**I2C BFM** –Quick Reference

**NOTE: As of UVVM v3.x, all shared variables have been made protected. This means that any access to shared variables must be done**

**using get- and set-methods. This documentation has not yet been updated with the methods for accessing these variables, but will be very soon.**

|  |
| --- |
| i2c\_master\_transmit (addr\_value, data, msg, i2c\_if, [action\_when\_transfer\_is\_done, [scope, [msg\_id\_panel, [config]]]]) 1 |
| Example: i2c\_master\_transmit(x”AA”, x”10”, “Sending data to Peripheral 1”, i2c\_if); -- Send byte x”10” to slave at address x”AA”  *Suggested usage: i2c\_master\_transmit(C\_ASCII\_A, “Transmitting ASCII A to DUT”); -- Suggested usage requires local overload (see section 5)* |

I2C Master (see page 2 for I2C Slave)

**BFM**

|  |
| --- |
| init\_i2c\_if\_signals (VOID) |
| Example: i2c\_if <= init\_i2c\_if\_signals(VOID); |

|  |
| --- |
| i2c\_master\_receive (addr\_value, data, msg, i2c\_if, [action\_when\_transfer\_is\_done, [scope, [msg\_id\_panel, [config, [proc\_name]]]]]) 1 |
| Example: i2c\_master\_receive(x”BB”, v\_data\_out, “Receive from Peripheral 1”, i2c\_if); -- Receive a single byte from slave  *Suggested usage: i2c\_master\_receive(v\_data\_out, “Receive from Peripheral 1”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| i2c\_master\_check (addr\_value, data\_exp, msg, i2c\_if, [action\_when\_transfer\_is\_done, [alert\_level, [scope, [msg\_id\_panel, [config]]]]]) 1 |
| Example: i2c\_master\_check (x”CC”, x"3B", “Checking data from I2C”, i2c\_if); -- Verify that byte received from slave at address x”CC” is equal to x”3B”  *Suggested usage: i2c\_master\_check(C\_CR\_BYTE, “Expecting carriage return”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| i2c\_master\_quick\_command (addr\_value, msg, i2c\_if, [rw\_bit, [exp\_ack, [action\_when\_transfer\_is\_done, [alert\_level, [scope, [msg\_id\_panel, [config]]]]]])1 |
| Example: i2c\_master\_quick\_command(C\_I2C\_SLAVE\_DUT\_ADDR\_1, "Quick Command to I2C slave", i2c\_if);  *Suggested usage: i2c\_master\_quick\_command(C\_I2C\_SLAVE\_DUT\_ADDR\_1, “Pinging I2C slave”); -- Suggested usage requires local overload (see section 5)*, [rw\_bit, [exp\_ack, |



Note 1: the BFM configuration has to be defined and used when calling the I2C BFM procedures. See section 6 for an example of how to define a local BFM config.

*i2c\_bfm\_pkg.vhd*

**I2C BFM** –Quick Reference

I2C Slave (see page 1 for I2C Master)

|  |
| --- |
| i2c\_slave\_transmit (data, msg, i2c\_if, [scope, [msg\_id\_panel, [config]]]) 1 |
| Example: i2c\_slave\_transmit(x”10”, “Sending data to master”, i2c\_if); -- Send byte x”10” to master  *Suggested usage: i2c\_slave\_transmit(C\_ASCII\_A, “Transmitting ASCII A to master DUT”); -- Suggested usage requires local overload (see section 5)* |

**BFM**



|  |
| --- |
| i2c\_slave\_receive (data, msg, i2c\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]]) 1 |
| Example: i2c\_slave\_receive(v\_data\_out, “Receive from master”, i2c\_if); -- Receive a single byte from master  *Suggested usage: i2c\_slave\_receive(v\_data\_out, “Receive from Master”); -- Suggested usage requires local overload (see section 5)* |

*i2c\_bfm\_pkg.vhd*

|  |
| --- |
| i2c\_slave\_check (data\_exp, msg, i2c\_if, [exp\_rw\_bit, [alert\_level, [scope, [msg\_id\_panel, [config]]]]]) 1 |
| Example: i2c\_slave\_check (x"3B", “Checking data from I2C”, i2c\_if); -- Verify that byte received from master is equal to x”3B”  *Suggested usage: i2c\_slave\_check(C\_CR\_BYTE, “Expecting carriage return”); -- Suggested usage requires local overload (see section 5)* |



