SPI BFM – Quick Reference

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.

BFM

spi_bfm_pkg.vhd

SPI Master (see page 2 for SPI Slave)

spi_master_transmit_and_receive (tx_data, rx_data, msg, spi_if, [see options below]) [1]

Options: action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config]

Master example: spi_master_transmit_and_receive(x"AA", v_data_out, "Sending data to Peripheral 1 and receiving data from Peripheral 1", spi_if);

***Suggested usage**: spi_master_transmit_and_receive(x"AA", v_data_out, "Transmitting 0xAA and receiving data from DUT"); -- Suggested usage requires local overload (see section 5)*

spi_master_transmit_and_check (tx_data, data_exp, msg, spi_if, [see options below]) [1]

Options: alert_level, action_when_transfer_is_done, action_between_words, alert_level, scope, msg_id_panel, config

Master example: spi_master_transmit_and_check(x"AA", x"F5", "Sending data to Peripheral 1 and checking received data from Peripheral 1", spi_if);

***Suggested usage**: spi_master_transmit_and_check(x"AA", x"F5", "Transmitting 0xAA and expecting 0xF5 from DUT"); -- Suggested usage requires local overload (see section 5)*

spi_master_transmit (tx_data, msg, spi_if, [see options below]) [1]

Options: action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config

Master example: spi_master_transmit(x"AA", "Sending data to Peripheral 1", spi_if);

***Suggested usage**: spi_master_transmit(C_ASCII_A, "Transmitting ASCII A to DUT"); -- Suggested usage requires local overload (see section 5)*

spi_master_receive (rx_data, msg, spi_if, [see options below]) [1]

Options: action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config

Master example: spi_master_receive(v_data_out, "Receive from Peripheral 1", spi_if);

***Suggested usage**: spi_master_receive(v_data_out, "Receive from Peripheral 1"); -- Suggested usage requires local overload (see section 5)*

spi_master_check (data_exp, msg, spi_if, [see options below]) [1]

Options: alert_level, action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config

Master example: spi_master_check(x"3B", "Expecting data from SPI", spi_if);

***Suggested usage**: spi_master_check(C_DATA_BYTE, "Expecting data byte"); -- Suggested usage requires local overload (see section 5)*

init_spi_if_signals (config, [master_mode])

Example: spi_if <= init_spi_if_signals(C_SPI_BFM_CONFIG_DEFAULT);

Note 1: the BFM configuration has to be defined and used when calling the SPI BFM procedures. See section 6 for an example of how to define a local BFM config.

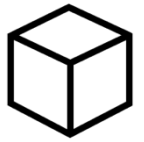
UVVM™

VHDL 2008 only

SPI BFM – Quick Reference

SPI Slave (see page 1 for SPI Master)

BFM



spi_bfm_pkg.vhd

spi_slave_transmit_and_receive (tx_data, rx_data, msg, spi_if, [see options below]) [1]

Options terminate_access, when_to_start_transfer, scope, msg_id_panel, config

Slave example: spi_slave_transmit_and_receive(x"AA", v_data_out, "Sending data to Peripheral 1 and receiving data from Peripheral 1", spi_if);

Suggested usage: *spi_slave_transmit_and_receive(x"AA", v_data_out, "Transmitting 0xAA and receiving data from DUT"); -- Suggested usage requires local overload (see section 5)*

spi_slave_transmit_and_check (tx_data, data_exp, msg, spi_if, [see options below]) [1]

Options: terminate_access, alert_level, when_to_start_transfer, scope, msg_id_panel, config

Slave example: spi_slave_transmit_and_check(x"AA", x"F5", "Sending data to Peripheral 1 and checking received data from Peripheral 1", spi_if);

Suggested usage: *spi_slave_transmit_and_check(x"AA", x"F5", "Transmitting 0xAA and expecting 0xF5 from DUT"); -- Suggested usage requires local overload (see section 5)*

spi_slave_transmit (tx_data, msg, spi_if, [see options below]) [1]

Options: terminate_access, when_to_start_transfer, scope, msg_id_panel, config

Slave example: spi_slave_transmit(x"AA", "Sending data to Peripheral 1", spi_if);

