Avalon-Stream BFM – Quick Reference

For general information see UVVM Essential Mechanisms located in uvvm_vvc_framework/doc.

BFM

avalon_st_bfm_pkg.vhd

Avalon-Stream Master (see page 2 for Avalon-Stream Slave)

avalon_st_transmit ([channel_value], data_array, msg, clk, avalon_st_if, [scope, [msg_id_panel, [config]]])

Example: avalon_st_transmit(v_channel, v_data_array(0 to v_numBytes-1), "Send v_numBytes bytes on v_channel", clk, avalon_st_if, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
Example: avalon_st_transmit(v_data_array(0 to v_numWords-1), "Send v_numWords words", clk, avalon_st_if, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
Example: avalon_st_transmit((x"01", x"02", x"03", x"04"), "Send 4 bytes", clk, avalon_st_if);

init_avalon_st_if_signals (is_master, channel_width, data_width, data_error_width, empty_width)

Example: avalon_st_if <= init_avalon_st_if_signals(true, avalon_st_if.channel'length, avalon_st_if.data'length, avalon_st_if.data_error'length, avalon_st_if.empty'length);

UVVM™

VHDL 2008 only

Avalon-Stream BFM – Quick Reference

Avalon-Stream Slave (see page 1 for Avalon-Stream Master)

BFM

avalon_st_bfm_pkg.vhd

avalon_st_receive ([channel_value], data_array, msg, clk, avalon_st_if, [scope, [msg_id_panel, [config, [ext_proc_call]]]])

Example: avalon_st_receive(v_channel, v_rx_data_array, “Receive packet”, clk, avalon_st_if, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
Example: avalon_st_receive(v_rx_data_array, “Receive packet”, clk, avalon_st_if);

avalon_st_expect ([channel_exp], data_exp, msg, clk, avalon_st_if, [alert_level, [scope, [msg_id_panel, [config]]]])

Example: avalon_st_expect(v_channel, v_data_array(0 to v_numBytes-1), "Expect v_numBytes bytes on v_channel", clk, avalon_st_if, ERROR, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
Example: avalon_st_expect(v_data_array(0 to v_numWords-1), "Expect v_numWords words", clk, avalon_st_if, ERROR, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
Example: avalon_st_expect((x”01”, x”02”, x”03”, x”04”), "Expect 4 bytes", clk, avalon_st_if)

init_avalon_st_if_signals (is_master, channel_width, data_width, data_error_width, empty_width)

Example: avalon_st_if <= init_avalon_st_if_signals(false, avalon_st_if.channel'length, avalon_st_if.data'length, avalon_st_if.data_error'length, avalon_st_if.empty'length);

UVVM™

BFM Configuration record '**t_avalon_st_bfm_config**'

Record element	Type	C_AVALON_ST_BFM_CONFIG_DEFAULT
max_wait_cycles	natural	100
max_wait_cycles_severity	t_alert_level	ERROR
clock_period	time	0 ns
clock_period_margin	time	0 ns
clock_margin_severity	t_alert_level	TB_ERROR
setup_time	time	0 ns
hold_time	time	0 ns
symbol_width	natural	8
first_symbol_in_msb	boolean	true
max_channel	natural	0
use_packet_transfer	boolean	true
id_for_bfm	t_msg_id	ID_BFM

Signal record **'t_avalon_st_if'**

Record element	Type
channel	std_logic_vector
data	std_logic_vector
data_error	std_logic_vector
ready	std_logic
valid	std_logic
empty	std_logic_vector
end_of_packet	std_logic
start_of_packet	std_logic

BFM signal parameters

Name	Type	Description
clk	std_logic	The clock signal used to read and write data in/out of the Avalon-Stream BFM.
avalon_st_if	t_avalon_st_if	See table "Signal record 't_avalon_st_if'" above. Note: All supported signals, including `channel` and `data_error` are included in the record type, even when not used or connected to DUT.

For more information on the Avalon-Stream signals, refer to "Avalon® Interface Specifications, Chapter: Avalon Streaming Interfaces", document number MNL-AVABUSREF, available from Intel.