Note 1: the BFM configuration has to be defined and used when calling the I2C BFM procedures. See section 6 for an example of how to define a local BFM config.

|  |  |  |
| --- | --- | --- |
| BFM Configuration record ´**t\_i2c\_bfm\_config´** |  |  |
| **Record element** | **Type** | **C\_I2C\_BFM\_CONFIG\_DEFAULT** |
| enable\_10\_bits\_addressing | boolean | FALSE |
| master\_sda\_to\_scl | time | 20 ns |
| master\_scl\_to\_sda | time | 20 ns |
| master\_stop\_condition\_hold\_time | time | 20 ns |
| max\_wait\_scl\_change | time | 10 ms |
| max\_wait\_scl\_change\_severity | t\_alert\_level | FAILURE |
| max\_wait\_sda\_change | time | 10 ms |
| max\_wait\_sda\_change\_severity | t\_alert\_level | FAILURE |
| i2c\_bit\_time | time | -1 ns |
| i2c\_bit\_time\_severity | t\_alert\_level | FAILURE |
| acknowledge\_severity | t\_alert\_level | FAILURE |
| slave\_mode\_address | unsigned | “0000000000” |
| slave\_mode\_address\_severity | t\_alert\_level | FAILURE |
| slave\_rw\_bit\_severity | t\_alert\_level | FAILURE |
| reserved\_address\_severity | t\_alert\_level | WARNING |
| match\_strictness | t\_match\_strictness | MATCH\_EXACT |
| id\_for\_bfm | t\_msg\_id | ID\_BFM |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL |

|  |  |
| --- | --- |
| BFM signal parameters | Signal record ´**t\_i2c\_if´** |
| |  |  |  | | --- | --- | --- | | **Name** | **Type** | **Description** | | i2c\_if | t\_i2c\_if | See table "Signal record 'i2c\_if'" | | |  |  | | --- | --- | | **Record element** | **Type** | | scl | std\_logic | | sda | std\_logic | |

|  |  |  |  |
| --- | --- | --- | --- |
| BFM non-signal parameters | | | |
| **Name** | **Type** | **Example(s)** | **Description** |
| addr\_value | unsigned | x”A3” | Slave address. Only applicable to the I2C master methods. Valid address lengths are 7 bits and 10 bits.  7-bit addresses with the four most-significant bits equal to x”0” and x”F” are reserved by the I2C standard. Please see the NXP I2C specification for more information about reserved addresses. |
| data | std\_logic\_vector  t\_byte\_array | x”D3”  [x”AB”, x“BA”, x”AD”, x”DA”] | The data value to be transmitted to the DUT, either a single byte or a byte array. |
| data\_exp | std\_logic\_vector  t\_byte\_array | x”0D”  [x”CB”, x“BF”, x”A0”, x”DB”] | The data value to expect when receiving the addressed register. A mismatch results in an alert with severity ‘alert\_level’. Either a single byte or a byte array. |
| exp\_rw\_bit | std\_logic | ‘0’ | Expected R/W# bit for the slave check procedure. ‘1’ for read, ‘0’ for write. |
| alert\_level | t\_alert\_level | ERROR or TB\_WARNING | Set the severity for the alert that may be asserted by the method. |
| msg | string | “Receiving data” | A custom message to be appended in the log/alert. |
| action\_when\_transfer\_is\_done | t\_action\_when\_transfer\_is\_done | RELEASE\_LINE\_AFTER\_TRANSFER or HOLD\_LINE\_AFTER\_TRANSFER | Sets whether or not the I2C master method shall generate a stop condition after the operation is finished. Only applicable to the I2C master methods.  RELEASE\_LINE\_AFTER\_TRANSFER: Generate stop condition at the end of current operation.  HOLD\_LINE\_AFTER\_TRANSFER: Do not generate a stop condition since the master shall continue to occupy the bus. The master will then generate another start condition at the beginning of the next operation, which the slave will interpret as a repeated start condition. See NXP I2C specification for details. |
| scope | string | "I2C BFM" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "I2C BFM". In a verification component typically "I2C\_VVC". |
| msg\_id\_panel | t\_msg\_id\_panel | shared\_msg\_id\_panel | Optional msg\_id\_panel, controlling verbosity within a specified scope. Defaults to a common ID panel defined in the adaptations package. |
| config | t\_i2c\_bfm\_config | C\_I2C\_BFM\_CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 2 for details. |
| Note: All signals are active high. | | | |

