SPI VVC – Quick Reference

For general information see UVVM VVC Framework Essential Mechanisms located in uvvm_vvc_framework/doc. **CAUTION**: shaded code/description is preliminary

spi_vvc.vhd

SPI Master (see page 2 for SPI Slave)

spi_master _transmit_and_receive (VVCT, vvc_instance_idx, data, [TO_SB,] msg, [see options below])

Options: action_when_transfer_is_done, action_between_words

Master example: spi_master_transmit_and_receive(SPI_VVCT, 1, x"AF", “SPI Master Tx and Rx to/from Peripheral 1. Rx data will be stored in VVC to be retrieved later using fetch_result.”);
spi_master_transmit_and_receive(SPI_VVCT, 1, x"AF", TO_SB, “SPI Master Tx and Rx to/from Peripheral 1. Rx data will be sent to the SPI scoreboard for checking.”);

spi_master _transmit_only (VVCT, vvc_instance_idx, data, msg, [see options below])

Options: action_when_transfer_is_done, action_between_words

Master example: spi_master_transmit_only(SPI_VVCT, 1, x"AF", “Sending data to Peripheral 1”);

spi_master_receive_only (VVCT, vvc_instance_idx, [TO_SB,] msg, [see options below])

Options: num_words, action_when_transfer_is_done, action_between_words

Master example: spi_master_receive_only(SPI_VVCT, 1, “Receive data from Peripheral 1 and store it in VVC to be retrieved later using fetch_result() ”);
spi_master_receive_only(SPI_VVCT, 1, TO_SB, “Receive data from Peripheral 1 and send it to scoreboard for checking”);

spi_master_transmit_and_check (VVCT, vvc_instance_idx, data, data_exp, msg, [see options below])

Options: alert_level, action_when_transfer_is_done, action_between_words

Master example: spi_master_transmit_and_check(SPI_VVCT, 1, x"42", x"AF", “Sending data to Peripheral 1 and expecting data from Peripheral 1”);

spi_master_check_only (VVCT, vvc_instance_idx, data_exp, msg, [see options below])

Options: alert_level, action_when_transfer_is_done, action_between_words

Master example: spi_master_check_only(SPI_VVCT, 1, x"42",“Expect data from Peripheral 1”);

VHDL 2008 only

SPI VVC – Quick Reference

SPI Slave (see page 1 for SPI Master)

spi_vvc.vhd

spi_slave_transmit_and_receive (VVCT, vvc_instance_idx, data, [TO_SB] msg, [see options below])

Options: when_to_start_transfer

Slave example: spi_slave_transmit_and_receive(SPI_VVCT, 1, x"AF", “SPI Slave Tx and Rx to/from Peripheral 1. Rx data will be stored in VVC to be retrieved later using fetch_result.”);
spi_slave_transmit_and_receive(SPI_VVCT, 1, x"AF", TO_SB, “SPI Slave Tx and Rx to/from Peripheral 1. Rx data will be sent to the SPI scoreboard for checking.”);

spi_slave_transmit_only (VVCT, vvc_instance_idx, data, msg, [see options below])

Options: when_to_start_transfer

Slave example: spi_slave_transmit_only(SPI_VVCT, 1, x"AF", “Sending data to Peripheral 1”);

spi_slave_receive_only (VVCT, vvc_instance_idx, [TO_SB] msg, [see options below])

Options: num_words, when_to_start_transfer

Slave example: spi_slave_receive_only(SPI_VVCT, 1, “Receive from Peripheral 1 and store data in VVC to be retrieved by means of fetch_result()”);
spi_slave_receive_only(SPI_VVCT, 1, TO_SB, “Receive from Peripheral 1 and send data to scoreboard”);

spi_slave_transmit_and_check (VVCT, vvc_instance_idx, data, data_exp, msg, [see options below])

Options: alert_level, when_to_start_transfer

Slave example: spi_slave_transmit_and_check(SPI_VVCT, 1, x"42", x"AF", “Sending data to Peripheral 1 and expecting data from Peripheral 1”);

spi_slave_check_only (VVCT, vvc_instance_idx, data_exp, msg, [see options below])

Options: alert_level, when_to_start_transfer

Slave example: spi_slave_check_only(SPI_VVCT, 1, x"42",“Expect data from Peripheral 1”);

Common VVC procedures applicable for this VVC
- See UVVM Methods QuickRef for details.