Suggested usage: *spi_slave_transmit(C_ASCII_A, "Transmitting ASCII A to DUT"); -- Suggested usage requires local overload (see section 5)*

spi_slave_receive (rx_data, msg, spi_if, [see options below]) [1]

Options: terminate_access, when_to_start_transfer, scope, msg_id_panel, config

Slave example: spi_slave_receive(v_data_out, "Receive from Peripheral 1", spi_if);

Suggested usage: *spi_slave_receive(v_data_out, "Receive from Peripheral 1"); -- Suggested usage requires local overload (see section 5)*

spi_slave_check (data_exp, msg, spi_if, [see options below]) [1]

Options: terminate_access, alert_level, when_to_start_transfer, scope, msg_id_panel, config

Slave example: spi_slave_check(x"3B", "Expecting data from SPI", spi_if);

Suggested usage: *spi_slave_check(C_DATA_BYTE, "Expecting data byte"); -- Suggested usage requires local overload (see section 5)*

Note 1: the BFM configuration has to be defined and used when calling the SPI BFM procedures. See section 6 for an example of how to define a local BFM config.

UVVM™

BFM Configuration record ´**t_spi_bfm_config**´

Record element	Type	C_SPI_BFM_CONFIG_DEFAULT
CPOL	std_logic	'0'
CPHA	std_logic	'0'
spi_bit_time	time	-1 ns
ss_n_to_sclk	time	20 ns
sclk_to_ss_n	time	20 ns
inter_word_delay	time	0 ns
match_strictness	t_match_strictness	MATCH_EXACT
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

Signal record ´**t_spi_if**´

Record element	Type
ss_n	std_logic
sclk	std_logic
mosi	std_logic
miso	std_logic

BFM non-signal parameters

Name	Type	Example(s)	Description
tx_data	std_logic_vector or t_slv_array	x"D3"	The data value to be transmitted to the DUT
rx_data	std_logic_vector or t_slv_array	x"D3"	SLV or array of SLVs where the received data will be stored
data_exp	std_logic_vector or t_slv_array	x"0D"	The data value to expect when receiving data from the slave. A mismatch results in an alert 'alert_level'
alert_level	t_alert_level	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the method.
action_when_transfer_is_done	t_action_when_transfer_is_done	RELEASE_LINE_AFTER_TRANSFER or HOLD_LINE_AFTER_TRANSFER	Determines if SPI master shall release or hold ss_n after the transfer is done. Default is RELEASE_LINE_AFTER_TRANSFER.
action_between_words	t_action_between_words	HOLD_LINE_BETWEEN_WORDS or RELEASE_LINE_BETWEEN_WORDS	Determines if SPI master shall release or hold ss_n between words when transmitting a t_slv_array. Default is HOLD_LINE_BETWEEN_WORDS.
terminate_access	std_logic	'0' or '1'	Determines if SPI slave transfer is performed. Setting this to '1' before a slave command is executed terminates the command. Default is '0'.
when_to_start_transfer	t_when_to_start_transfer	START_TRANSFER_ON_NEXT_SS or START_TRANSFER_IMMEDIATE	Determines if SPI slave shall wait for next ss_n if a transfer has already started. Default is START_TRANSFER_ON_NEXT_SS.
msg	string	"Receiving data"	A custom message to be appended in the log/alert.
scope	string	"SPI BFM"	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "SPI BFM". In a verification component, typically "SPI_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 ID panel defined in the adaptations package.
config	t_spi_bfm_config	C_SPI_BFM_CONFIG_DEFAULT	Configuration of BFM behaviour and restrictions. See section 2 for details.

BFM signal parameters

Name	Type	Description
spi_if	t_spi_if	See table "Signal record 't_spi_if'"

BFM details

1 BFM procedure details and examples

Procedure	Description

spi_master_transmit_and_receive()	**spi_master_transmit_and_receive (tx_data, rx_data, msg, spi_if, [see options below])**
	