BFM details

# BFM procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **i2c\_master\_transmit()** | **i2c\_master\_transmit (addr\_value, data, msg, i2c\_if, [action\_when\_transfer\_is\_done, [scope, [msg\_id\_panel, [config]]]])**  The i2c\_master\_transmit() procedure transmits the data in ‘data’ to the slave DUT at address ‘addr\_value’ using the I2C protocol. For protocol details, see the NXP I2C specification.   * The default value of scope is C\_SCOPE (“I2C BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_I2C\_BFM\_CONFIG\_DEFAULT, see table on the first page. * The default value of action\_when\_transfer\_is\_done is ‘RELEASE\_LINE\_AFTER\_TRANSFER’. * A log message is written if ID\_BFM ID is enabled for the specified message ID panel.   The procedure reports an alert if:   * The i2c\_if signals do not change within the timeouts given in config. Verifies that the bus is alive. * The ‘addr\_value’ is wider than 7 bits in 7-bit addressing mode * The ‘addr\_value’ is wider than 10 bits in 10-bit addressing mode * The ‘addr\_value’ is equal to a I2C specification reserved address in 7-bit addressing mode * If ‘data’ is of type std\_logic\_vector: The data is wider than 8 bits. * If ‘data’ is of type t\_byte\_array: The byte array is descending (using downto). * A slave holds the ‘scl’ signal low for longer than ‘config.i2c\_bit\_time’. * The acknowledge bit set by the slave DUT after every transmitted byte is not ‘0’.   Examples:  i2c\_master\_transmit(x”AA”, x”10”, “Transmitting data to peripheral 1”, i2c\_if);  i2c\_master\_transmit(x”AA”, byte\_array(0 to 3), “Transmitting data to peripheral 1”, i2c\_if, RELEASE\_LINE\_AFTER\_TRANSFER,   C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5);  i2c\_master\_transmit(C\_ASCII\_A, “Transmitting ASCII A to DUT”); |
| **i2c\_master\_receive()** | **i2c\_master\_receive (addr\_value, data, msg, i2c\_if, [action\_when\_transfer\_is\_done, [scope, [msg\_id\_panel, [config, [proc\_name]]]]])**  The i2c\_master\_receive() procedure receives data from the slave DUT at address ‘addr\_value’ using the I2C protocol and stores it in ‘data’. For protocol details, see the NXP I2C specification. In addition to the specifications listed in the i2c\_master\_transmit() procedure, the following applies:   * The default value of proc\_name is “i2c\_master\_receive”. This argument is intended to be used internally, when procedure is called by i2c\_master\_check(). * A log message is written if ID\_BFM ID is enabled for the specified message ID panel. This will only occur if the argument proc\_name is left unchanged.   The procedure will report alerts for the same conditions as the i2c\_master\_transmit() procedure.  Examples:  i2c\_master\_receive(x”BB”, v\_data\_out, “Receive from Peripheral 1”, i2c\_if);  i2c\_master\_receive(x”BB”, v\_data\_out, “Receive from Peripheral 1”, i2c\_if, RELEASE\_LINE\_AFTER\_TRANSFER,   C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  i2c\_master\_receive(v\_data\_out, “Receive from Peripheral 1”); |
| **i2c\_master\_check()** | **i2c\_master\_check (addr\_value, data\_exp, msg, i2c\_if, [action\_when\_transfer\_is\_done, [alert\_level, [scope, [msg\_id\_panel, [config]]]]])**  The i2c\_master\_check() procedure receives data from the slave DUT add address ‘addr\_value’, using the receive procedure as described in the i2c\_master\_receive() procedure. After receiving data, the data is compared with the expected data, ‘data\_exp’.  In addition to the specifications listed in the i2c\_master\_transmit() procedure, the following applies:   * The default value of alert\_level is ERROR * If the data was received successfully, and the received data matches the expected data, a log message is written with ID ID\_BFM (if this ID has been enabled). * If the received data did not match the expected data, an alert with severity ‘alert\_level’ will be reported.   The procedure will also report alerts for the same conditions as the i2c\_master\_receive() procedure.  Example:  i2c\_master\_check(x”CC”, x"3B", ERROR, “Expect data on I2C”, i2c\_if);  Suggested usage (requires local overload, see section 5):  i2c\_master\_check (C\_CR\_BYTE, “Expecting carriage return”);  i2c\_master\_check (C\_CR\_BYTE, ERROR, “Expecting carriage return”); |
| **i2c\_slave\_transmit()** | **i2c\_slave\_transmit (data, msg, i2c\_if, [scope, [msg\_id\_panel, [config]]])**  The i2c\_slave\_transmit() procedure transmits the data in ‘data’ to the I2C master DUT using the I2C protocol. For protocol details, see the NXP I2C specification.   * The default value of scope is C\_SCOPE (“I2C BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_I2C\_BFM\_CONFIG\_DEFAULT, see table on the first page. * A log message is written if ID\_BFM ID is enabled for the specified message ID panel.   The procedure reports an alert if:   * The i2c\_if signals do not change within the timeouts given in config. Verifies that the bus is alive. * The ‘config.slave\_mode\_address’ has its 3 most-significant bits (9-7) set when in 7-bit addressing mode * The ‘config.slave\_mode\_address’ is equal to a I2C specification reserved address in 7-bit addressing mode * If ‘data’ is of type std\_logic\_vector: The data is wider than 8 bits. * If ‘data’ is of type t\_byte\_array: The byte array is descending (using downto). * The received address is not equal to the address set in ‘config.slave\_mode\_address’. * The Read/Write bit received from the master is not as expected. * The acknowledge bit set by the master DUT after every transmitted byte is not as expected. Expects ACK (‘0’) after every byte except the very last byte where a NACK (‘1’) is expected.     Examples:  i2c\_slave\_transmit(x”AA”, “Transmitting data to master”, i2c\_if);  i2c\_slave\_transmit(x”AA”, “Transmitting data to master”, i2c\_if, C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  i2c\_slave\_transmit(C\_ASCII\_A, “Transmitting ASCII A to master”); |
