**SBI 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.

|  |
| --- |
| sbi\_write (VVCT, vvc\_instance\_idx, addr, data | { num\_words, randomisation}, msg, [scope]) |
| Example: sbi\_write(SBI\_VVCT, 1, x”1000”, x“40”, “Set baud rate to 9600”);  sbi\_write(SBI\_VVCT, 1, x”1001”, 7, RANDOM, “Write 7 random bytes to UART TX”); |

*sbi\_vvc.vhd*

|  |
| --- |
| sbi\_read (VVCT, vvc\_instance\_idx, addr, [TO\_SB,] msg, [scope]) |
| Example: sbi\_read(SBI\_VVCT, 1, x”1000”, “Read baud rate”);  sbi\_read(SBI\_VVCT, 1, x”1002”, TO\_SB, “Read UART RX and send to Scoreboard”, C\_SCOPE); |

|  |
| --- |
| sbi\_check (VVCT, vvc\_instance\_idx, addr, data, msg, [alert\_level, [scope]]) |
| Example: sbi\_check(SBI\_VVCT, 1, x”1155, x”3B”, “Check data from UART RX”); |

|  |
| --- |
| sbi\_poll\_until (VVCT, vvc\_instance\_idx, addr, data, msg, [max\_polls, [timeout, [alert\_level, [scope]]]]) |
| Example: sbi\_poll\_until(SBI\_VVCT, 1, x”1155”, x”0D”, “Read UART RX until CR is found”); |

SBI VVC Configuration record **´vvc\_config´ --** accessible via **shared\_sbi\_vvc\_config**

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

**await\_completion**() **enable\_log\_msg**() **disable\_log\_msg**()

**fetch\_result**()

**flush\_command\_queue**()  
**terminate\_current\_command**() **terminate\_all\_commands**() **insert\_delay**()

**get\_last\_received\_cmd\_idx()**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_SBI\_VVC\_CONFIG\_DEFAULT** |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_SBI\_INTER\_BFM\_DELAY\_DEFAULT |
| cmd\_queue\_count\_max | natural | C\_CMD\_QUEUE\_COUNT\_MAX |
| cmd\_queue\_count\_threshold | natural | C\_CMD\_QUEUE\_COUNT\_THRESHOLD |
| cmd\_queue\_count\_threshold\_severity | t\_alert\_level | C\_CMD\_QUEUE\_COUNT\_THRESHOLD\_SEVERITY |
| result\_queue\_count\_max | natural | C\_RESULT\_QUEUE\_COUNT\_MAX |
| result\_queue\_count\_threshold | natural | C\_RESULT\_QUEUE\_COUNT\_THRESHOLD |
| result\_queue\_count\_threshold\_severity | t\_alert\_level | C\_RESULT\_QUEUE\_COUNT\_THERSHOLD\_SEVERITY |
| bfm\_config | t\_sbi\_bfm\_config | C\_SBI\_BFM\_CONFIG\_DEFAULT |
| msg\_id\_panel | t\_msg\_id\_panel | C\_VVC\_MSG\_ID\_PANEL\_DEFAULT |
|  |  |  |

SBI VVC Status record signal **´vvc\_status´ --** accessible via **shared\_sbi\_vvc\_status**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** |  |
| current\_cmd\_idx | natural |  | |
| previous\_cmd\_idx | natural |  |
| pending\_cmd\_cnt | natural |  |

VVC target parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| VVCT | t\_vvc\_target\_record | SBI\_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** |
| addr | unsigned | x”5A” | The address of a SW accessible register. Could be offset or full address depending on the DUT |
| data | std\_logic\_vector | x”D3” | The data to be written (in sbi\_write) or the expected data (in sbi\_check/sbi\_poll\_until). |
| msg | string | “Read from DUT” | A custom message to be appended in the log/alert |
| timeout | time | 100 ns | Timeout to be used in the sbi\_poll\_until BFM procedure. 0 ns means no timeout. |
| max\_polls | integer | 1 | Maximum number of polls allowed in the sbi\_poll\_until procedure. 0 means no limit. |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the procedure. |
| scope | string | “SBI VVC” | A string describing the scope from which the log/alert originates. In a simple single sequencer typically  "SBI BFM". In a verification component typically "SBI VVC ". |

