**SPI VVC** –Quick Reference

**VVC**

**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.**

**Please refer to section 2 of Avalon\_mm\_vvc\_QuickRef for example usage of protected shared variables**

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

SPI Master (see page 2 for SPI Slave)

*spi\_vvc.vhd*

|  |
| --- |
| 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”); |



**SPI VVC** –Quick Reference   
SPI Slave (see page 1 for SPI Master)

**VVC**

|  |
| --- |
| 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\_vvc.vhd*

|  |
| --- |
| 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. |  | **SPI VVC Configuration record ‘t\_vvc\_config’** - Accessible via **shared\_spi\_vvc\_config** – see section 2. |  | **SPI VVC Status record signal ‘t\_vvc\_status’** - Accessible via **shared\_spi\_vvc\_status** – see section 3. |
| **Name** |  | **Record element** |  | **Record element** |
| await\_completion() |  | inter\_bfm\_delay |  | current\_cmd\_idx |
| await\_any\_completion() |  | [cmd/result]\_queue\_count\_max |  | previous\_cmd\_idx |
| enable\_log\_msg() |  | [cmd/result]\_queue\_count\_threshold |  | pending\_cmd\_idx |
| disable\_log\_msg() |  | [cmd/result]\_queue\_count\_threshold\_severity |  |  |
| flush\_command\_queue() |  | bfm\_config |  |  |
| terminate\_current\_command() |  | msg\_id\_panel |  |  |
| fetch\_result() |  |  |  |  |
| insert\_delay() |  |  |  |  |

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 C\_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 [ ].*

# 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, HOLD\_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\_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); |

# 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. |

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’;

# 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 |

# 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) **-** accessiblevia **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.

# 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.

# 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.

# 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”.

# Compilation

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

* ***UVVM Utility Library (UVVM-Util), version 2.15.0 and up***
* ***UVVM VVC Framework, version 2.11.0 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 |

# 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 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**