| **i2c\_slave\_receive()** | **i2c\_slave\_receive (data, msg, i2c\_if, [scope, [msg\_id\_panel, [config, [proc\_name]]]])**  The i2c\_slave\_receive() procedure receives data from the I2C master DUT using the I2C protocol and stores it in ‘data’. For protocol details, see the I2C specification. In addition to the specifications listed in the i2c\_slave\_transmit() procedure, the following applies:   * The default value of proc\_name is “i2c\_slave\_receive”. This argument is intended to be used internally, when procedure is called by i2c\_slave\_check(). * A log message is written if ID\_BFM ID is enabled for the specified message ID panel. This will only occur if the argument proc\_name is left unchanged.   The procedure will also report alerts for the same conditions as the i2c\_slave\_receive() procedure.  Examples:  i2c\_slave\_receive(v\_data\_out, “Receive from master”, i2c\_if);  i2c\_slave\_receive(v\_data\_out, “Receive from master”, i2c\_if, C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5):  i2c\_slave\_receive(v\_data\_out, “Receive from master”); |
| **i2c\_slave\_check()** | **i2c\_slave\_check (data\_exp, msg, i2c\_if, [exp\_rw\_bit, [alert\_level, [scope, [msg\_id\_panel, [config]]]]])**  The i2c\_slave\_check() procedure receives data from the master DUT, using the receive procedure as described in the i2c\_slave\_receive() procedure. After receiving data, the data is compared with the expected data, ‘data\_exp’. In addition to the specifications listed in the i2c\_slave\_transmit() procedure, the following applies:   * The default for exp\_rw\_bit is ‘0’ (Write). If this parameter is set to ‘1’ (read) the data\_exp parameter needs to be an empty byte\_array. If this is not the case, an error will be reported. * The default value of alert\_level is ERROR * If the data was received successfully, and the received data matches the expected data, a log message is written with ID ID\_BFM (if this ID has been enabled). * If the received data did not match the expected data, an alert with severity ‘alert\_level’ will be reported.   The procedure will also report alerts for the same conditions as the i2c\_slave\_receive() procedure.  Example:  i2c\_slave\_check(x"3B", “Expect data on I2C”, i2c\_if);  Suggested usage (requires local overload, see section 5):  i2c\_slave-check (C\_CR\_BYTE, “Expecting carriage return”);  i2c\_slave\_check (C\_CR\_BYTE, ERROR, “Expecting carriage return”); |
| **i2c\_master\_quick\_command()** | **i2c\_master\_quick\_command (addr\_value, msg, i2c\_if, [rw\_bit, [exp\_ack, [action\_when\_transfer\_is\_done, [alert\_level, [scope, [msg\_id\_panel, [config]]]]]]])**  The i2c\_master\_quick\_command() procedure transmits a zero-byte message to a slave DUT at address ‘addr\_value’ using the I2C protocol. The I2C Quick Command allows R/W# bit to be either Read(1) or Write(0). The R/W# bit for the command can be set in the ‘rw\_bit’ argument. It is also possible to set the ‘action\_when\_transfer\_is\_done’ to HOLD\_LINE\_AFTER\_TRANSFER in order to allow for restart condition in the next transmission. Since this command can often is used to check if a slave DUT is present on the bus, the ‘exp\_ack’ argument can be set to either true or false depending on whether or not the slave is expected to acknowledge the quick command.  In addition to the specifications listed in the i2c\_master\_transmit() procedure, the following applies:   * The default value of rw\_bit is ‘0’ (Write) * The default value of exp\_ack is true * The default value of alert\_level is ERROR * The default value of action\_when\_transfer\_is\_done is RELEASE\_LINE\_AFTER\_TRANSFER   The procedure reports an alert for the same conditions as the i2c\_master\_transmit procedure. It also reports an error of ‘alert\_level’ severity if ‘exp\_ack’ is false and the DUT acks the quick command or if ‘exp\_ack’ is true and the DUT does not ack the quick command.  Examples:  i2c\_master\_quick\_command(x”AA”, “Pinging I2C slave, expecting ACK”, i2c\_if);  i2c\_master\_quick\_command(x”AA”, “Sending read QC to I2C slave”, i2c\_if, ‘1’, true, HOLD\_LINE\_AFTER\_TRANSFER, ERROR,   C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_BFM\_CONFIG\_DEFAULT);  Suggested usage (requires local overload, see section 5);  i2c\_master\_quick\_command(C\_ADDR\_S1, “Pinging I2C slave, expecting ACK”); |
| **init\_i2c\_if\_signals()** | **init\_i2c\_if\_signals (VOID)**  This function initializes the I2C interface. All the BFM ports are set to high-impedance ('Z').  Example:  i2c\_if <= init\_i2c\_if\_signals(VOID) |

