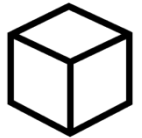


AXI4 BFM – Quick Reference

BFM



axi_bfm_pkg.vhd

axi_write (awid_value, awaddr_value, awlen_value, awsize_value, awburst_value, awlock_value, awcache_value, awprot_value, awqos_value, awregion_value, awuser_value, wdata_value, wstrb_value, wuser_value, buser_value, bresp_value, msg, clk, axi_if, [scope, [msg_id_panel, [config]]])

Example: axi_write(

```
awid_value      => x"01",
awaddr_value    => x"00000004",
awlen_value     => x"01",
awsize_value    => 4,
awburst_value   => INCR,
awlock_value    => NORMAL,
awcache_value   => "0000",
awprot_value    => UNPRIVILEGED_UNSECURE_DATA,
awqos_value     => "0000",
awregion_value  => "0000",
awuser_value    => x"01",
wdata_value     => t_slv_array'(x"12345678", x"33333333"),
wstrb_value     => t_slv_array'(x"F", x"F"),
wuser_value     => t_slv_array'(x"01", x"01"),
buser_value     => v_buser_value,
bresp_value     => v_bresp_value,
msg            => "Writing data to Peripheral 1",
clk            => clk,
axi_if         => axi_if;
```

Suggested usage: axi_write(

```
awaddr_value    => x"00000004",
awlen_value     => x"01",
wdata_value     => t_slv_array'(x"12345678", x"33333333"),
msg            => "Writing data to Peripheral 1";
-- Suggested usage requires local overload (see section 5)
```

axi_read (arid_value, araddr_value, arlen_value, arsize_value, arburst_value, arlock_value, arcache_value, arprot_value, arqos_value, arregion_value, aruser_value, rdata_value, rresp_value, ruser_value, msg, clk, axi_if, [scope, [msg_id_panel, [config, [proc_name]]]])

Example: axi_read(

```
arid_value      => x"01",
araddr_value    => x"00000004",
arlen_value     => x"01",
arsize_value    => 4,
arburst_value   => INCR,
arlock_value    => NORMAL,
arcache_value   => "0000",
arprot_value    => UNPRIVILEGED_UNSECURE_DATA,
arqos_value     => "0000",
arregion_value  => "0000",
aruser_value    => x"01",
rdata_value     => v_rdata_value,
rresp_value     => v_rresp_value,
ruser_value     => v_ruser_value,
msg            => "Read from Peripheral 1",
clk            => clk,
axi_if         => axi_if;
```

Suggested usage: axi_read(

```
araddr_value    => C_ADDR_IO,
arlen_value     => x"01",
rdata_value     => v_data_out,
msg            => "Read from IO";
-- Suggested usage requires local overload (see section 5)
```



UVVM™

axi_check (arid_value, araddr_value, arlen_value, arsize_value, arburst_value, arlock_value, arcache_value, arprot_value, arqos_value, arregion_value, aruser_value, rdata_exp, rresp_exp, ruser_exp, msg, clk, axi_if, [alert_level, [scope, [msg_id_panel, [config]]]])

Example: axi_check(
 arid_value => x"01",
 araddr_value => x"00000004",
 arlen_value => x"01",
 arsize_value => 4,
 arburst_value => INCR,
 arlock_value => NORMAL,
 arcache_value => "0000",
 arprot_value => UNPRIVILEGED_UNSECURE_DATA,
 arqos_value => "0000",
 arregion_value => "0000",
 aruser_value => x"01",
 rdata_exp => t_slv_array(x"12345678", x"33333333"),
 rresp_exp => t_slv_array("00", "00"),
 ruser_exp => t_slv_array(x"00", x"00"),
 msg => "Check data from Peripheral 1",
 clk => clk,
 axi_if => axi_if);

Suggested usage: axi_check(
 araddr_value => C_ADDR_IO,
 arlen_value => "01",
 rdata_exp => t_slv_array(x"12345678", x"33333333"),
 msg => "Checking data from Peripheral 1");
 -- Suggested usage requires local overload (see section 5)

init_axi_if_signals (addr_width, data_width, id_width, user_width)

Example: axi_if <= init_axi_if_signals(addr_width, data_width, id_width, user_width);

BFM Configuration record 't_axi_bfm_config'

