 Top Level Ports

The XAUI BFM top level is called *xauibfm*. *xauibfm* fully encapsulates the XAUI BFM with the exception of *xau_i_loopback* which must be instantiated outside *xauibfm* when loopback is required.

xauibfm interfaces directly to the DUT line interfaces and it interfaces directly to the testbench. Additionally, it optionally monitors the DUT internal state.

The following tables list the ports of *xauibfm* and provide usage guidance.

TX Path Packet Interface		
Port Type	Port Name	Port Usage Description
input [3:0]	txvalid	All zero indicates idle column, all one indicates a data column, the last column of a data transfer may have some number of lower bits be one and the upper bits be zero. A data transfer must be contiguous from start to end without any idle columns. Hold to all zero when <i>xgxs_tx_en</i> is one.
input	txeop	Must coincide with the column containing the /T/ character.
input	in_error	When <i>in_error</i> is one for the first column of the packet a /E/ character is inserted into the preamble.
input [31:0]	txdword	The 4 data Bytes. <i>txdword</i> [7:0] corresponds to <i>txvalid</i> [0].
output	txpop	Indicates the BFM consumed <i>txdword</i> . <i>txdword</i> should not advance unless <i>txpop</i> is one.
output	txready	Indicates the BFM is ready for a data transfer. <i>Txvalid</i> should remain all zero until <i>txready</i> becomes one.
input	tx_column_clk	The BFM acts on these signals near the posedge, so testbench should act on them near the negedge.
input	xgxs_tx_en	One indicates <i>xgxs</i> interface is used, Zero indicates <i>txvalid</i> interface used.

input [31:0]	xgxs_txd	xgmii data bus
input [3:0]	xgxs_rxc	xgmii control bus

Table 5.1 TX Path Packet interface to testbench or loopback

RX Path Packet Interface		
Port Type	Port Name	Port Usage Description
output [3:0]	out_valid	All zero indicates idle or error column, all one indicates a data column, the last column of a data transfer may have some number of lower bits be one and the upper bits be zero. A data transfer must be contiguous from start to end without any idle columns.
output	out_eop	Coincident with the column containing the /T/ code or first column following payload where /T/ should be.
output	out_err	When indicates erred transmission. Either /E/ in payload or incorrect termination.
output	out_push	out_push = out_eop (out_valid);
output[31:0]	rx dword	The 4 data Bytes. rx dword[7:0] corresponds to rx valid [0].
output	rx_column_clk	The BFM acts on these signals near the posedge, so testbench should act on them near the negedge.
output[31:0]	xgxs_rxd	xgmii data bus
output[3:0]	xgxs_rxc	xgmii control bus
output	xgxs_rxclk	xgmii clock

Table 5.2 RX Path Packet interface from testbench or loopback

TX Path Line Interface		
Port Type	Port Name	Port Usage Description
output [3:0]	txserial_d	serial data output transitions on posedge txserial_clk
input	txserial_clk	txserial_clk must have exactly 10 clock cycles for each cycle of tx_column_clk.
output [3:0]	txclk	txclk = {4{txserial_clk}};

output [19:0]	tx_tbid20_0	little endian twenty bit interface. valid only with tbimd20 asserted
output [19:0]	tx_tbid20_1	little endian twenty bit interface. valid only with tbimd20 asserted
output [19:0]	tx_tbid20_2	little endian twenty bit interface. valid only with tbimd20 asserted
output [19:0]	tx_tbid20_3	little endian twenty bit interface. valid only with tbimd20 asserted
output [3:0]	tx_tbic20	twenty bit interface clock with data transitions on posedge. Every posedge of tx_tbic coincides with an edge on txtbic20.
output [9:0]	tx_tbid0	little endian TBI data. valid only with tbimd asserted
output [9:0]	tx_tbid1	little endian TBI data. valid only with tbimd asserted
output [9:0]	tx_tbid2	little endian TBI data. valid only with tbimd asserted
output [9:0]	tx_tbid3	little endian TBI data. valid only with tbimd asserted
output [3:0]	tx_tbic	TBI clock with data transitions on posedge. tx_tbic = !tx_column_clk && (tbimd tbimd20);

Table 5.3 TX Path line interface to DUT (serial, 10bit TBI, 20bit)

RX Path Line Interface		
Port Type	Port Name	Port Usage Description
input [3:0]	rxserial_d	serial data input must transition on posedge
input [3:0]	rxserial_clk	serial data clock
input [19:0]	rx_tbid20_0	little endian twenty bit interface. must transition on posedge.
input [19:0]	rx_tbid20_1	little endian twenty bit interface. must transition on posedge.
input [19:0]	rx_tbid20_2	little endian twenty bit interface. must transition on posedge.
input [19:0]	rx_tbid20_3	little endian twenty bit interface. must transition on posedge.
input [9:0]	rx_tbid0	little endian TBI data. must transition on posedge.
input [9:0]	rx_tbid1	little endian TBI data. must transition on posedge.
input [9:0]	rx_tbid2	little endian TBI data. must transition on posedge.
input [9:0]	rx_tbid3	little endian TBI data. must transition on posedge.
input [3:0]	rx_tbic	clock for TBI (must be active in both tbimd and tbimd20)

input [3:0]	rx_tbic20	clock for twenty bit interface. (if tbimd20 active, rx_tbic20 must have all edges coincide with rx_tbic edges.)
-------------	-----------	---

Table 5.4 RX Path line interface from DUT (serial, 10bit TBI, 20bit)