Name
await_completion()
await_any_completion()
enable_log_msg()
disable_log_msg()
flush_command_queue()
terminate_current_command()
fetch_result()
insert_delay()

SPI VVC Configuration record 't_vvc_config'
- Accessible via **shared_spi_vvc_config** – see section 2.

Record element
inter_bfm_delay
[cmd/result]_queue_count_max
[cmd/result]_queue_count_threshold
[cmd/result]_queue_count_threshold_severity
bfm_config
msg_id_panel
unwanted_activity_severity

SPI VVC Status record signal 't_vvc_status'
- Accessible via **shared_spi_vvc_status** – see section 3.

Record element
current_cmd_idx
previous_cmd_idx
pending_cmd_idx

VVC target parameters

Name	Type	Example(s)	Description
VVCT	t_vvc_target_record	SPI_VVCT	VVC target type compiled into each VVC in order to differentiate between VVCs.
vvc_instance_idx	integer	1	Instance number of the VVC

VVC functional parameters

Name	Type	Example(s)	Description
data	std_logic_vector or t_slv_array	x"FF"	The data to be transmitted (in spi_<master/slave>_transmit_and_check or spi_<master/slave>_transmit_only).
data_exp	std_logic_vector or t_slv_array	x"FF"	The expected data to be received (in spi_<master/slave>_transmit_and_check or spi_<master/slave>_check_only).
msg	string	"Send to peripheral 1"	A custom message to be appended in the log/alert
num_words	positive	1, 2, 10	Number of words that shall be received. Default is 1.
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 STAR_TRANSFER_ON_NEXT_SS.
alert_level	t_alert_level	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the method.
scope	string	"SPI VVC"	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 ".

VVC entity signals

Name	Type	Direction	Description
spi_vvc_if	t_spi_if	Inout	See SPI BFM documentation

VVC entity generic constants

Name	Type	Default	Description
GC_DATA_WIDTH	natural	8	Bits in the SPI data word
GC_DATA_ARRAY_WIDTH	natural	C_SPI_VVC_DATA_ARRAY_WIDTH	Number of SPI data words in a data word array of type t_slv_array. Set in Util adaptations_pkg with default value of 32.
GC_INSTANCE_IDX	natural	1	Instance number to assign the VVC
GC_MASTER_MODE	boolean	TRUE	Whether the VVC shall act as an SPI master or an SPI slave on the bus.
GC_SPI_CONFIG	t_spi_bfm_config	C_SPI_BFM_CONFIG_DEFAULT	Configuration for the SPI BFM, see SPI BFM documentation.
GC_CMD_QUEUE_COUNT_MAX	natural	1000	Absolute maximum number of commands in the VVC command queue
GC_CMD_QUEUE_COUNT_THRESHOLD	natural	950	An alert will be generated when reaching this threshold to indicate that the command queue is almost full. The queue will still accept new commands until it reaches GC_CMD_QUEUE_COUNT_MAX.
GC_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY	t_alert_level	WARNING	Alert severity which will be used when command queue reaches GC_CMD_QUEUE_COUNT_THRESHOLD.
GC_RESULT_QUEUE_COUNT_MAX	natural	1000	Maximum number of unfetched results before result_queue is full.
GC_RESULT_QUEUE_COUNT_THRESHOLD	natural	950	An alert with severity 'result_queue_count_threshold_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0.
GC_RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY	t_alert_level	WARNING	Severity of alert to be initiated if exceeding result_queue_count_threshold

VVC details

All VVC procedures are defined in vvc_methods_pkg (dedicated this VVC), and uvvm_vvc_framework.td_vvc_framework_common_methods_pkg (common VVC procedures)
It is also possible to send a multicast to all instances of a VVC with ALL_INSTANCES as parameter for vvc_instance_idx.
Note: Every procedure here can be called without the optional parameters enclosed in [ ].