	Options: action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config
	
	The spi_master_transmit_and_receive() procedure transmits the data in 'tx_data' to the DUT and stores the received data in 'rx_data', using the SPI protocol.
	For protocol details, see the SPI specification. When called, the spi_master_transmit_and_receive() procedure will set ss_n low . For a slave DUT to be able to transmit to a receiving master BFM, the master BFM must drive the sclk and ss_n signals and transmit data to the slave DUT.
	- This procedure is responsible for driving sclk and ss_n.
	- The SPI bit timing is given by config.spi_bit_time, config.spi_ss_n_to_sclk and config.sclk_to_ss_n.
	- The default value of action_when_transfer_is_done is RELEASE_LINE_AFTER_TRANSFER.
	- The default value of action_between_words is HOLD_LINE_BETWEEN_WORDS.
	- The default value of scope is C_SCOPE ("SPI BFM").
	- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
	- The default value of config is C_SPI_BFM_CONFIG_DEFAULT, see table on page 3.
	- A log message is written if ID_BFM ID is enabled for the specified message ID panel.
	- An error is reported if ss_n is not kept low during the entire transmission.
	- Note that action_between_words only apply for t_slv_array multi-word transfers.
	
	Examples:
	`spi_master_transmit_and_receive(x"AA", v_data_out, "Transmitting data to peripheral 1 and receiving data from peripheral 1", spi_if);`
	`spi_master_transmit_and_receive(x"AA", v_data_out, "Transmitting data to peripheral 1 and receiving data from peripheral 1", spi_if, RELEASE_LINE_AFTER_TRANSFER, HOLD_LINE_BETWEEN_WORDS, C_SCOPE, shared_msg_id_panel, C_SPI_BFM_CONFIG_DEFAULT);`
	Suggested usage (requires local overload, see section 5):
	`spi_master_transmit_and_receive(C_ASCII_A, v_data_out, "Transmitting ASCII A to DUT and receiving data from DUT");`

spi_master_transmit_and_check()	**spi_master_transmit_and_check (tx_data, data_exp, msg, spi_if, [see options below])** **Options**: alert_level, action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config The spi_master_transmit_and_check() procedure transmits the data in 'tx_data' and receives data from the DUT, using the transmit and receive procedure as described in the spi_master_transmit_and_receive() procedure. After receiving data from the DUT, the data is compared with the expected data, 'data_exp'. If the received data does not match the expected data, an alert with severity 'alert_level' will be triggered. If the received data matches 'data_exp', a message with ID config.id_for_bfm will be logged. In addition to the specifications listed in procedure spi_master_transmit_and_receive(), the following applies to the spi_master_transmit_and_check() procedure: - When called, the spi_master_transmit_and_check() procedure will in turn call spi_master_transmit_and_receive(). - The default value of alert_level is ERROR. - The procedure will report alerts for the same conditions and use similar default values as the spi_master_transmit_and_receive() procedure. - Note that action_between_words only apply for t_slv_array multi-word transfers. Example: `spi_master_transmit_and_check(x"AA", x"3B", "Transmitting data and checking received data on SPI interface", spi_if);`

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

```
spi_master_transmit_and_check(x"AA", C_CR_BYTE, "Transmitting 0xAA and expecting carriage return");
```

spi_master_transmit()

spi_master_transmit (tx_data, msg, spi_if, [see options below])

Options: action_when_transfer_is_done, actions_between_words, scope, msg_id_panel, config

The spi_master_transmit() procedure transmits the data in 'tx_data' to the DUT, using the transmit and receive procedure as described in the spi_master_transmit_and_receive() procedure.
In addition to the specifications listed in procedure spi_master_transmit_and_receive(), the following applies to the spi_master_transmit() procedure:

- When called, the spi_master_transmit() procedure will in turn call spi_master_transmit_and_receive().
- The received data from the slave DUT is ignored.
- The procedure will report alerts for the same conditions and use similar default values as the spi_master_transmit_and_receive() procedure.
- Note that action_between_words only apply for t_slv_array multi-word transfers.

Example:

```
spi_master_transmit(x"AA", "Transmitting data to peripheral 1", spi_if);
```

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

```
spi_master_transmit(C_ASCII_A, "Transmitting ASCII A to DUT");
```

spi_master_receive()

spi_master_receive (rx_data, msg, spi_if, [see options below])

Options: action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config

The spi_master_receive() procedure receives data from the DUT at the given address, using the transmit and receive procedure as described in the spi_master_transmit_and_receive() procedure.
In addition to the specifications listed in procedure spi_master_transmit_and_receive(), the following applies to the spi_master_receive() procedure:

- When called, the spi_master_receive() procedure will in turn call spi_master_transmit_and_receive().
- The spi_master_receive() procedure will transmit dummy data (0x0) to the DUT.
- The procedure will report alerts for the same conditions and use similar default values as the spi_master_transmit_and_receive() procedure.
- Note that action_between_words only apply for t_slv_array multi-word transfers.