Record element	Type	C_AXI_BFM_CONFIG_DEFAULT
max_wait_cycles	natural	1000
max_wait_cycles_severity	t_alert_level	TB_FAILURE
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
num_aw_pipe_stages	natural	1
num_w_pipe_stages	natural	1
num_ar_pipe_stages	natural	1
num_r_pipe_stages	natural	1
num_b_pipe_stages	natural	1
id_for_bfm	t_msg_id	ID_BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT
id_for_bfm_poll	t_msg_id	ID_BFM_POLL

BFM non-signal parameters

Name	Type	Example(s)	Description
axid_value	std_logic_vector	x"01"	Identification tag for a read or write transaction
axaddr_value	unsigned	x"125A"	The address of the first transfer in a read or write transaction
axlen_value	unsigned(7 downto 0)	x"01"	The number of data transfers in a read or write transactions
axsize_value	Integer range 1 to 128	4	The number of bytes in each data transfer in a read or write transaction (Must be a power of two)
axburst_value	t_axburst	INCR	Burst type, indicates how address changes between each transfer in a read or write transaction
axlock_value	t_axlock	NORMAL	Provides information about the atomic characteristics of a read or write transaction
axcache_value	std_logic_vector(3 downto 0)	"0000"	Indicates how a read or write transaction is required to progress through a system
axprot_value	t_axprot	UNPRIVILEGED_UNSECURE_DATA	Protection attributes of a read or write transaction. Privilege, security level and access type
axqos_value	std_logic_vector(3 downto 0)	"0000"	Quality of Service identifier for a read or write transaction
axregion_value	std_logic_vector(3 downto 0)	"0000"	Region indicator for a read or write transaction
axuser_value	std_logic_vector	x"01"	User-defined extension for the read or write address channel
wdata_value	t_slv_array	t_slv_array(x"20D3", x"1234")	Array of data values to be written to the addressed registers
wstrb_value	t_slv_array	t_slv_array("1111", "1111")	Array of write strobes, indicates which byte lanes hold valid data. (all '1' means all bytes are updated)
wuser_value	t_slv_array	t_slv_array(x"00", x"01")	Array of user-defined extension for the write data channel
buser_value	std_logic_vector	v_buser_value	Output variable containing the user-defined extension for the write response channel
bresp_value	t_xresp	v_bresp_value	Output variable containing the write response which indicates the status of a write transaction
rdata_value	t_slv_array	v_rdata_value	Output variable containing an array of read data
rresp_value	t_xresp_array	v_rresp_value	Output variable containing an array of read responses which indicates the status of a read transfer
ruser_value	t_slv_array	v_ruser_value	Output variable containing an array of user-defined extensions for the read data channel
rdata_exp	t_slv_array	t_slv_array(x"ABCD", x"1234")	Array of expected read data values. A mismatch results in an alert 'alert_level'
rresp_exp	t_xresp_array	t_xresp_array(OKAY, OKAY)	Array of expected read responses which indicates the status of a read transfer
ruser_exp	t_slv_array	t_slv_array(x"01", x"01")	Array of expected user-defined extensions for the read data channel
alert_level	t_alert_level	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the procedure.
msg	string	"Set state active on peripheral 1"	A custom message to be appended in the log/alert.
scope	string	"AXI BFM"	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "AXI BFM". In a verification component typically "AXI_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_axi_bfm_config	C_AXI_BFM_CONFIG_DEFAULT	Configuration of BFM behaviour and restrictions. See section 2 for details.

BFM signal parameters

Name	Type	Description
clk	std_logic	The clock signal used to read and write data in/out of the AXI4 BFM.
axi_if	t_axi_if	See table "Signal record 'axi_if'"

Note: All signals are active high. See AXI4 documentation for protocol description.

For more information on the AXI4 signals, please see the AXI4 specification.

Signal record 'axi_if'

Record element	Type
write_address_channel	t_axi_write_address_channel
write_data_channel	t_axi_write_data_channel
write_response_channel	t_axi_write_response_channel
read_address_channel	t_axi_read_address_channel
read_data_channel	t_axi_read_data_channel

Record element	Type
write_address_channel	t_axi_write_address_channel
awid	std_logic_vector
awaddr	std_logic_vector
awlen	std_logic_vector(7 downto 0)
awsize	std_logic_vector(2 downto 0)
awburst	std_logic_vector(1 downto 0)
awlock	std_logic
awcache	std_logic_vector(3 downto 0)
awprot	std_logic_vector(2 downto 0)
awqos	std_logic_vector(3 downto 0)
awregion	std_logic_vector(3 downto 0)
awuser	std_logic_vector
awvalid	std_logic
awready	std_logic