1 VVC procedure details and examples

Procedure	Description

spi_master_transmit_and_receive()	**spi_master_transmit_and_receive (VVCT, vvc_instance_idx, data, [TO_SB,] msg, [see options below])**
	
	Options: action_when_transfer_is_done, action_between_words, scope
	
	The spi_master_transmit_and_receive() VVC procedure adds a transmit and receive command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit and receive command is scheduled to run, the executor calls the SPI BFM spi_master_transmit_and_receive() procedure, described in the SPI BFM QuickRef. Note that action_between_words only apply for t_slv_array multi-word transfers.
	
	There is one requirement for running the spi_master_transmit_and_receive() procedure:
	- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to TRUE.
	
	If the option TO_SB is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).
	
	Example usage with fetch_result:
	`spi_master_transmit_and_receive (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1 and receiving data from Peripheral 1");`
	`spi_master_transmit_and_receive (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1 and receiving data from Peripheral 1", RELEASE_LINE_AFTER_TRANSFER, C_SCOPE);`
	Example with fetch_result() call: - result is placed in **v_data**
	`variable v_cmd_idx : natural; -- Command index for the last read`
	`variable v_data : t_vvc_result; -- Result from read`
	(…)
	`spi_master_transmit_and_receive(SPI_VVCT, 1, (x"AB", x"CD"), "Transmitting two bytes to Peripheral 1 and receiving from Peripheral 1");`
	`v_cmd_idx := get_last_received_cmd_idx(SPI_VVCT, 1);`
	`await_completion(SPI_VVCT,1, v_cmd_idx, 1 us, "Wait for transmit and receive to finish");`
	`fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching first byte from transmit and receive operation");`
	`fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching second byte from transmit and receive operation");`

spi_master_transmit_only()	**spi_master_transmit_only (VVCT, vvc_instance_idx, data, msg, [see options below])**
	
	Options: action_when_transfer_is_done, action_between_words, scope
	
	The spi_master_transmit_only() VVC procedure adds a transmit command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit command is scheduled to run, the executor calls the SPI BFM spi_master_transmit() procedure, described in the SPI BFM QuickRef.
	The SPI BFM spi_master_transmit () procedure will ignore the received data from the slave DUT. Note that action_between_words only apply for t_slv_array multi-word transfers.
	
	There is one requirement for running the spi_master_transmit_only() procedure:
	- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to TRUE.
	
	Examples:
	`spi_master_transmit_only (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1");`
	`spi_master_transmit_only (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1", RELEASE_LINE_AFTER_TRANSFER, HOLD_LINE_BETWEEN_WORDS, C_SCOPE);`

spi_master_receive_only()

spi_master_receive_only (VVCT, vvc_instance_idx, data, [TO_SB,] msg, [see options below])

Options: num_words, action_when_transfer_is_done, action_between_words, scope

The spi_master_receive_only() VVC procedure adds a receive command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the receive command is scheduled to run, the executor calls the SPI BFM spi_master_receive() procedure, described in the SPI BFM QuickRef.
The received data from DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data will be stored in the VVC for a potential future fetch (see example with fetch_result below). When receiving multiple words, each word must be fetched separately with the same command index. The SPI BFM spi_master_transmit() procedure will transmit dummy data (0x0) while receiving data from the slave DUT.

There is one requirement for running the spi_master_receive_only() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to TRUE.

If the option TO_SB is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).

Note: The data returned from fetch_result is of type t_vvc_result. It is a SLV with length C_VVC_CMD_DATA_MAX_LENGTH. The received data is located at indices (GC_DATA_WIDTH-1 downto 0).

