**UART BFM** –Quick Reference

**BFM**

**CAUTION**: shaded code/description is preliminary.

|  |
| --- |
| uart\_transmit (data\_value, msg, tx, [config, [scope, [msg\_id\_panel]]]) 1 |
| Example: uart\_transmit(x”AA”, “Sending data to DUT UART instance 1”, tx);  *Suggested usage: uart\_transmit(C\_ASCII\_A, “Transmitting ASCII A to DUT UART instance 1”); -- Suggested usage requires local overload (see section 5)* |

*uart\_bfm\_pkg.vhd*

|  |
| --- |
| uart\_receive (data\_value, msg, rx, terminate\_loop, [config, [scope, [msg\_id\_panel, [proc\_name]]]]) 1 |
| Example: uart\_receive(v\_data\_out, “Receive from DUT UART instance 1”, rx, terminate\_signal);  *Suggested usage: uart\_receive(v\_data\_out, “Receive from DUT UART instance 1”); -- Suggested usage requires local overload (see section 5)* |

|  |
| --- |
| uart\_expect (data\_exp, msg, rx, terminate\_loop, [max\_receptions, [timeout, [alert\_level, [config, [msg\_id\_panel, [scope]]]]]]) 1 |
| Example: uart\_expect(x"3B",1, 0 ns, “Expecting data on UART RX”, rx, terminate\_signal);  *Suggested usage: uart\_expect(C\_CR\_BYTE, C\_TIMEOUT, C\_MAX\_RECEPTIONS, “Expecting carriage return”); -- Suggested usage requires local overload (see section 5)* |

BFM Configuration record ´**t\_uart\_bfm\_config´**

|  |  |  |
| --- | --- | --- |
| **Record element** | **Type** | **C\_UART\_BFM\_CONFIG\_DEFAULT** |
| bit\_time | time | -1 ns |
| num\_data\_bits | natural | 8 |
| idle\_state | std\_logic | ‘1’ |
| num\_stop\_bits | t\_stop\_bits | STOP\_BITS\_ONE |
| parity | t\_parity | PARITY\_ODD |
| timeout | time | 0 ns |
| timeout\_severity | t\_alert\_level | ERROR |
| received\_data\_to\_log\_before\_expected\_data | natural | 10 |
| 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 |
| id\_for\_bfm\_poll\_summary | t\_msg\_id | ID\_BFM\_POLL\_SUMMARY |
| error\_injection | t\_bfm\_error\_injection | C\_ERROR\_INJECTION\_INACTIVE |

Note 1: the BFM configuration has to be defined and used when calling the UART BFM procedures. See section **Feil! Fant ikke referansekilden.** for an example of how to define a local BFM config.

BFM non-signal parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Example(s)** | **Description** |
| data\_value | std\_logic\_vector | x”D3” | The data value to be transmitted to the DUT |
| data\_exp | std\_logic\_vector | x”0D” | The data value to expect when receiving the addressed register. A mismatch results in an alert ‘alert\_level’ |
| max\_receptions | natural | 1 | The maximum number of bytes received before the expected data must be received. Exceeding this limit results in an alert with severity ‘alert\_level’. Default value is 1. |
| timeout | time | 100 ns | The maximum time to pass before the expected data must be received. Exceeding this limit results in an alert with severity ‘alert\_level’. |
| 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. |
| scope | string | "UART BFM" | A string describing the scope from which the log/alert originates. In a simple single sequencer typically "UART BFM". In a verification component typically "UART\_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\_uart\_bfm\_config | C\_UART\_BFM\_CONFIG\_DEFAULT | Configuration of BFM behaviour and restrictions. See section 2 for details. |

BFM signal parameters

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| terminate\_loop | std\_logic | External control of loop termination to e.g. stop expect procedure prematurely |
| tx | std\_logic | The UART BFM transmission signal. Must be connected to the UART DUT 'rx' port. |
| rx | std\_logic | The UART BFM reception signal. Must be connected to the UART DUT 'tx' port. |

Note: All signals are active high.

BFM Error injection record (inside the BFM configuration record)

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Type** | **Default value** | **Description** |
| parity\_bit\_error | Boolean | False | Will invert the parity bit in a transmission if TRUE, and thus generate a parity error. |
| stop\_bit\_error | Boolean | False | Will invert the first stop bit in a transmission if TRUE. Note that the following UART frame may be misinterpreted if there is no Idle period or additional stop bits after the error injection. Hence a stop\_bit\_error may lead to multiple following UART frame errors. |

Error injection in general is explained in ‘UVVM Essential Mechanisms’ located in uvvm\_vvc\_framework/doc.  
Note: The error\_injection\_config in the VVC config will override any error injection specified in the BFM config when using VVCs.