VVC entity signals

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Direction** | **Description** |
| clk | std\_logic | Input | VVC Clock signal |
| sbi\_vvd\_master\_if | t\_sbi\_if | Inout | See SBI BFM documentation |

VVC entity generic constants

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| GC\_ADDR\_WIDTH | integer | 8 | Width of the SBI address bus |
| GC\_DATA\_WIDTH | integer | 32 | Width of the SBI data bus |
| GC\_INSTANCE\_IDX | natural | 1 | Instance number to assign the VVC |
| GC\_SBI\_CONFIG | t\_sbi\_bfm\_config | C\_SBI\_BFM\_CONFIG\_DEFAULT | Configuration for the SBI BFM, see SBI 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** |
| **sbi\_write()** | **sbi\_write(VVCT, vvc\_instance\_idx, addr, data | { num\_words, randomisation}, msg, [scope])**  The sbi\_write() VVC procedure adds a write command to the SBI VVC executor queue, which will run as soon as all preceding commands have completed.  The sbi\_write() command has two variants using either just data for a basic single transaction, or num\_words + randomisation for a more advanced version.  When the basic write command is scheduled to run, the executor calls the SBI BFM sbi\_write() procedure, described in the SBI BFM QuickRef.  When the more advanced randomisation command is applied the basic BFM sbi\_write() transaction is executed num\_words times with new random data each time – according to the given randomisation profile.  Current defined randomisation profiles are: RANDOM: Standard uniform random. This is provided as an example.  Example:  sbi\_write(SBI\_VVCT, 1, x”1000”, x“40”, “Set UART baud rate to 9600”, C\_SCOPE);  sbi\_write(SBI\_VVCT, 1, x”1001”, 7, RANDOM, “Write 7 random bytes to UART TX”);  It is recommended to use constants to improve the readability of the code, e.g.:  sbi\_write(SBI\_VVCT, 1, C\_ADDR\_UART\_BAUDRATE, C\_BAUDRATE\_9600, “Set UART baud rate to 9600”); |
| **sbi\_read()** | **sbi\_read (VVCT, vvc\_instance\_idx, addr, [TO\_SB,] msg, [scope])**  The sbi\_read() VVC procedure adds a read command to the SBI VVC executor queue, which will run as soon as all preceding commands have completed. When the read command is scheduled to run, the executor calls the SBI BFM sbi\_read() procedure, described in the SBI BFM QuickRef.  The value read from DUT will not be returned in this procedure call since it is non-blocking for the sequencer/caller, but the read data will be stored in the VVC for a potential future fetch (see example with *fetch\_result* below).  If the option TO\_SB is applied the read data will be sent to the SBI\_VVC dedicated scoreboard where it will be checked against the expected value (provided by the testbench)  Example:  sbi\_read(SBI\_VVCT, 1, x”1000”, “Read UART baud rate”, C\_SCOPE);  sbi\_read(SBI\_VVCT, 1, x”1002”, TO\_SB, “Read UART RX and send to Scoreboard”, C\_SCOPE);  It is recommended to use constants to improve the readability of the code, e.g.:  sbi\_read(SBI\_VVCT, 1, C\_ADDR\_UART\_BAUDRATE, “Read UART baud rate”);  **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 : work.vvc\_cmd\_pkg.t\_vvc\_result; -- Result from read.  (…)  sbi\_read(SBI\_VVCT, 1, C\_ADDR\_UART\_BAUDRATE, “Read from Peripheral 1”);  v\_cmd\_idx := get\_last\_received\_cmd\_idx(SBI\_VVCT, 1);  await\_completion(SBI\_VVCT,1, v\_cmd\_idx, 1 us, "Wait for read to finish");  fetch\_result(SBI\_VVCT,1, v\_cmd\_idx, **v\_data**, "Fetching result from read operation"); |
| **sbi\_check()** | **sbi\_check (VVCT, vvc\_instance\_idx, addr, data, msg, [alert\_level, [scope]])**  The sbi\_check() VVC procedure adds a check command to the SBI 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 SBI BFM sbi\_check() procedure, described in the SBI BFM QuickRef. The sbi\_check() procedure will perform a read operation, then check if the read data is equal to the expected data in the ‘data’ parameter. If the read data is not equal to the expected ‘data’ parameter, an alert with severity ‘alert\_level’ will be issued. The read data will not be stored in this procedure.  Examples:  sbi\_check(SBI\_VVCT, 1, x”1155, x”3B”, “Check data from UART RX”);  sbi\_check(SBI\_VVCT, 1, x”1155, x”3B”, “Check data from UART RX”, TB\_ERROR, C\_SCOPE):  It is recommended to use constants to improve the readability of the code, e.g.:  sbi\_check(SBI\_VVCT, 1, C\_ADDR\_UART\_RX, C\_UART\_START\_BYTE, “Check data from UART RX”); |
| **sbi\_poll\_until()** | **sbi\_poll\_until (VVCT, vvc\_instance\_idx, addr, data, msg, [max\_polls, [timeout, [alert\_level, [scope]]]])**  The sbi\_poll\_until() VVC procedure adds a poll\_until command to the SBI VVC executor queue, which will run as soon as all preceding commands have completed. When the write command is scheduled to run, the executor calls the SBI BFM sbi\_poll\_until() procedure, described in the SBI BFM QuickRef. The sbi\_poll\_until() procedure will perform a read operation, then check if the read data is equal to the data in the ‘data’ parameter. If the read data is not equal to the expected ‘data’ parameter, the process will be repeated until the read data is equal to the expected data, or the procedure is terminated by either a terminate command, a timeout or the poll limit set in max\_polls.  The read data will not be stored by this procedure.  Examples:  sbi\_poll\_until(SBI\_VVCT, 1, x”1155”, x”0D”, “Read UART RX until CR is found”);  sbi\_poll\_until(SBI\_VVCT, 1, x”1155”, x”0D”, “Read UART RX until CR is found”, 5, 0 ns, TB\_WARNING, C\_SCOPE);  It is recommended to use constants to improve the readability of the code, e.g.:  sbi\_poll\_until(SBI\_VVCT, 1, C\_ADDR\_UART\_RX, C\_CR\_BYTE, “Read UART RX until CR is found”); |