Example usage with fetch_result:

```
spi_master_receive_only (SPI_VVCT, 1, "Receiving from Peripheral 1");
spi_master_receive_only (SPI_VVCT, 1, "Receiving from Peripheral 1", 6, RELEASE_LINE_AFTER_TRANSFER,
                         RELEASE_LINE_BETWEEN_WORDS, C_SCOPE);
```

Example with fetch_result() call: - result is placed in **v_data**

```
variable v_cmd_idx         : natural;      -- Command index for the last read
variable v_data            : t_vvc_result; -- Result from read
(…)
spi_master_receive_only(SPI_VVCT, 1, "Receiving from Peripheral 1");
v_cmd_idx := get_last_received_cmd_idx(SPI_VVCT, 1);
await_completion(SPI_VVCT,1, v_cmd_idx, 1 us, "Wait for receive to finish");
fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching result from receive operation");
```

spi_master_transmit_and_check()

spi_master_transmit_and_check (VVCT, vvc_instance_idx, data, data_exp, msg, [see options below])

Options: alert_level, action_when_transfer_is_done, action_between_words, scope

The spi_master_transmit_and_check() VVC procedure adds a transmit and a check command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit and the check command is scheduled to run, the executor calls the SPI BFM spi_master_transmit_and_check() procedure, described in the SPI BFM QuickRef. Note that action_between_words only apply to t_slv_array multi-word transfers and the default value of alert_level is ERROR.

There is one requirement for running the spi_master_transmit_and_check() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to TRUE.

Examples:

```
spi_master_transmit_and_check (SPI_VVCT, 1, x"0D", x"5F", "Transmitting carriage return to Peripheral 1 and expecting data from
                               Peripheral 1");
spi_master_transmit_and_check (SPI_VVCT, 1, C_CR_BYTE, x"5F", "Transmitting carriage return to Peripheral 1 and expecting data
                               from Peripheral 1", ERROR, RELEASE_LINE_AFTER_TRANSFER, HOLD_LINE_BETWEEN_WORDS, C_SCOPE);
```

spi_master_check_only()

spi_master_check_only (VVCT, vvc_instance_idx, data, msg, [see options below])

Options: alert_level, action_when_transfer_is_done, action_between_words, scope

The spi_master_check_only() VVC procedure adds a check command to the SPI VVC executor queue, which will run as soon as all preceding commands have completed. When the check command is scheduled to run, the executor calls the SPI BFM spi_master_check() procedure, described in the SPI BFM QuickRef. The received data will not be stored by this procedure and the SPI BFM spi_master_check() procedure will transmit dummy data (0x0) while receiving data from the slave DUT.
Note that action_between_words only apply to t_slv_array multi-word transfers and the default value of alert_level is ERROR.

There is one requirement for running the spi_master_check_only() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to TRUE.

Examples:

```
spi_master_check_only (SPI_VVCT, 1, x"0D", "Expecting carriage return from Peripheral 1");
spi_master_check_only (SPI_VVCT, 1, C_CR_BYTE, "Expecting carriage return from Peripheral 1", ERROR,
                       RELEASE_LINE_AFTER_TRANSFER, HOLD_LINE_BETWEEN_WORDS, C_SCOPE);
```

spi_slave_transmit_and_receive()

spi_slave_transmit_and_receive (VVCT, vvc_instance_idx, data, [TO_SB,] msg, [see options below])

Options: when_to_start_transfer, scope

The spi_slave_transmit_and_receive() VVC procedure adds a transmit and receive command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit and receive command is scheduled to run, the executor calls the SPI BFM spi_slave_transmit_and_receive () procedure, described in the SPI BFM QuickRef.

There is one requirement for running the spi_slave_transmit_and_reveice () procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

If the option TO_SB is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).

