**Avalon-Stream BFM** –Quick Reference

**BFM**

For general information see UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc.

**NOTE: As of UVVM v3.x, all shared variables have been made protected. This means that any access to shared variables must be done**

**using get- and set-methods. This documentation has not yet been updated with the methods for accessing these variables, but will be very soon.**

Avalon-Stream Master

*avalon\_st\_bfm\_pkg.vhd*

|  |
| --- |
| 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); |

Avalon-Stream Slave

|  |
| --- |
| 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 ); |



Signal record ´**t\_axistream\_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 Configuration record ‘**t\_avalon\_st\_bfm\_config’** Signal record **‘t\_avalon\_st\_if’**

|  |  |  |
| --- | --- | --- |
| 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 | -1 ns |
| clock\_period\_margin | time | 0 ns |
| clock\_margin\_severity | t\_alert\_level | TB\_ERROR |
| setup\_time | time | -1 ns |
| hold\_time | time | -1 ns |
| bfm\_sync | t\_bfm\_sync | SYNC\_ON\_CLOCK\_ONLY |
| match\_strictness | t\_match\_strictness | MATCH\_EXACT |
| 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 |

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  channel\_exp | std\_logic\_vector | x”01” | Channel number for the data being transferred or expected.  The value is limited by max\_channel in the BFM config. |
| data\_array  data\_exp | 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);  For simplicity, the word\_length can only be the size of the configured symbol or the size of the data bus.  variable v\_data\_array : t\_slv\_array(0 to C\_MAX\_WORDS-1)(C\_SYMBOL\_WIDTH-1 downto 0);  variable v\_data\_array : t\_slv\_array(0 to C\_MAX\_WORDS-1)(C\_DATA\_BUS\_LENGTH-1 downto 0); |
| 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 0 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. The signal width in bits is ceil[log2(symbols\_per\_cycle)]. Only required when the data signal carries more than one symbol of data per cycle and has a variable packet length. |
| 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

# 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') |

# 

# 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 | -1 ns | Period of the clock signal. |
| 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 | -1 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 | -1 ns | Hold time for generated signals. Suggested value is clock\_period/4.  An alert is reported if hold\_time exceed clock\_period/2. |
| bfm\_sync | t\_bfm\_sync | SYNC\_ON\_CLOCK\_ONLY | When set to SYNC\_ON\_CLOCK\_ONLY the BFM will enter on the first falling edge, estimate the clock period, synchronise the output signals and exit ¼ clock period after a succeeding rising edge. When set to SYNC\_WITH\_SETUP\_AND\_HOLD the BFM will use the configured setup\_time, hold\_time and clock\_period to synchronise output signals with clock edges. |
| match\_strictness | t\_match\_strictness | MATCH\_EXACT | Matching strictness for std\_logic values in check procedures.  MATCH\_EXACT requires both values to be the same. Note that the expected value  can contain the don’t care operator ‘-‘.  MATCH\_STD allows comparisons between ‘H’ and ‘1’, ‘L’ and ‘0’ and ‘-‘ in both values. |
| 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. |

# 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.

# 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.

## Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see UVVM-Util Quick reference.

# 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

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**