Record element	Type
read_address_channel	t_axi_read_address_channel
arid	std_logic_vector
araddr	std_logic_vector
arlen	std_logic_vector(7 downto 0)
arsize	std_logic_vector(2 downto 0)
arburst	std_logic_vector(1 downto 0)
arlock	std_logic
arcache	std_logic_vector(3 downto 0)
arprot	std_logic_vector(2 downto 0)
arqos	std_logic_vector(3 downto 0)
arregion	std_logic_vector(3 downto 0)
aruser	std_logic_vector
arvalid	std_logic
arready	std_logic

Record element	Type
write_data_channel	t_axi_write_data_channel
wdata	std_logic_vector
wstrn	std_logic_vector
wlast	std_logic
wuser	std_logic_vector
wvalid	std_logic
wready	std_logic

Record element	Type
write_response_channel	t_axi_write_response_channel
bid	std_logic_vector
bresp	std_logic_vector(1 downto 0)
buser	std_logic_vector
bvalid	std_logic
bready	std_logic

Record element	Type
read_data_channel	t_axi_read_data_channel
rid	std_logic_vector
rdata	std_logic_vector
rresp	std_logic_vector(1 downto 0)
rlast	std_logic
ruser	std_logic_vector
rvalid	std_logic
rready	std_logic

AXI parameter record types

Type name	Allowed value
t_axburst	FIXED
	INCR
	WRAP
t_axlock	NORMAL
	EXCLUSIVE
t_axprot	UNPRIVILEGED_NONSECURE_DATA
	UNPRIVILEGED_NONSECURE_INSTRUCTION
	UNPRIVILEGED_SECURE_DATA
	UNPRIVILEGED_SECURE_INSTRUCTION
	PRIVILEGED_NONSECURE_DATA
	PRIVILEGED_NONSECURE_INSTRUCTION
	PRIVILEGED_SECURE_DATA
	PRIVILEGED_SECURE_INSTRUCTION
t_xresp	OKAY
	EXOKAY
	SLVERR
	DECERR

BFM details

1 BFM procedure details and examples

Procedure	Description
axi_write()	

axi_write(awid_value, awaddr_value, awlen_value, awsize_value, awburst_value, awlock_value, awcache_value, awprot_value, awqos_value, awregion_value, awuser_value, wdata_value, wstrb_value, wuser_value, buser_value, bresp_value, msg, clk, axi_if, [scope, [msg_id_panel, [config]]])

The axi_write() procedure writes the given data to the given address of the DUT, using the AXI4 protocol. For protocol details, see the AXI4 specification.

- A log message is written if ID_BFM ID is enabled for the specified message ID panel.
- The default values of the various parameters are shown in the table below.

Parameter name	Type	Default value
awid_value	std_logic_vector	0
awaddr_value	unsigned	None
awlen_value	unsigned(7 downto 0)	0
awsize_value	Integer range 1 to 128	4
awburst_value	t_axburst	INCR
awlock_value	t_axlock	NORMAL
awcache_value	std_logic_vector(3 downto 0)	0
awprot_value	t_axprot	UNPRIVILEGED_UNSECURE_DATA
awqos_value	std_logic_vector(3 downto 0)	0
awregion_value	std_logic_vector(3 downto 0)	0
awuser_value	std_logic_vector	0
wdata_value	t_slv_array	None
wstrb_value	t_slv_array	(others => '1') for all words
wuser_value	t_slv_array	0 for all words
buser_value	std_logic_vector	None
bresp_value	std_logic_vector(1 downto 0)	None
msg	string	None
clk	std_logic	None
axi_if	t_axi_if	None
scope	string	C_SCOPE ("AXI BFM")
msg_id_panel	t_msg_id_panel	shared_msg_id_panel
config	t_axi_bfm_config	C_AXI_BFM_CONFIG_DEFAULT

The procedure reports an alert if:

- wready does not occur within max_wait_cycles clock cycles (alert level: max_wait_cycles_severity, set in the config)
- awready does not occur within max_wait_cycles clock cycles (alert level: max_wait_cycles_severity, set in the config)
- bvalid is not set within max_wait_cycles clock cycles (alert level: max_wait_cycles_severity, set in the config)