Example usage with fetch_result:

```
    spi_slave_transmit_and_receive (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1 and receiving data from
                                    Peripheral 1");
    spi_slave_transmit_and_receive (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1 and receiving data from
                                    Peripheral 1", START_TRANSFER_ON_NEXT_SS, C_SCOPE);
Example with fetch_result() call: - result is placed in v_data
    variable v_cmd_idx         : natural;      -- Command index for the last read
    variable v_data            : t_vvc_result; -- Result from read
(…)
    spi_slave_transmit_and_receive(SPI_VVCT, 1, (x"AB", x"CD"), "Transmitting two bytes to Peripheral 1 and receiving from
                                   Peripheral 1");
    v_cmd_idx := get_last_received_cmd_idx(SPI_VVCT, 1);
    await_completion(SPI_VVCT,1, v_cmd_idx, 1 us, "Wait for transmit and receive to finish");
    fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching first byte from transmit and receive operation");
    fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching second byte from transmit and receive operation");
```

spi_slave_transmit_only()

spi_slave_transmit_only (VVCT, vvc_instance_idx, data, msg, [see options below])

Options: when_to_start_transfer, scope

The spi_slave_transmit_only() VVC procedure adds a transmit command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit command is scheduled to run, the executor calls the SPI BFM spi_slave_transmit () procedure, described in the SPI BFM QuickRef. The SPI BFM spi_slave_transmit() procedure will ignore the data received from the master DUT.

There is one requirement for running the spi_slave_transmit () procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

Examples:

```
spi_slave_transmit_only (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1");
spi_slave_transmit_only (SPI_VVCT, 1, x"0D", "Transmitting carriage return to Peripheral 1", START_TRANSFER_ON_NEXT_SS, C_SCOPE);
```

spi_slave_receive_only()

spi_slave_receive_only (VVCT, vvc_instance_idx, [TO_SB,] msg, [see options below])

Options: num_words, when_to_start_transfer, scope

The spi_slave_receive_only() VVC procedure adds a receive command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the receive command is scheduled to run, the executor calls the SPI BFM spi_slave_receive () procedure, described in the SPI BFM QuickRef.
The received data will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the received data will be stored in the VVC for a potential future fetch (see example with *fetch_result* below). When receiving multiple words, each word must be fetched separately with the same command index. The SPI BFM spi_slave_receive() procedure will transmit dummy data (0x0) while receiving data from the master DUT.

There is one requirement for running the spi_slave_receive_only() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

If the option TO_SB is applied, the received data will be sent to the I2C dedicated scoreboard. There, it is checked against the expected value (provided by the testbench).

Note: The data returned from fetch_result is of type t_vvc_result. It is a SLV with length C_VVC_CMD_DATA_MAX_LENGTH. The received data is located at indices (GC_DATA_WIDTH-1 downto 0).

Example usage with fetch_result:

```
spi_slave_receive_only (SPI_VVCT, 1, "Receiving from Peripheral 1");
spi_slave_receive_only (SPI_VVCT, 1, "Receiving from Peripheral 1", 6, START_TRANSFER_IMMEDIATE, C_SCOPE);
```

Examples with fetch_result() call: - result is placed in **v_data**

```
variable v_cmd_idx         : natural;      -- Command index for the last read
variable v_data            : t_vvc_result; -- Result from read
(…)
spi_slave_receive_only(SPI_VVCT, 1, "Receiving from Peripheral 1");
v_cmd_idx := get_last_received_cmd_idx(SPI_VVCT, 1);
await_completion(SPI_VVCT,1, v_cmd_idx, 1 us, "Wait for receive to finish");
fetch_result(SPI_VVCT,1, v_cmd_idx, v_data, "Fetching result from receive operation");
```

spi_slave_transmit_and_check()

spi_slave_transmit_and_check (VVCT, vvc_instance_idx, data, data_exp, msg, [see_options_below])

Options: alert_level, when_to_start_transfer, scope

The spi_slave_transmit_and_check() VVC procedure adds a transmit and a check command to the SPI VVC executor queue, that will run as soon as all preceding commands have completed. When the transmit and the check command is scheduled to run, the executor calls the SPI BFM spi_slave_transmit_and_check() procedure, described in the SPI BFM QuickRef. Note that the default value of alert_level is ERROR.