BFM details

# BFM procedure details and examples

|  |  |
| --- | --- |
| **Procedure** | **Description** |
| **uart\_transmit()** | **uart\_transmit (data\_value, msg, tx, [config, [scope, [msg\_id\_panel]]])**  The uart\_transmit() procedure transmits the data in ‘data\_value’ to the DUT, using the UART protocol. For protocol details, see the UART specification.   * The start bit, stop bit, parity, number of stop bits and number of data bits per transmission is defined in the ‘config’ parameter. * The default value of scope is C\_SCOPE (“UART BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_UART\_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.   Errors may be injected – depending on the error\_injection\_config sub-record within the bfm\_config    Examples:  uart\_transmit(x”AA”, “Transmitting data to DUT UART instance 1”, tx);  uart\_transmit(x”AA”, “Transmitting data to DUT UART instance 1”, tx, C\_UART\_BFM\_CONFIG\_DEFAULT, C\_SCOPE, shared\_msg\_id\_panel);  Suggested usage (requires local overload, see section 5):  uart\_transmit(C\_ASCII\_A, “Transmitting ASCII A to DUT UART instance 1”); |
| **uart\_receive()** | **uart\_receive (data\_value, msg, rx, terminate\_loop, [config, [scope, [msg\_id\_panel, [proc\_name]]]])**  The uart\_receive() procedure receives data from the DUT at the given address, using the UART protocol. For protocol details, see the UART specification. When called, the uart\_receive procedure will wait for the start bit to be present on the rx line. The initial wait for the start bit will be terminated if one of the following occurs:   1. The start bit is present on the rx line. 2. The terminate\_loop flag is set to ‘1’. 3. The number of clock cycles waited for the start bit exceeds ‘config.max\_wait\_cycles’ clock cycles.   Once all the bits have been received according to the UART specification, the parity and stop bit are checked. If correct, the read data is placed on the output ‘data\_value’ and the procedure returns.   * The default value of scope is C\_SCOPE (“UART BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_UART\_BFM\_CONFIG\_DEFAULT, see table on the first page. * The default value of proc\_name is “uart\_receive”. This argument is intended to be used internally, when procedure is called by uart\_expect(). * 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 reports an alert if:   * timeout occurs, i.e. start bit does not occur within ‘config.max\_wait\_cycles’ clock cycles (alert level: ‘config.max\_wait\_cycles\_severity’) * terminate\_loop is set to ‘1’ (alert level: WARNING) * expected stop\_bit does not match received stop bit(s) (alert level: ERROR) * Calculated parity ‘config.parity’ does not match received parity (alert level: ERROR)   Examples:  uart\_receive(v\_data\_out, “Receive from DUT UART instance 1”, clk, terminate\_signal);  uart\_receive(v\_data\_out, “Receive from DUT UART instance 1”, clk, terminate\_signal, C\_UART\_BFM\_CONFIG\_DEFAULT, C\_SCOPE,   shared\_msg\_id\_panel);  Suggested usage (requires local overload, see section 5):  uart\_receive(v\_data\_out, “Receive from DUT UART instance 1”); |
| **uart\_expect()** | **uart\_expect (data\_exp, msg, rx, terminate\_loop, [max\_receptions, [timeout, [alert\_level, [config, [msg\_id\_panel, [scope]]]]]])**  The uart\_expect() procedure receives data from the DUT on the BFM rx line, using the receive procedure as described in the uart\_receive() procedure. After receiving data from the UART rx line, the data is compared with the expected data, ‘data\_exp’. If the received data does not match the expected data, another uart\_receive() procedure will be initiated. This process will repeat until one of the following occurs:   1. The received data matches the expected data. 2. A timeout occurs. 3. The process has repeated ‘max\_receptions’ number of times. 4. The ‘terminate\_loop’ signal is set to ‘1’.  * The default value of alert\_level is ERROR * The default value of scope is C\_SCOPE (“UART BFM”) * The default value of msg\_id\_panel is shared\_msg\_id\_panel, defined in UVVM\_Util. * The default value of config is C\_UART\_BFM\_CONFIG\_DEFAULT, see table on the first page. * The default value of max\_receptions is 1. * A log message with ID ID\_BFM is issued when the uart\_expect procedure starts * 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.   This procedure reports an alert if:   * ‘max\_receptions’ and ‘timeout’ are set to 0, which will result in a possible infinite loop (alert\_level: ERROR) * the expected data is not received within the time set in ‘timeout’ (alert\_level: ‘alert\_level’) * the expected data is not received within the number of received packets set in ‘max\_receptions’ (alert\_level: ‘alert\_level’) * ‘terminate\_loop’ is set to ‘1’ (alert\_level: WARNING)   The procedure will also report alerts for the same conditions as the uart\_receive() procedure.  Example:  uart\_expect(x"3B", “Expect data on UART RX”, rx, terminate\_signal, 1, 0 ns);  Suggested usage (requires local overload, see section 5):  uart\_expect(C\_CR\_BYTE, “Expecting carriage return”);  uart\_expect(C\_CR\_BYTE, “Expecting carriage return”, C\_TIMEOUT, C\_MAX\_RECEPTIONS); |