BFM non-signal parameters

Name	Type	Example(s)	Description
channel_value	std_logic_vector	x"01"	Channel number for the data being transferred. The value is limited by max_channel in the BFM config.
channel_exp	std_logic_vector	x"01"	Expected channel number for the data being transferred. The value is limited by max_channel in the BFM config.
data_array	t_slv_array	(x"D0D1", x"D2D3")	An array of SLVs containing the data to be sent/received. data_array(0) is sent/received first, while data_array(data_array'high) is sent/received last. For clarity, data_array is required to be ascending, for example defined by the test sequencer as follows: `variable v_data_array : t_slv_array(0 to C_MAX_WORDS-1)(C_MAX_WORD_LENGTH-1 downto 0);`
data_exp	t_slv_array	(x"D0D1", x"D2D3")	An array of SLVs containing the data that is expected to be received. The data_array specifications listed above applies for data_exp as well.
alert_level	t_alert_level	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the procedure.
msg	string	"Send packet"	A custom message to be appended in the log/alert.
scope	string	"AVALON_ST_BFM"	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AVALON_ST_BFM". In a verification component typically "AVALON_ST_VVC ".
msg_id_panel	t_msg_id_panel	shared_msg_id_panel	Optional msg_id_panel, controlling verbosity within a specified scope. Defaults to a common message ID panel defined in the UVVM-Util adaptations package.
config	t_avalon_st_bfm_config	C_AVALON_ST_BFM_ CONFIG_DEFAULT	Configuration of BFM behaviour and restrictions. See section 2 for details.

BFM features

The following signals are supported:

Signal	Source	Width	Supported by BFM	Description
associatedClock	Clock	1	Yes	Sample on the rising edge.
associatedReset	Reset	-	No	BFM doesn't control the reset.
channel	Master	1-128	Yes	Channel number for the data being transferred on the current cycle.
data	Master	1-4096	Yes	Data word. It can consist of several symbols.
error	Master	1-256	No	Bit mask to mark errors affecting the data being transferred on the current cycle. The error_descriptor in the BFM config defines the error signal properties.
ready	Slave	1	Yes	Indicates that the slave can accept data. A transfer takes place when both valid and ready are asserted.
valid	Master	1	Yes	This signal qualifies all other master to slave signals. A transfer takes place when both valid and ready are asserted.
empty	Master	1-5	Yes	Number of symbols that are empty during the end_of_packet cycle.
end_of_packet	Master	1	Yes	When '1', it indicates that the data is the last word of the packet.
start_of_packet	Master	1	Yes	When '1', it indicates that the data is the first word of the packet.

BFM details

1 BFM procedure details

Procedure	Description
avalon_st_transmit()	**avalon_st_transmit ([channel_value], data_array, msg, clk, avalon_st_if, [scope, [msg_id_panel, [config]]])** The avalon_st_transmit() procedure transmits a stream/packet on the Avalon interface. The length and data are defined by the "data_array" argument, which is a t_slv_array. data_array(0) is sent first. data_array(data_array'high) is sent last. When the config use_packet_transfer is enabled: During the first word, the BFM asserts the start_of_packet signal. During the last word, the BFM asserts the end_of_packet signal and it sets the number of invalid symbols in the word on the empty signal.
avalon_st_receive()	**avalon_st_receive ([channel_value], data_array, msg, clk, avalon_st_if, [scope, [msg_id_panel, [config, [ext_proc_call]]]])** The avalon_st_receive() procedure receives a stream/packet on the Avalon interface. The received data is stored in the data_array output, which is a t_slv_array. When the config use_packet_transfer is enabled: The signal start_of_packet is expected to be set during the first word. The signal end_of_packet is expected to be set during the last word. Also during this word the empty signal is used to determine the number of invalid symbols.
avalon_st_expect()	**avalon_st_expect ([channel_exp], data_exp, msg, clk, avalon_st_if, [alert_level, [scope, [msg_id_panel, [config]]]])** Calls the avalon_st_receive() procedure, then compares the received data with data_exp and the optional channel with channel_exp.
init_avalon_st_if_signals()	**init_avalon_st_if_signals(is_master, channel_width, data_width, data_error_width, empty_width)** This function initializes the Avalon-Stream interface. All the BFM outputs are set to zeros ('0')