There is one requirement for running the spi_slave_transmit_and_check() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

Example:

```
spi_slave_transmit_and_check (SPI_VVCT, 1, x"0D", x"5F", "Transmitting carriage return to Peripheral 1 and expecting data from
                              Peripheral 1");
spi_slave_transmit_and_check (SPI_VVCT, 1, x"0D", x"5F", "Transmitting carriage return to Peripheral 1 and expecting data from
                              Peripheral 1", ERROR, START_TRANSFER_IMMEDIATE, C_SCOPE);
```

spi_slave_check_only()

spi_slave_check_only (VVCT, vvc_instance_idx, data, msg, [see options below])

Options: alert_level, when_to_start_transfer, scope

The spi_slave_check_only() VVC procedure adds a check command to the SPI VVC executor queue, which will run as soon as all preceding commands have completed. When the check command is scheduled to run, the executor calls the SPI BFM spi_slave_check() procedure, described in the SPI BFM QuickRef. The received data will not be stored by this procedure and the SPI BFM spi_slave_check() procedure will transmit dummy data (0x0) while receiving data from the master DUT.

There is one requirement for running the spi_slave_check_only() procedure:

- The VVC entity with instance index corresponding to the 'vvc_instance_idx' parameter must have the generic constant GC_MASTER_MODE set to FALSE.

Examples.

```
spi_slave_check_only(SPI_VVCT, 1, x"0D", "Expecting carriage return from Peripheral 1");
spi_slave_check_only(SPI_VVCT, 1, C_CR_BYTE, "Expecting carriage return from Peripheral 1", ERROR,
                     START_TRANSFER_ON_NEXT_SS, C_SCOPE);
```

2 VVC Configuration

Record element	Type	C_SPI_VVC_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_SPI_INTER_BFM_DELAY_DEFAULT	Delay between any requested BFM accesses towards the DUT. - TIME_START2START: Time from a BFM start to the next BFM start (A TB_WARNING will be issued if access takes longer than TIME_START2START). - TIME_FINISH2START: Time from a BFM end to the next BFM start. Any insert_delay() command will add to the above minimum delays, giving for instance the ability to skew the BFM starting time.
cmd_queue_count_max	natural	C_MAX_COMMAND_QUEUE	Maximum pending number in command queue before queue is full. Adding additional commands will result in an ERROR.
cmd_queue_count_threshold	natural	C_CMD_QUEUE_COUNT_THRESHOLD	An alert with severity "cmd_queue_count_threshold_severity" will be issued if command queue exceeds this count. Used for early warning if command queue is almost full. Will be ignored if set to 0.
cmd_queue_count_threshold_severity	t_alert_level	C_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be triggered if command count exceeding cmd_queue_count_threshold

result_queue_count_max	natural	C_RESULT_QUEUE_COUNT_MAX	Maximum number of unfetched results before result_queue is full.
result_queue_count_threshold	natural	C_RESULT_QUEUE_COUNT_THRESHOLD	An alert with severity 'result_queue_count_threshold_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0.
result_queue_count_threshold_severity	t_alert_level	C_ RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be initiated if exceeding result_queue_count_threshold
bfm_config	t_spi_bfm_config	C_SPI_BFM_CONFIG_DEFAULT	Configuration for SPI BFM. See QuickRef for SPI BFM
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT	VVC dedicated message ID panel. See section 16 of uvvm_vvc_framework/doc/UVVM_VVC_Framework_Essential_Mechanisms.pdf for how to use verbosity control.
unwanted_activity_severity	t_alert_level	C_UNWANTED_ACTIVITY_SEVERITY	Severity of alert to be initiated if unwanted activity on the DUT outputs is detected. Unwanted activity detection is enabled (ERROR) by default.