Examples:

```

axi_write(
    awid_value      => x"01",
    awaddr_value    => x"00000004",
    awlen_value     => x"01",
    awsize_value    => 4,
    awburst_value   => INCR,
    awlock_value    => NORMAL,
    awcache_value   => "0000",
    awprot_value    => UNPRIVILEGED_UNSECURE_DATA,
    awqos_value     => "0000",
    awregion_value  => "0000",
    awuser_value    => x"01",
    wdata_value     => t_slv_array'(x"12345678", x"33333333"),
    wstrb_value     => t_slv_array'(x"F", x"F"),
    wuser_value     => t_slv_array'(x"01", x"01"),
    buser_value     => v_buser_value,
    bresp_value     => v_bresp_value,
    msg             => "Writing data to Peripheral 1",
    clk             => clk,
    axi_if          => axi_if,
    scope           => C_SCOPE,
    msg_id_panel    => shared_msg_id_panel,
    config          => C_AXI_BFM_CONFIG_DEFAULT);

axi_write(
    awaddr_value    => x"00000004",
    wdata_value     => t_slv_array'(x"12345678", x"33333333"),
    buser_value     => v_buser_value,
    bresp_value     => v_bresp_value,
    msg             => "Writing data to Peripheral 1");

```

Suggested usage (requires local overload, see section 5):

```

axi_write(C_ADDR_DMA, x"AAAA", "Writing data to DMA");
axi_write(C_ADDR_MEMORY, x"FF", v_data_array, "Writing 256 data words to MEMORY");

```

axi_read()

axi_read(arid_value, araddr_value, arlen_value, arsize_value, arburst_value, arlock_value, arcache_value, arprot_value, arqos_value, arregion_value, aruser_value, rdata_value, rresp_value, ruser_value, msg, clk, axi_if, [scope, [msg_id_panel, [config, [proc_name]]]])

The axi_read() procedure reads data from the DUT at the given address, using the AXI4 protocol. For protocol details, see the AXI4 specification. The read data is placed on the output 'rdata_value' when the read has completed.

- The argument "ext_proc_call" is intended to be used internally, when the procedure is called by axi_check().
- A log message is written if ID_BFM ID is enabled for the specified message ID panel. This will only occur if the argument proc_name is left unchanged.
- The default values of the various parameters are shown in the table below.

Parameter name	Type	Default value
arid_value	std_logic_vector	0
araddr_value	unsigned	None
arlen_value	unsigned(7 downto 0)	0
arsize_value	Integer range 1 to 128	4
arburst_value	t_axburst	INCR

arlock_value	t_axlock	NORMAL
arcache_value	std_logic_vector(3 downto 0)	0
arprot_value	t_axprot	UNPRIVILEGED_UNSECURE_DATA
argos_value	std_logic_vector(3 downto 0)	0
arregion_value	std_logic_vector(3 downto 0)	0
aruser_value	std_logic_vector	0
rdata_value	t_slv_array	None
rresp_value	t_xresp_array	None
ruser_value	t_slv_array	None
msg	string	None
clk	std_logic	None
axi_if	t_axi_if	None
scope	string	C_SCOPE ("AXI BFM")
msg_id_panel	t_msg_id_panel	shared_msg_id_panel
config	t_axi_bfm_config	C_AXI_BFM_CONFIG_DEFAULT
ext_proc_call	string	""

The procedure reports an alert if:

- The received rid is different from the transmitted arid_value
- arready does not occur within max_wait_cycles clock cycles (alert level: max_wait_cycles_severity, set in the config)
- rvalid is not set within max_wait_cycles clock cycles (alert level: max_wait_cycles_severity, set in the config)

Examples:

```

axi_read(
  arid_value      => x"01",
  araddr_value    => x"00000004",
  arlen_value     => x"01",
  arsize_value    => 4,
  arburst_value   => INCR,
  arlock_value    => NORMAL,
  arcache_value   => "0000",
  arprot_value    => UNPRIVILEGED_UNSECURE_DATA,
  argos_value     => "0000",
  arregion_value  => "0000",
  aruser_value    => x"01",
  rdata_value     => v_rdata_value,
  rresp_value     => v_rresp_value,
  ruser_value     => v_ruser_value,
  msg             => "Read from Peripheral 1",
  clk             => clk,
  axi_if          => axi_if,
  scope           => C_SCOPE,
  msg_id_panel    => shared_msg_id_panel,
  config          => C_AXI_BFM_CONFIG_DEFAULT);

axi_read(
  araddr_value    => x"00000004",
  rdata_value     => v_rdata_value,
  rresp_value     => v_rresp_value,
  ruser_value     => v_ruser_value,
  msg             => "Read from Peripheral 1",
  clk             => clk,
  axi_if          => axi_if);