2 BFM Configuration record

Type name: t_avalon_st_bfm_config

Record element	Type	C_AVALON_ST_BFM_CONFIG_DEFAULT	Description
max_wait_cycles	natural	100	Used for setting the maximum cycles to wait before an alert is issued when waiting for ready or valid signals from the DUT.
max_wait_cycles_severity	t_alert_level	ERROR	Severity if max_wait_cycles expires.
clock_period	time	0 ns	Period of the clock signal. Default is 0 ns to detect if not set by user.
clock_period_margin	time	0 ns	Input clock period margin to specified clock_period.
clock_margin_severity	t_alert_level	TB_ERROR	The above margin will have this severity.
setup_time	time	0 ns	Setup time for generated signals. Suggested value is clock_period/4. An alert is reported if setup_time exceed clock_period/2.
hold_time	time	0 ns	Hold time for generated signals. Suggested value is clock_period/4. An alert is reported if hold_time exceed clock_period/2.

symbol_width	natural	8	Number of data bits per symbol.
first_symbol_in_msb	boolean	true	Symbol ordering. When true, first-order symbol is in most significant bits.
max_channel	natural	0	Maximum number of channels that the interface supports.
use_packet_transfer	boolean	true	When true, packet signals are enabled: start_of_packet, end_of_packet & empty.
id_for_bfm	t_msg_id	ID_BFM	The message ID used as a general message ID in the BFM.

3 Additional Documentation

For additional documentation on the Avalon-Stream standard, refer to “Avalon® Interface Specifications, Chapter: Avalon Streaming Interfaces”, document number MNL-AVABUSREF, available from Intel.

4 Compilation

The Avalon-Stream BFM may only be compiled with VHDL 2008. It is dependent on the UVVM Utility Library (UVVM-Util), which is only compatible with VHDL 2008.
See the separate UVVM-Util documentation for more info. After UVVM-Util has been compiled, the avalon_st_bfm_pkg.vhd BFM can be compiled into any desired library.
See UVVM Essential Mechanisms located in uvvm_vvc_framework/doc for information about compile scripts.

4.1 Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see UVVM-Util Quick reference.

5 Local BFM overloads

A good approach for better readability and maintainability is to make simple, local overloads for the BFM procedures in the TB process.
This allows calling the BFM procedures with the key parameters only
e.g.

```
avalon_st_transmit(v_data_array(0 to 1), "msg");
```

rather than

```
avalon_st_transmit(v_data_array(0 to 1), "msg", clk, avalon_st_if, C_SCOPE, shared_msg_id_panel, avalon_st_bfm_config);
```

By defining the local overload as e.g.:

```
 procedure avalon_st_transmit(
   constant data_array : in t_slv_array;
   constant msg        : in string) is
begin
    avalon_st_transmit(data_array,                    -- keep as is
                       msg,                           -- keep as is
                       clk,                           -- Clock signal
                       avalon_st_if,                  -- Signal must be visible in local process scope
                       C_SCOPE,                       -- Just use the default
                       shared_msg_id_panel,           -- Use global, shared msg_id_panel
```

```
                        C_AVALON_ST_BFM_CONFIG_LOCAL);    -- Use locally defined configuration or C_AVALON_ST_BFM_CONFIG_DEFAULT
    end;
```

Using a local overload like this also allows the following – if wanted:

- Set up defaults for constants. May be different for two overloads of the same BFM
- Apply dedicated message_id_panel to allow dedicated verbosity control

IMPORTANT
This is a simplified Bus Functional Model (BFM) for Avalon-Stream. The given BFM complies with the basic Avalon-Stream protocol and thus allows a normal access towards an Avalon-Stream interface. This BFM is not Avalon-Stream protocol checker. For a more advanced BFM please contact Bitvis AS at support@bitvis.no