Note: cmd/result queue parameters in the VVC Configuration are unused and will be removed in v3.0, use instead the entity generic constants.

The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

```
shared_spi_vvc_config(C_VVC_IDX_MASTER_1).inter_bfm_delay.delay_in_time := 10 ms;
shared_spi_vvc_config(C_VVC_IDX_SLAVE_1).bfm_config.CPOL               := '1';
```

3 VVC Status

The current status of the VVC can be retrieved during simulation. This is done by reading from the shared variable shared_spi_vvc_status record from the test sequencer. The record contains status for both channels, specified with the channel axis of the shared_spi_vvc_status array.The record contents can be seen below:

Record element	Type	Description
current_cmd_idx	natural	Command index currently running
previous_cmd_idx	natural	Previous command index to run
pending_cmd_cnt	natural	Pending number of commands in the command queue

4 Transaction Info

This VVC supports transaction info, a UVVM concept for distributing transaction information in a controlled manner within the complete testbench environment. The transaction info may be used in many different ways, but the main purpose is to share information directly from the VVC to a DUT model.

*Table 5.1 SPI transaction info record fields. Transaction type: t_base_transaction (BT) **- accessible via shared_spi_vvc_transaction_info.bt**.*

Info field	Type	Default	Description
operation	t_operation	NO_OPERATION	Current VVC operation, e.g. INSERT_DELAY, POLL_UNTIL, READ, WRITE.
data	t_slv_array	(others => (others => '0'))	The data to be transmitted (in spi_<master/slave>_transmit_and_check or spi_<master/slave>_transmit_only).
data_exp	t_slv_array	(others => (others => '0'))	The expected data to be received (in spi_<master/slave>_transmit_and_check or spi_<master/slave>_check_only).
num_words	natural	0x0	Number of words that shall be received. Default is 1.
word_length	natural	0x0	Length of words to be sent or received.
when_to_start_transfer	when_to_start_transfer	START_TRANSFER_IMMEDIATE	Determines if SPI slave shall wait for next ss_n if a transfer has already started.
action_when_transfer_is_done	action_when_transfer_is_done	RELEASE_LINE_AFTER_TRANSFER	Determines if SPI master shall release or hold ss_n after the transfer is done.
action_between_words	action_between_words	HOLD_LINE_BETWEEN_WORDS	Determines if SPI master shall release or hold ss_n between words when transmitting a t_slv_array.
vvc_meta	t_vvc_meta	C_VVC_META_DEFAULT	VVC meta data of the executing VVC command.
→ msg	string	" "	Message of executing VVC command.

→ cmd_idx	integer	-1	Command index of executing VVC command.
transaction_status	t_transaction_status	C_TRANSACTION_STATUS_DEFAULT	Set to INACTIVE, IN_PROGRESS, FAILED or SUCCEEDED during a transaction.

See UVVM VVC Framework Essential Mechanisms PDF, section 6, for additional information about transaction types and transaction info usage.

5 Scoreboard

This VVC has built in Scoreboard functionality where data can be routed by setting the `TO_SB` parameter in supported method calls, e.g. `master_receive_only()`. Note that the data is only stored in the scoreboard and not accessible with the `fetch_result()` method when the `TO_SB` parameter is applied.

The SPI VVC scoreboard is per default a 32 bits wide standard logic vector. When sending expected data to the scoreboard, where the data width is smaller than the default scoreboard width, we recommend zero-padding the data with the pad_spi_sb () function. E.g. `SPI_VVC_SB.add_expected(<SPI VVC instance number>, pad_spi_sb (<exp data>));`

See the Generic Scoreboard Quick Reference PDF in the Bitvis VIP Scoreboard document folder for a complete list of available commands and additional information. The SPI VVC scoreboard is accessible from the testbench as a shared variable `SPI_VVC_SB`, located in the `vvc_methods_pkg.vhd`. All of the listed Generic Scoreboard commands are available for the SPI VVC scoreboard using this shared variable.