```

Suggested usage (requires local overload, see section 5):


```
axi_read(C_ADDR_IO, v_data_out, "Reading from IO device");
axi_read(C_ADDR_MEMORY, x"FF", v_data_array_out, "Reading 256 data words from MEMORY");
```

axi_check()

axi_check(arid_value, araddr_value, arlen_value, arsize_value, arburst_value, arlock_value, arcache_value, arprot_value, arqos_value, arregion_value, aruser_value, rdata_exp, rresp_exp, ruser_exp, msg, clk, axi_if, [alert_level, [scope, [msg_id_panel, [config]]]])

The axi_check() procedure reads data from the DUT at the given address, using the AXI4 protocol. For protocol details, see the AXI4 specification. After reading data from the AXI4 bus, the read data is compared with the expected data, 'rdata_exp'.

- If the check was successful, and the read data matches the expected data, a log message is written with ID_BFM ID (if this ID has been enabled).
- If the read data did not match the expected data, an alert with severity 'alert_level' will be reported.

Parameter name	Type	Default value
arid_value	std_logic_vector	0
araddr_value	unsigned	None
arlen_value	unsigned(7 downto 0)	0
arsize_value	Integer range 1 to 128	4
arburst_value	t_axburst	INCR
arlock_value	t_axlock	NORMAL
arcache_value	std_logic_vector(3 downto 0)	0
arprot_value	t_axprot	UNPRIVILEGED_UNSECURE_DATA
arqos_value	std_logic_vector(3 downto 0)	0
arregion_value	std_logic_vector(3 downto 0)	0
aruser_value	std_logic_vector	0
rdata_exp	t_slv_array	None
rresp_exp	t_xresp_array	OKAY for all words
ruser_exp	t_slv_array	0 for all words
msg	string	None
clk	std_logic	None
axi_if	t_axi_if	None
alert_level	t_alert_level	ERROR
scope	string	C_SCOPE ("AXI BFM")
msg_id_panel	t_msg_id_panel	shared_msg_id_panel
config	t_axi_bfm_config	C_AXI_BFM_CONFIG_DEFAULT

The procedure also report alerts for the same conditions as the axi_read() procedure.

Examples:

```
axi_check(
  arid_value    => x"01",
  araddr_value  => x"00000004",
  arlen_value   => x"01",
  arsize_value  => 4,
  arburst_value => INCR,
  arlock_value  => NORMAL,
  arcache_value => "0000",
  arprot_value  => UNPRIVILEGED_UNSECURE_DATA,
  arqos_value   => "0000",
  arregion_value => "0000",
  aruser_value  => x"01",
```

```
        rdata_exp      => t_slv_array'(x"12345678", x"33333333"),
        rresp_exp      => t_xresp_array'(OKAY, OKAY),
        ruser_exp      => t_slv_array'(x"00", x"00"),
        msg            => "Check data from Peripheral 1",
        clk            => clk,
        axi_if          => axi_if,
        alert_level     => ERROR,
        scope           => C_SCOPE,
        msg_id_panel    => shared_msg_id_panel,
        config          => C_AXI_BFM_CONFIG_DEFAULT);

axi_check(
    araddr_value      => x"00000004",
    rdata_exp         => v_rdata_exp,
    msg               => "Check data from Peripheral 1",
    clk               => clk,
    axi_if            => axi_if);
```

Suggested usage (requires local overload, see section 5):

```
axi_check(C_ADDR_UART_RX, x"3B", "Checking data in UART RX register");
axi_check(C_ADDR_MEMORY, x"FF", v_rdata_exp_array, "Checking 256 data words from MEMORY");
```

init_axi_if_signals()

init_axi_if_signals(addr_width, data_width, id_width, user_width)

This function initializes the AXI4 interface. All the BFM outputs are set to zeros ('0') and BFM inputs are set to 'Z'.