# VVC Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record element** | **Type** | **C\_SBI\_BFM\_CONFIG\_DEFAULT** | | **Description** | |
| inter\_bfm\_delay | t\_inter\_bfm\_delay | C\_SBI\_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\_CMD\_QUEUE\_COUNT\_MAX | 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 initiated if 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\_sbi\_bfm\_config | C\_SBI\_BFM\_CONFIG\_DEFAULT | Configuration for SBI BFM. See quick reference for SBI 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\_sbi\_vvc\_config(1).inter\_bfm\_delay.delay\_in\_time := 50 ns;

shared\_sbi\_vvc\_config(1).bfm\_config.id\_for\_bfm := ID\_BFM;

# VVC Status

The current status of the VVC can be retrieved during simulation. This is achieved by reading from the shared variable shared\_sbi\_vvc\_status record from the test sequencer. 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 |

# 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.

# 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 SBI transaction info record fields. Transaction Type: t\_base\_transaction (BT) **-** accessiblevia **shared\_sbi\_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. |
| address | unsigned(31 downto 0) | 0x0 | Address of the SBI read or write transaction. |
| data | slv(31 downto 0) | 0x0 | Data for SBI read or write transaction. |
| 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. |

Table 5.2 SBI transaction info record fields. Transaction type: t\_compound\_transaction (CT) ) **-** accessiblevia **shared\_sbi\_vvc\_transaction\_info.ct**.

|  |  |  |  |
| --- | --- | --- | --- |
| **DTT field** | **Type** | **Default** | **Description** |
| operation | t\_operation | NO\_OPERATION | Current VVC operation, e.g. INSERT\_DELAY, POLL\_UNTIL, READ, WRITE. |
| address | unsigned(31 downto 0) | 0x0 | Address of the SBI read or write transaction. |
| data | slv(31 downto 0) | 0x0 | Data for SBI read or write transaction. |
| randomisation | t\_randomisation | NA | sbi\_write() will generate random data when set to RANDOM. |
| num\_words | natural | 1 | Use with randomisation to write a *num\_words* number of random words using a single sbi\_write() command. |
| max\_polls | integer | 1 | Maximum number of polls allowed in the sbi\_poll\_until procedure. 0 means no limit. |
| 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, i.e. sbi\_read(). 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 SBI scoreboard is accessible from the testbench as a shared variable SBI\_VVC\_SB, located in the vvc\_methods\_pkg.vhd. E.g. SBI\_VVC\_SB.add\_expected(C\_SBI\_VVC\_IDX, pad\_sb\_slv(v\_expected), “Adding expected”);

The SBI 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\_sbi\_sb () function. E.g. SBI\_VVC\_SB.add\_expected(<SBI VVC instance number>, pad\_sbi\_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. All of the listed Generic Scoreboard commands are available for the SBI VVC scoreboard using the SBI\_VVC\_SB.

# VVC Interface

In this VVC, the interface has been encapsulated in a signal record of type *t\_sbi\_if* in order to improve readability of the code. Since the SBI interface busses can be of arbitrary size, the interface vectors have been left unconstrained. These unconstrained vectors need to be constrained when the interface signals are instantiated. For this interface, it could look like:

signal sbi\_if\_1 : t\_sbi\_if( addr (C\_ADDR\_WIDTH-1 downto 0),

wdata(C\_DATA\_WIDTH-1 downto 0),

rdata(C\_DATA\_WIDTH-1 downto 0) );

# Additional Documentation

Additional documentation about UVVM and its features can be found under “/uvvm\_vvc\_framework/doc/”.

For additional documentation on the SBI protocol, please see the SBI BFM QuickRef.

# Compilation

The SBI 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***
* ***SBI BFM***
* ***Bitvis VIP Scoreboard***

Before compiling the SBI VVC, assure 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 SBI VVC:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_sbi | sbi\_bfm\_pkg.vhd | SBI BFM |
| bitvis\_vip\_sbi | transaction\_pkg.vhd | SBI transaction package with DTT types, constants etc. |
| bitvis\_vip\_sbi | vvc\_cmd\_pkg.vhd | SBI VVC command types and operations |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_target\_support\_pkg.vhd | UVVM VVC target support package, compiled into the SBI VVC library. |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_framework\_common\_methods\_pkg.vhd | Common UVVM framework methods compiled into the SBI VVC library |
| bitvis\_vip\_sbi | vvc\_methods\_pkg.vhd | SBI VVC methods |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_queue\_pkg.vhd | UVVM queue package for the VVC |
| bitvis\_vip\_sbi | ../uvvm\_vvc\_framework/src\_target\_dependent/td\_vvc\_entity\_support\_pkg.vhd | UVVM VVC entity support compiled into the SBI VVC library |
| bitvis\_vip\_sbi | sbi\_vvc.vhd | SBI 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 SBI.

The given VIP complies with the basic SBI protocol and thus allows a normal access towards a SBI interface. This VIP is not a SBI protocol checker.

For a more advanced VIP please contact Bitvis AS at [support@bitvis.no](mailto: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**