Example:

```
spi_master_receive(v_data_out, "Receive from Peripheral 1", spi_if);
```

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

```
spi_master_receive(v_data_out, "Receive from Peripheral 1");
```

spi_master_check()

spi_master_check (data_exp, msg, spi_if, [see options below])

Options: alert_level, action_when_transfer_is_done, action_between_words, scope, msg_id_panel, config

The spi_master_check() procedure receives data from the DUT, using the transmit and receive procedure as described in the spi_master_transmit_and_receive() procedure. After receiving data from the DUT, the data is compared with the expected data, 'data_exp'. If the received data does not match the expected data, an alert with severity 'alert_level' will be triggered. If the received data matches 'data_exp', a message with ID config.id_for_bfm will be logged.
In addition to the specifications listed in procedure spi_master_transmit_and_receive(), the following applies to the spi_master_check() procedure:

- When called, the spi_master_check() procedure will in turn call procedure spi_master_transmit_and_receive().
- The default value of alert_level is ERROR.
- The procedure will report alerts for the same conditions and use similar default values as the spi_master_transmit_and_receive() procedure.
- Note that action_between_words only apply for t_slv_array multi-word transfers.
- The spi_master_check() procedure will transmit dummy data (0x0) to the DUT.

Example:

```
spi_master_check(x"3B", "Checking data on SPI interface", spi_if);
```

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

```
spi_master_check(C_CR_BYTE, "Expecting carriage return");
```

spi_slave_transmit_and_receive()

spi_slave_transmit_and_receive (tx_data, rx_data, msg, spi_if, [see options below])

Options: terminate_access, when_to_start_transfer, scope, msg_id_panel, config

The spi_slave_transmit_and_receive() procedure transmits the data in 'tx_data' to the DUT and stores the received data in 'rx_data', using the SPI protocol.
For protocol details, see the SPI specification.

- When called, the spi_slave_transmit_and_receive() procedure will wait for next ss_n, or start transfer and receive immediately, depending on the selection of when_to_start_transfer and if ss_n is already set. If terminate_access is '1' when this happens, the transfer and receive will be terminated in stead.
- The default value of terminate_access is '0'.
- The default value of when_to_start_transfer is START_TRANSFER_ON_NEXT_SS.
- The default value of scope is C_SCOPE ("SPI BFM")
- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
- The default value of config is C_SPI_BFM_CONFIG_DEFAULT, see table on page 3.
- A log message is written if ID_BFM ID is enabled for the specified message ID panel.
- An error is reported if ss_n is not kept low during the entire transmission.

Examples:

```
spi_slave_transmit_and_receive(x"AA", v_data_out, "Transmitting and receiving data from peripheral 1", spi_if);
spi_slave_transmit_and_receive(x"AA", v_data_out, "Transmitting and receiving data from peripheral 1", spi_if,
                               '0', START_TRANSFER_ON_NEXT_SS, C_SCOPE, shared_msg_id_panel, C_SPI_BFM_CONFIG_DEFAULT);
```

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

```
spi_slave_transmit_and_receive(C_ASCII_A, v_data_out, "Transmitting ASCII A to DUT and receiving data from DUT");
```

spi_slave_transmit_and_check()

spi_slave_transmit_and_check (tx_data, data_exp, msg, spi_if, [see options below])

Options: terminate_access, alert_level, when_to_start_transfer, scope, msg_id_panel, config

The spi_slave_transmit_and_check() procedure transmits the data in 'tx_data' and receives data from the DUT, using the transmit and receive procedure as described in the spi_slave_transmit_and_receive() procedure. After receiving data from the DUT, the data is compared with the expected data, 'data_exp'. If the received data does not match the expected data, an alert with severity 'alert_level' will be triggered. If the received data matches 'data_exp', a message with ID config.id_for_bfm will be logged.
In addition to the specifications listed in procedure spi_slave_transmit_and_receive(), the following applies to the spi_slave_transmit_and_check() procedure:

- When called, the spi_slave_transmit_and_check() procedure will in turn call spi_slave_transmit_and_receive().
- The default value of alert_level is ERROR.
- The procedure will report alerts for the same conditions and use similar default values as the spi_slave_transmit_and_receive() procedure.