# BFM Configuration record

Type name: t\_i2c\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_I2C\_BFM\_CONFIG\_DEFAULT** | **Description** |
| enable\_10\_bits\_addressing | boolean | FALSE | Turn on/off 10-bits addressing. True: 10-bits addressing enabled. False: 7-bits addressing in use. |
| master\_sda\_to\_scl | time | 20 ns | Time from activation of SDA until activation of SCL. Used for start condition. |
| master\_scl\_to\_sda | time | 20 ns | Last SCL until SDA off. Used for stop condition. |
| master\_stop\_condition\_hold\_time | time | 20 ns | Used in master methods for holding the stop condition. Ensures that the master holds the stop condition for a certain amount of time before the next operation is started. |
| max\_wait\_scl\_change | time | 10 ms | Used when receiving and in slave transmit. |
| max\_wait\_scl\_change\_severity | t\_alert\_level | FAILURE | The above timeout will have this severity. |
| max\_wait\_sda\_change | time | 10 ms | Used when receiving and in slave transmit. |
| max\_wait\_sda\_change\_severity | t\_alert\_level | FAILURE | The above timeout will have this severity. |
| i2c\_bit\_time | time | -1 ns | The bit period. -1 ns will give a TB\_ERROR if not set. |
| i2c\_bit\_time\_severity | t\_alert\_level | FAILURE | A master method will report an alert with this severity if a slave performs clock stretching for longer than i2c\_bit\_time. |
| acknowledge\_severity | t\_alert\_level | FAILURE | An unexpected value for the acknowledge bit will trigger an alert with this severity. |
| slave\_mode\_address | unsigned(9 downto 0) | “0000000000” | The slave methods expect to receive this address from the I2C master DUT. |
| slave\_mode\_address\_severity | t\_alert\_level | FAILURE | The methods will report an alert with this severity if the address format is wrong or the address is not as expected. |
| slave\_rw\_bit\_severity | t\_alert\_level | FAILURE | The methods will report an alert with this severity if the Read/Write bit is not as expected. |
| reserved\_address\_severity | t\_alert\_level | WARNING | The methods will trigger an alert with this severity if the slave address is equal to one of the reserved addresses from the NXP I2C Specification. For a list of reserved addresses, please see the document referred to in section 0. |
| match\_strictness | t\_match\_strictness | MATCH\_EXACT | Matching strictness for std\_logic values in check procedures.  MATCH\_EXACT requires both values to be the same. Note that the expected value  can contain the don’t care operator ‘-‘.  MATCH\_STD allows comparisons between ‘H’ and ‘1’, ‘L’ and ‘0’ and ‘-‘ in both values. |
| id\_for\_bfm | t\_msg\_id | ID\_BFM | The message ID used as a general message ID in the I2C BFM. |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT | The message ID used for logging waits in the I2C BFM. |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL | The message ID used for logging polling in the I2C BFM. |

# Additional Documentation

For additional documentation on the I2C protocol, please see the NXP I2C specification “UM10204 I2C-bus specification and user manual Rev. 6”, available from NXP Semiconductors.