# BFM Configuration record

Type name: t\_uart\_bfm\_config

|  |  |  |  |
| --- | --- | --- | --- |
| **Record element** | **Type** | **C\_UART\_BFM\_CONFIG\_DEFAULT** | **Description** |
| bit\_time | time | -1 ns | The time it takes to transfer one bit. Will raise an error if not set. |
| num\_data\_bits | natural | 8 | Number of data bits to send per transmission |
| idle\_state | std\_logic | ‘1’ | Bit value when line is idle |
| num\_stop\_bits | t\_stop\_bits | STOP\_BITS\_ONE | Number of stop-bits to use per transmission {STOP\_BITS\_ONE, STOP\_BITS\_ONE\_AND\_HALF, STOP\_BITS\_TWO} |
| parity | t\_parity | PARITY\_ODD | Transmission parity bit {PARITY\_NONE, PARITY\_ODD, PARITY\_EVEN} |
| timeout | time | 0 ns | The maximum time to wait for the UART start bit on the RX line before timeout |
| timeout\_severity | t\_alert\_level | error | The above timeout will have this severity |
| received\_data\_to\_log\_before\_expected\_data | natural | 10 | Maximum number of bytes to save ahead of the expected data in the receive buffer. The bytes in the receive buffer will be logged. |
| 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 UART BFM |
| id\_for\_bfm\_wait | t\_msg\_id | ID\_BFM\_WAIT | The message ID used for logging waits in the UART BFM |
| id\_for\_bfm\_poll | t\_msg\_id | ID\_BFM\_POLL | The message ID used for logging polling in the UART BFM |
| id\_for\_bfm\_poll\_summary | t\_msg\_id | ID\_BFM\_POLL\_SUMMARY | The message ID used for logging polling summary in the UART BFM |
| error\_injection | t\_bfm\_error\_injection | C\_ERROR\_INJECTION\_INACTIVE | See error injection record on page 2. Error injection in general is explained in ‘UVVM Essential Mechanisms’ located in uvvm\_vvc\_framework/doc. |
|  |  |  |  |

# Additional Documentation

For additional documentation on the UART protocol, please see the UART specification.

# Compilation

The UART 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 uart\_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.

uart\_transmit(C\_ASCII\_A, “Transmitting ASCII A”);

rather than

uart\_transmit(C\_ASCII\_A, “Transmitting ASCII A”, tx, C\_UART\_CONFIG\_LOCAL, C\_SCOPE, shared\_msg\_id\_panel);

By defining the local overload as e.g.:

procedure uart\_transmit(

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

constant msg : in string) is

begin

uart\_transmit(data\_value, -- keep as is

msg, -- keep as is

tx, -- Signals must be visible in local process scope

C\_UART\_CONFIG\_LOCAL, -- Use locally defined configuration

C\_SCOPE, -- Just use the default

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

end;

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

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

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

# Local BFM configuration

The UART 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 clock period of -1 ns so that the BFM can detect and alert the user that the configuration has not been set. See page 1 for the UART BFM configuration record fields.

Defining a local UART BFM configuration:

constant C\_UART\_CONFIG\_LOCAL : t\_uart\_bfm\_config := (

bit\_time => C\_UART\_BIT\_TIME,

num\_data\_bits => 8,

idle\_state => '1',

num\_stop\_bits => STOP\_BITS\_ONE,

parity => PARITY\_ODD,

timeout => 0 ns,

timeout\_severity => error,

num\_bytes\_to\_log\_before\_expected\_data => 10,

match\_strictness => MATCH\_EXACT,

id\_for\_bfm => ID\_BFM,

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

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

id\_for\_bfm\_poll\_summary => ID\_BFM\_POLL\_SUMMARY,

error\_injection => C\_BFM\_ERROR\_INJECTION\_INACTIVE

);

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 UART TX and RX.  
The given BFM complies with the basic UART protocol and thus allows a normal access towards a UART interface. This BFM is not a UART 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**