I2C BFM – Quick Reference

I2C Master (see page 2 for I2C Slave)

BFM

i2c_bfm_pkg.vhd

i2c_master_transmit (addr_value, data, msg, i2c_if, [action_when_transfer_is_done, [scope, [msg_id_panel, [config]]]])

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_receive (addr_value, data, msg, i2c_if, [action_when_transfer_is_done, [scope, [msg_id_panel, [config, [proc_name]]]]])

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]]]]])

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]]]]]])

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,

init_i2c_if_signals (VOID)

Example: i2c_if <= init_i2c_if_signals(VOID);

UVVM™

VHDL 2008 only

I2C BFM – Quick Reference

I2C Slave (see page 1 for I2C Master)

i2c_slave_transmit (data, msg, i2c_if, [scope, [msg_id_panel, [config]]])

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)*

i2c_slave_receive (data, msg, i2c_if, [scope, [msg_id_panel, [config, [proc_name]]]])

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_slave_check (data_exp, msg, i2c_if, [exp_rw_bit, [alert_level, [scope, [msg_id_panel, [config]]]]])

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)*

BFM

i2c_bfm_pkg.vh

UVVM™

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
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

Name	Type	Description
i2c_if	t_i2c_if	See table "Signal record 'i2c_if'"

Signal record ´t_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.

1 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)
```

2 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 3.
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.

3 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.

4 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.

4.1 Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see UVVM-Util Quick reference.

5 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_BFM_CONFIG_DEFAULT);
```

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 or C_I2C_CONFIG_DEFAULT


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

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