6 Activity watchdog

The VVCs support a centralized VVC activity register which the activity watchdog uses to monitor the VVC activities. The VVCs will register their presence to the VVC activity register at start-up, and report when ACTIVE and INACTIVE, using dedicated VVC activity register methods, and trigger the global_trigger_vvc_activity_register signal during simulations. The activity watchdog is continuously monitoring the VVC activity register for VVC inactivity and raises an alert if no VVC activity is registered within the specified timeout period.

Include `activity_watchdog(num_exp_vvc, timeout, [alert_level, [msg]])` in the testbench to start using the activity watchdog.
Note that setting the exact number of expected VVCs in the VVC activity register can be omitted by setting `num_exp_vvc = 0`.

More information can be found in UVVM Essential Mechanisms PDF in the UVVM VVC Framework doc folder.

7 Unwanted Activity Detection

This VVC supports detection of unwanted activity from the DUT. This mechanism will give an alert if the DUT generates any unexpected bus activity. It assures that no data is output from the DUT when it is not expected, e.g. SPI receive/check VVC methods are not called. Once the VVC is inactive, it starts to monitor continuously on the DUT outputs. When unwanted activity is detected, the VVC issues an alert of severity.

The unwanted activity detection can be configured from the central testbench sequencer, where the severity of alert can be changed to a different value.
To disable this feature in the testbench, e.g.:

```
shared_spi_vvc_config(C_VVC_INDEX).unwanted_activity_severity := NO_ALERT;
```

For SPI VVC, the unwanted activity detection is enabled (`unwanted_activity_severity := ERROR`) by default.

8 Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm_vvc_framework/doc/”.
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”.

9 Compilation

The SPI VVC must be compiled with VHDL 2008.
It is dependent on the following libraries

- ***UVVM Utility Library (UVVM-Util), version 2.19.5 and up***
- ***UVVM VVC Framework, version 2.12.7 and up***
- ***SPI BFM***
- ***Bitvis VIP Scoreboard***

Before compiling the SPI VVC, make sure that uvvm_vvc_framework, uvvm_util and bitvis_vip_scoreboard have been compiled.

See UVVM Essential Mechanisms located in uvvm_vvc_framework/doc for information about compile scripts.

Compile order for the SPI VVC:

Compile to library	File	Comment
bitvis_vip_spi	spi_bfm_pkg.vhd	SPI BFM
bitvis_vip_spi	transaction_pkg.vhd	SPI transaction package with DTT types, constants etc.
bitvis_vip_spi	vvc_cmd_pkg.vhd	SPI VVC command types and operations
bitvis_vip_spi	../uvvm_vvc_framework/src_target_dependent/td_target_support_pkg.vhd	UVVM VVC target support package, compiled into the SPI VVC library.
bitvis_vip_spi	../uvvm_vvc_framework/src_target_dependent/td_vvc_framework_common_methods_pkg.vhd	UVVM framework common methods compiled into the SPI VVC library
bitvis_vip_spi	vvc_methods_pkg.vhd	SPI VVC methods
bitvis_vip_spi	../uvvm_vvc_framework/src_target_dependent/td_queue_pkg.vhd	UVVM queue package for the VVC
bitvis_vip_spi	../uvvm_vvc_framework/src_target_dependent/td_vvc_entity_support_pkg.vhd	UVVM VVC entity methods compiled into the SPI VVC library
bitvis_vip_spi	spi_vvc.vhd	SPI VVC

10 Simulator compatibility and setup

See README.md for a list of supported simulators.
For required simulator setup see ***UVVM-Util*** Quick reference.

IMPORTANT
This is a simplified Verification IP (VIP) for SPI.
The given VIP complies with the basic SPI protocol and thus allows a normal access towards a SPI interface. This VIP is not a SPI protocol checker.
For a more advanced VIP please contact UVVM at info@uvvm.org