Control Inputs for TX Path		
Port Type	Port Name	Port Usage Description
input	tx_statg	When asserted holds xau_i_txsm in garbage state.
input	tx_states	When asserted and tx_statg deasserted, holds xau_i_txsm in sync state.
input	tx_staten	When asserted and tx_statg && tx_states deasserted, holds xau_i_txsm in normal state.
input [2:0]	tx_gt	garbage type when in garbage state. 1: Invalid codes sent. Per lane controlled by tx_gv. 2: Random valid codes are sent except K 3: Random valid codes are sent except A
input [2:0]	tx_gv0	garbage vector selects invalid codes per lane: 1: send random invalid codes. 2: hold lane low 3: hold lane high
input [2:0]	tx_gv1	garbage vector selects invalid codes per lane: 1: send random invalid codes. 2: hold lane low 3: hold lane high
input [2:0]	tx_gv2	garbage vector selects invalid codes per lane: 1: send random invalid codes. 2: hold lane low 3: hold lane high
input [2:0]	tx_gv3	garbage vector selects invalid codes per lane: 1: send random invalid codes. 2: hold lane low 3: hold lane high

input [2:0]	tx_sk	<p>sync type determines the columns sent in sync state.</p> <p>0: easy sync. 8 K followed by 15 random K or R followed by A .</p> <p>1: difficult sync. violate A spacing with random periods of possibly back to back A intermixed with periods without any A .</p>
input	tx_ilegal	<p>A one indicates to xaui_txsm to generate the idle sequence to follow the IEEE specification. See the following 5 *wt inputs.</p>
input [31:0]	tx_awt	<p>This value must be between 0 and 1000 (realistically 0 to 70) Nominally, it is the probability that a randomly generated idle is an A where 1000 indicates all A and 0 indicates no A . When tx_ilegal == 1, tx_awt has no effect on the transmission of A columns, but it has a side effect on the transmission of other idles. The probability of sending a K is approximated by $1000 - (tx_awt + tx_rwt)$ for small values of tx_awt.</p>
input [31:0]	tx_rwt	<p>This value must be between 0 and 1000. Nominally, it is the probability that a randomly generated idle is an R where 1000 indicates all R and 0 indicates no R . The priority selection if Idles selects A columns first, then R so if tx_awt is non-zero then it is impossible to achieve all R s even if tx_rwt == 1000.</p>
input [31:0]	tx_rfwt	<p>This value must be between 0 and 1000. Nominally, it is the probability that a randomly generated idle is an RF where 1000 indicates all RF and 0 indicates no RF . When tx_ilegal == 1, tx_rfwt has a simpler function. If it is non-zero then every A is followed by a RF as per spec.</p>
input [31:0]	tx_lfwt	<p>This value must be between 0 and 1000. Nominally, it is the probability that a randomly generated idle is an LF where 1000 indicates all LF and 0 indicates no LF . When tx_ilegal == 1, tx_lfwt has a simpler function. If it is non-zero then every A is followed by a LF as per spec.</p>

input [31:0]	tx_qrsvwt	This value must be between 0 and 1000. Nominally, it is the probability that a randomly generated idle is an invalid Q where 1000 indicates all invalid Q and 0 indicates no invalid Q . When tx_illegal == 1, tx_qrsvwt has no effect.
input [31:0]	epn_constant_error	This value must be between 0 and 1000. It is the probability that a column will have a bad character where 0 indicates no errors and 1000 indicates all errors.
input [31:0]	error_probability_number	Each lane in an idle column is subjected to this check twice, once for disparity error and once for code error: \$urandom_range (error_probability_num==1) If the check is met, then that lane gets that error.
input [31:0]	txmin_ipg	Minimum number of idle columns between packets.
input [6:0]	txbit_delay0	Number of bits to delay the bitstream for lane0. For example, when TBI mode is enabled if the bit delay is 3, then the 10 bit codes will be offset by 3 bits. Must be in the range 1-129, but to avoid deskew problems in the DUT, the range 1-80 is advised.
input [6:0]	txbit_delay1	Number of bits to delay the bitstream for lane1. For example, when TBI mode is enabled if the bit delay is 3, then the 10 bit codes will be offset by 3 bits. Must be in the range 1-129, but to avoid deskew problems in the DUT, the range 1-80 is advised.
input [6:0]	txbit_delay2	Number of bits to delay the bitstream for lane2. For example, when TBI mode is enabled if the bit delay is 3, then the 10 bit codes will be offset by 3 bits. Must be in the range 1-129, but to avoid deskew problems in the DUT, the range 1-80 is advised.
input [6:0]	txbit_delay3	Number of bits to delay the bitstream for lane3. For example, when TBI mode is enabled if the bit delay is 3, then the 10 bit codes will be offset by 3 bits. Must be in the range 1-129, but to avoid deskew problems in the DUT, the range 1-80 is advised.

input	enable4889	When set to one, the check4889 module is enabled to report errors.
input	enable487	When set to one, the check487 module is enabled to report errors.
input	tx_reset	Included only for future use. tie to zero.

Table 5.5 Control Inputs for TX Path

Control Inputs RX Path		
Port Type	Port Name	Port Usage Description
input [3:0]	rxresync	Included only for future use. tie to zero.
input	rx_reset	Included only for future use. tie to zero.

Table 5.6 Control Inputs for RX Path

Control Inputs shared RX and TX Paths		
Port Type	Port Name	Port Usage Description
input	tbimd	Asserted puts BFM in TBI mode. Never assert both tbimd and tbimd20.
input	tbimd20	Asserted puts BFM in twenty bit interface mode. Never assert both tbimd and tbimd20.

Table 5.7 Control Inputs for RX and TX Paths