Example:

```
spi_slave_transmit_and_check(x"AA", x"3B", "Transmitting data and checking received data on SPI interface", spi_if);
```

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

```
spi_slave_transmit_and_check(x"AA", C_CR_BYTE, "Transmitting 0xAA and expecting carriage return");
```

spi_slave_transmit()

spi_slave_transmit (tx_data, msg, spi_if, [see options below])

Options: terminate_access, when_to_start_transfer, scope, msg_id_panel, config

The spi_slave_transmit() procedure transmits the data in 'tx_data' to the DUT, using the spi_slave_transmit_and_receive() procedure.
In addition to the specifications listed in procedure spi_slave_transmit_and_receive(), the following applies to the spi_slave_transmit() procedure:

- When called, the spi_slave_transmit() procedure will in turn call procedure spi_slave_transmit_and_receive().
- The received data from the DUT is ignored.
- The procedure will report alerts for the same conditions and use similar default values as the spi_slave_transmit_and_receive() procedure.

Example:

```
spi_slave_transmit(x"AA", "Transmitting data to peripheral 1", spi_if);
```
Suggested usage (requires local overload, see section 5):
```
spi_slave_transmit(C_ASCII_A, "Transmitting ASCII A to DUT");
```

spi_slave_receive()

spi_slave_receive (rx_data, msg, spi_if, [see options below])

Options: terminate_access, when_to_start_transfer, scope, msg_id_panel, config

The spi_slave_receive() procedure receives data from the DUT, using the transmit and receive procedure as described in the spi_slave_transmit_and_receive() procedure.
In addition to the specifications listed in procedure spi_slave_transmit_and_receive(), the following applies to the spi_slave_receive() procedure:

- When called, the spi_slave_receive() procedure will in turn call spi_slave_transmit_and_receive().
- The spi_slave_receive() procedure will transmit dummy data (0x0) to the DUT.
- The procedure will report alerts for the same conditions and use similar default values as the spi_slave_transmit_and_receive() procedure.

Example:
```
spi_slave_receive(v_data_out, "Receive from Peripheral 1", spi_if);
```
Suggested usage (requires local overload, see section 5):
```
spi_slave_receive(v_data_out, "Receive from Peripheral 1");
```

spi_slave_check()

spi_slave_check (data_exp, msg, spi_if, [see options below])

Options: terminate_access, alert_level, when_to_start_transfer, scope, msg_id_panel, config

The spi_slave_check() procedure receives data from the DUT, using the transmit and receive procedure as described in the spi_slave_transmit_and_receive() procedure. After receiving data from the DUT, the data is compared with the expected data, 'data_exp'. If the received data does not match the expected data, an alert with severity 'alert_level' will be triggered. If the received data matches 'data_exp', a message with ID config.id_for_bfm will be logged.
In addition to the specifications listed in procedure spi_slave_transmit_and_receive(), the following applies to the spi_slave_check() procedure:

- When called, the spi_slave_check() procedure will in turn call procedure spi_slave_transmit_and_receive().
- The default value of alert_level is ERROR
- The spi_slave_check() procedure transmit dummy data (0x0) to the DUT.
- The procedure will report alerts for the same conditions and use similar default values as the spi_slave_transmit_and_receive() procedure.

Example:
```
spi_slave_check(x"3B", "Checking data on SPI interface", spi_if);
```
Suggested usage (requires local overload, see section 5):
```
spi_slave_check(C_CR_BYTE, "Expecting carriage return");
```

init_spi_if_signals

init_spi_if_signals(config, [master_mode])

This function initializes the SPI interface.

