**SPI BFM** –Quick Reference

**BFM**

SPI Master (see page 2 for SPI Slave)

*spi\_bfm\_pkg.vhd*

|  |
| --- |
| 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]  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]) |
| 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]) |
| 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]) |
| 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]) |
| 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); |

**SPI BFM** –Quick Reference

SPI Slave (see page 1 for SPI Master)

**BFM**

|  |
| --- |
| spi\_slave\_transmit\_and\_receive (tx\_data, rx\_data, msg, spi\_if, [see options below]) |
| Options 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\_bfm\_pkg.vhd*

|  |
| --- |
| spi\_slave\_transmit\_and\_check (tx\_data, data\_exp, msg, spi\_if, [see options below]) |
| Options: 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]) |
| Options: 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]) |
| Options: 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]) |
| Options: 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)* |



BFM Configuration record ´**t\_spi\_bfm\_config´** Signal record ´**t\_spi\_if´**

|  |  |  |
| --- | --- | --- |
| **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 |
| 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 |
| ss\_n\_to\_sclk | time | 20 ns |

|  |  |
| --- | --- |
| **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. |
| 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

# 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**: 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. * 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,   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**: 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**: 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**: 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**: 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 |

# 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. |
| 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 |
|  |  |  |  |

# 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”.

# 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.

## 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.

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\_BFM\_CONFIG\_DEFAULT);

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 or C\_SPI\_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 message ID panel to allow dedicated verbosity control

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

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**