Note: This function assumes that awid, bid, arid and rid shares a common width (id_width) and that awuser, buser, aruser, ruser also share a common width (user_width)

Example:

```
axi_if <= init_axi_if_signals(addr_width, data_width, id_width, user_width);
```

2 BFM Configuration record

Type name: t_axi_bfm_config

Record element	Type	C_AXI_BFM_CONFIG_DEFAULT	Description
max_wait_cycles	natural	10	Used for setting the maximum cycles to wait before an alert is issued when waiting for ready and valid signals from the DUT.
max_wait_cycles_severity	t_alert_level	TB_FAILURE	The above timeout will have this severity
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 the 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.
num_aw_pipe_stages	natural	1	Write Address Channel pipeline steps
num_w_pipe_stages	natural	1	Write Data Channel pipeline steps
num_ar_pipe_stages	natural	1	Read Address Channel pipeline steps
num_r_pipe_stages	natural	1	Read Data Channel pipeline steps
num_b_pipe_stages	natural	1	Response Channel pipeline steps
id_for_bfm	t_msg_id	ID_BFM	The message ID used as a general message ID in the AXI BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT	The message ID used for logging waits in the AXI BFM
id_for_bfm_poll	t_msg_id	ID_BFM_POLL	The message ID used for logging polling in the AXI BFM

3 Additional Documentation

For additional documentation on the AXI4 standard, please see the AXI4 specification “AMBA® AXI™ and ACE™ Protocol Specification”, available from ARM.

4 Compilation

The AXI4 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 `axi_bfm_pkg.vhd` BFM can be compiled into any desired library. See the 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.

```
axi_write(C_ADDR_PERIPHERAL_1, C_TEST_DATA, "Sending data to Peripheral 1");
```

rather than

```
axi_write(
    awid_value      => x"01",
    awaddr_value    => x"00000004",
    awlen_value     => x"01",
    awsize_value    => 4,
    awburst_value   => INCR,
    awlock_value    => NORMAL,
    awcache_value   => "0000",
    awprot_value    => UNPRIVILEGED_UNSECURE_DATA,
    awqos_value     => "0000",
    awregion_value  => "0000",
    awuser_value    => x"01",
    wdata_value     => t_slv_array'(x"12345678", x"33333333"),
    wstrb_value     => t_slv_array'(x"F", x"F"),
    wuser_value     => t_slv_array'(x"01", x"01"),
    buser_value     => v_buser_value,
    bresp_value     => v_bresp_value,
    msg            => "Writing data to Peripheral 1",
    clk            => clk,
    axi_if         => axi_if,
    scope          => C_SCOPE,
    msg_id_panel   => shared_msg_id_panel,
    config         => C_AXI_BFM_CONFIG_DEFAULT);
```

By defining the local overload as e.g.:

```
procedure axi_write(
    constant addr_value : in unsigned;
    constant data_value : in std_logic_vector;
    constant msg        : in string
) is
    variable v_buser_value : std_logic_vector(C_USER_WIDTH-1 downto 0);
    variable v_bresp_value : t_xresp;
begin
    axi_write(
        awid_value      => x"00",           -- Setting a default value
        awaddr_value    => addr_value,      -- keep as is
        awlen_value     => x"00",          -- Set to length=1
        awsize_value    => 4,              -- Setting a default value
        awburst_value   => INCR,            -- Setting a default value
```

```

awlock_value    => NORMAL,                -- Setting a default value
awcache_value   => "0000",                -- Setting a default value
awprot_value    => UNPRIVILEGED_UNSECURE_DATA, -- Setting a default value
awqos_value     => "0000",                -- Setting a default value
awregion_value  => "0000",                -- Setting a default value
awuser_value    => x"01",                 -- Setting a default value
wdata_value     => data_value,            -- keep as is
wstrb_value     => x"f",                  -- Setting a default value
wuser_value     => x"01",                 -- Setting a default value
buser_value     => v_buser_value,         -- Assigning to a local variable
bresp_value     => v_bresp_value,        -- Assigning to a local variable
msg             => msg,                   -- keep as is
clk             => clk,                   -- Clock signal
axi_if          => axi_if,                -- Signal must be visible in local process scope
scope           => C_SCOPE,              -- Setting a default value
msg_id_panel    => shared_msg_id_panel,   -- Use global, shared msg_id_panel
config          => C_AXI_BFM_CONFIG_LOCAL); -- Use locally defined configuration or C_AXI_BFM_CONFIG_DEFAULT
end;
```

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

- Have address value as natural – and convert in the overload
- Set up defaults for constants. May be different for two overloads of the same BFM
- Apply dedicated msg_id_panel to allow dedicated verbosity control

IMPORTANT

This is a simplified Bus Functional Model (BFM) for AXI4.

The given BFM complies with the AXI4 protocol and thus allows a normal access towards an AXI4 interface. This BFM is not AXI4 protocol checker.

For a more advanced BFM please contact Bitvis AS at support@bitvis.no

INTELLECTUAL PROPERTY

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.