Master mode set true:

- ss_n initialized to 'H'
- if config.CPOL = '1', sclk initialized to 'H'. Otherwise, sclk initialized to 'L'
- miso and mosi initialized to 'Z'

Master mode set false:

- All signals initialized to 'Z'

Examples:
```
spi_if  <= init_spi_if_signals(C_SPI_BFM_CONFIG_DEFAULT); -- implicitly master mode since default is 'true'
spi_if  <= init_spi_if_signals(C_SPI_BFM_CONFIG_DEFAULT, true); -- explicitly indicating master mode
spi_if  <= init_spi_if_signals(C_SPI_BFM_CONFIG_DEFAULT, false); -- master_mode is false, i.e., shall act as a slave
```

2 BFM Configuration record

Type name: t_spi_bfm_config

Record element	Type	C_SPI_BFM_CONFIG_DEFAULT	Description
CPOL	std_logic	'0'	sclk polarity, i.e. the base value of the clock. If CPOL is '0', the clock will be set to '0' when inactive, i.e., ordinary positive polarity.
CPHA	std_logic	'0'	sclk phase, i.e. when data is sampled and transmitted w.r.t. sclk. If '0', sampling occurs on the first sclk edge and data is transmitted on the sclk active to idle state. If '1', data is sampled on the second sclk edge and transmitted on sclk idle to active state.
spi_bit_time	time	-1 ns	Used in master for dictating the sclk period. Default is -1 ns so that an alert can be raised if user forget to specify this.
ss_n_to_sclk	time	20 ns	Time from ss_n low until sclk active.
sclk_to_ss_n	time	20 ns	Time from last sclk until ss_n is released.
inter_word_delay	time	0 ns	Minimum time between words, from ss_n inactive to ss_n active.
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.
id_for_bfm	t_msg_id	ID_BFM	The message ID used as a general message ID in the SPI BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT	The message ID used for logging waits in the SPI BFM
id_for_bfm_poll	t_msg_id	ID_BFM_POLL	The message ID used for logging polling in the SPI BFM

3 Additional Documentation

For additional documentation on the SPI protocol, please see the SPI specification, e.g. "ST TN0897 Technical note ST SPI protocol. ID 023176 Rev 2".

4 Compilation

The SPI 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 spi_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.

```
spi_master_transmit_and_receive(C_ASCII_A, v_data_out, "Transmitting ASCII A");
```

rather than

```
spi_master_transmit_and_receive(C_ASCII_A, v_data_out, "Transmitting ASCII A", spi_if, RELEASE_LINE_AFTER_TRANSFER,
                                HOLD_LINE_BETWEEN_WORDS, C_SCOPE, shared_msg_id_panel, C_SPI_CONFIG_LOCAL);
```

By defining the local overload as e.g.:

```
procedure spi_master_transmit(
   constant tx_data      : in std_logic_vector;
   variable rx_data      : out std_logic_vector;
   constant msg          : in string) is
begin
   spi_master_transmit(tx_data,                        -- keep as is
                       rx_data,                        -- keep as is
                       msg,                            -- keep as is
                       spi_if,                         -- Signals must be visible in local process scope
                       RELEASE_LINE_AFTER_TRANSFER,    -- Use default, unless passing SLVs to master in a multi-word transfer
                       HOLD_LINE_BETWEEN_WORDS,        -- Use default, unless a t_slv_array is not intended as multi-word
                       C_SCOPE,                        -- Just use the default
                       shared_msg_id_panel,            -- Use global, shared msg id panel
                       C_SPI_CONFIG_LOCAL);            -- Use locally defined configuration

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 message ID panel to allow dedicated verbosity control

See section 6 for defining a BFM configuration to use with the local overload and when calling the BFM procedures.

6 Local BFM configuration

The SPI BFM requires that a local configuration is declared in the testbench and used in the BFM procedure calls. The default BFM configuration is defined with a bit period of -1 ns so that the BFM can detect and alert the user that the configuration has not been set. See section 2 for the SPI BFM configuration record fields.

Defining a local SPI BFM configuration:

```
constant C_SPI_CONFIG_local : t_spi_bfm_config := (
  CPOL             => '0',
  CPHA             => '0',
  spi_bit_time     => 200 ns,
  ss_n_to_sclk     => 301 ns,
  sclk_to_ss_n     => 301 ns,
  inter_word_delay => 0 ns,
  match_strictness => MATCH_EXACT,
  id_for_bfm       => ID_BFM,
  id_for_bfm_wait  => ID_BFM_WAIT,
  id_for_bfm_poll  => ID_BFM_POLL
);
```

See section 5 for how to define a local overload procedure and how to use a BFM config with the procedure call.

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