# Compilation

The I2C BFM may only be compiled with VHDL 2008. It is dependent on the UVVM Utility Library (UVVM-Util), which is only compatible with VHDL 2008.

See the separate UVVM-Util documentation for more info. After UVVM-Util has been compiled, the i2c\_bfm\_pkg.vhd BFM can be compiled into any desired library.

See UVVM Essential Mechanisms located in uvvm\_vvc\_framework/doc for information about compile scripts.

## Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see UVVM-Util Quick reference.

# Local BFM overloads

A good approach for better readability and maintainability is to make simple, local overloads for the BFM procedures in the TB process.

This allows calling the BFM procedures with the key parameters only

e.g.

i2c\_master\_transmit(C\_SLAVE\_ADDR, C\_ASCII\_A, “Transmitting ASCII A”);

rather than

i2c\_master\_transmit(C\_SLAVE\_ADDR, C\_ASCII\_A, “Transmitting ASCII A”, i2c\_if, RELEASE\_LINE\_AFTER\_TRANSFER,

C\_SCOPE, shared\_msg\_id\_panel, C\_I2C\_CONFIG\_LOCAL);

By defining the local overload as e.g.:

procedure i2c\_master\_transmit(

constant addr\_value : in unsigned;

constant data\_value : in std\_logic\_vector;

constant msg : in string) is

begin

i2c\_master\_transmit(addr\_value, -- keep as is

data\_value, -- keep as is

msg, -- keep as is

i2c\_if, -- Signals must be visible in local process scope

RELEASE\_LINE\_AFTER\_TRANSFER, -- Shall generate stop condition at the end of every transmit

C\_SCOPE, -- Just use the default

shared\_msg\_id\_panel, -- Use global, shared msg\_id\_panel

C\_I2C\_CONFIG\_LOCAL); -- Use locally defined configuration

end;

Using a local overload like this also allows the following – if wanted:

* Have data value as natural – and convert in the overload
* Set up defaults for constants. May be different for two overloads of the same BFM
* Apply dedicated message ID panel to allow dedicated verbosity control

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

# Local BFM configuration

The I2C BFM requires that a local configuration is declared in the testbench and used in the BFM procedure calls. The default BFM configuration is defined with a i2c bit time of -1 ns so that the BFM can detect and alert the user that the configuration has not been set. See page 2 for the I2C BFM configuration record fields.

Defining a local I2C BFM configuration:

constant C\_I2C\_CONFIG\_LOCAL : t\_i2c\_bfm\_config := (

enable\_10\_bits\_addressing => false,

master\_sda\_to\_scl => 400 ns,

master\_scl\_to\_sda => 505 ns,

master\_stop\_condition\_hold\_time => 505 ns,

max\_wait\_scl\_change => 10 ms,

max\_wait\_scl\_change\_severity => failure,

max\_wait\_sda\_change => 10 ms,

max\_wait\_sda\_change\_severity => failure,

i2c\_bit\_time => 1100 ns,

i2c\_bit\_time\_severity => failure,

acknowledge\_severity => failure,

slave\_mode\_address => C\_I2C\_SLAVE\_DUT\_ADDR,

slave\_mode\_address\_severity => failure,

slave\_rw\_bit\_severity => failure,

reserved\_address\_severity => warning,

match\_strictness => MATCH\_EXACT,

id\_for\_bfm => ID\_BFM,

id\_for\_bfm\_wait => ID\_BFM\_WAIT,

id\_for\_bfm\_poll => ID\_BFM\_POLL

);

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

IMPORTANT   
This is a simplified Bus Functional Model for I2C.  
The given BFM complies with the basic I2C protocol and thus allows a normal access towards an I2C interface. This BFM is not an I2C protocol checker.   
For a more advanced BFM please contact Bitvis AS at support@bitvis.no

Disclaimer: This IP and any part thereof are provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.  
In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with this IP.

**INTELLECTUAL**

**PROPERTY**