GPIO BFM – Quick Reference

BFM

gpio_bfm_pkg.vhd

gpio_set (data_value, msg, data_port, [scope, [msg_id_panel, [config]]])

Example: gpio_set(C_BAUD_RATE, “Setting Baudrate to 9600”, data_port, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_set(C_BAUD_RATE, “Setting Baudrate to 9600”, data_port);

gpio_get (data_value, msg, data_port, [scope, [msg_id_panel, [config]]])

Example: gpio_get(v_baudrate, “Read baudrate”, data_port, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_get(v_baudrate, “Read baudrate”, data_port);

gpio_check (data_exp, msg, data_port, [alert_level, [scope, [msg_id_panel, [config]]]])

Example: gpio_check(x”3B”, “Check data from UART RX”, data_port, ERROR, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_check(x”3B”, “Check data from UART RX”, data_port);

gpio_check_stable (data_exp, stable_req, msg, data_port, [alert_level, [scope, [msg_id_panel, [config]]]])

Example: gpio_check_stable(x”3B”, 100 us, “Check data from UART RX has been stable for 100 us”, data_port, ERROR, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_check_stable(x”3B”, 100 us, “Check data from UART RX has been stable for 100 us”, data_port);

gpio_expect (data_exp, msg, data_port, [timeout, [alert_level, [scope, [msg_id_panel, [config]]]]])

Example: gpio_expect(x”0D”, “Read UART RX until CR is found”, data_port, 10 ms, ERROR, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_expect(x”0D”, “Read UART RX until CR is found”, data_port);

gpio_expect_stable (data_exp, stable_req, stable_req_from, msg, data_port, [timeout, [alert_level, [scope, [msg_id_panel, [config]]]]])

Example: gpio_expect_stable(x”0D”, 100 us, FROM_NOW, “Read UART RX until CR is found and check it remains stable for 100 us”, data_port, 10 ms, ERROR, C_SCOPE, shared_msg_id_panel, gpio_bfm_config);
Example: gpio_expect_stable(x”0D”, 100 us, FROM_LAST_EVENT, “Read UART RX and check it has been stable for 100 us since the last event”, data_port);

UVVM™

VHDL 2008 only

Copyright © 2020 by Bitvis AS. All rights reserved.

BFM Configuration record ´**t_gpio_bfm_config**´

Record element	Type	C_GPIO_BFM_CONFIG_DEFAULT
clock_period	time	-1 ns
match_strictness	t_match_strictness	MATCH_STD
id_for_bfm	t_msg_id	ID_BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT
timeout	time	-1 ns

BFM non-signal parameters

Name	Type	Example(s)	Description
data_value	std_logic_vector	x"D3"	The data value to be written to the register.
data_exp	std_logic_vector	x"0D" or C_UART_CR	The data value expected when reading the register. A mismatch results in an alert 'alert_level'.
stable_req	time	1 ms	The time that the expected data value should remain stable in the register.
stable_req_from	t_from_point_in_time	FROM NOW or FROM_LAST_EVENT	The point in time where stable_req starts.
timeout	time	10 ms or C_CLK_PERIOD	The maximum time to pass before the expected data must be found. A timeout result in an alert 'alert_level'.
alert_level	string	ERROR or TB_WARNING	Set the severity for the alert that may be asserted by the method.
msg	string	"Set baudrate to 1MHz"	A custom message to be appended in the log/alert.
scope	string	"GPIO_BFM" or C_SCOPE	A string describing the scope from which the log/alert originates. In a simple single sequencer typically "GPIO_BFM". In a verification component, typically "GPIO_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_gpio_bfm_config	C_GPIO_BFM_CONFIG_DEFAULT	Configuration of BFM behaviour and restrictions. See section 0 for details.

1 BFM procedure details and examples

Procedure	Description

gpio_set()	**gpio_set (data_value, msg, data_port, [scope, [msg_id_panel, [config]]])**
	
	The gpio_set() procedure will write the given data in 'data_value' to the DUT.
	When called, the gpio_set() procedure will write to the DUT register immediately, except bits set to "don't care" ('-').
	
	- The default value of scope is C_SCOPE ("GPIO BFM")
	- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
	- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
	- A log message is written if ID_BFM is enabled for the specified message ID panel.
	- `Data_value` is normalised to `data_port` direction.
	
	Example:
	`gpio_set(C_BAUDRATE_9600, "Set baudrate to 9600", data_port, C_SCOPE, shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);`
	Suggested usage (requires local overload, see section 4):
	`gpio_set(C_BAUDRATE_9600, "Set baudrate to 9600", data_port);`

gpio_get()	**gpio_get (data_value, msg, data_port, [scope, [msg_id_panel, [config]]])**
	
	The gpio_get() procedure reads the DUT register and returns it in the 'data_value' parameter.
	
	- The default value of scope is C_SCOPE ("GPIO BFM")
	- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
	- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
	- A log message is written if ID_BFM is enabled for the specified message ID panel.
	
	Example:
	`gpio_get(v_baudrate, "Read baudrate", data_port, C_SCOPE, shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);`
	Suggested usage (requires local overload, see section 4):
	`gpio_get(v_baudrate, "Read baudrate", data_port);`

gpio_check()	**gpio_check (data_exp, msg, data_port, [alert_level, [scope, [msg_id_panel, [config]]]])** The gpio_check() procedure reads the DUT register and compares the data with the expected data in 'data_exp'. If the DUT data does not match 'data_exp', an alert with severity 'alert_level' will be triggered. If the DUT data matches 'data_exp', a message with ID config.id_for_bfm will be logged. - The default value of alert_level is ERROR. - The default value of scope is C_SCOPE ("GPIO BFM") - The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.

- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
- A log message is written if ID_BFM is enabled for the specified message ID panel.
- Data_exp is normalised to data_port direction.

Example:
```
gpio_check(x"3B", "Check data from UART RX", data_port, ERROR, C_SCOPE, shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);
```
Suggested usage (requires local overload, see section 4):
```
gpio_check(x"3B", "Check data from UART RX", data_port);
```

gpio_check_stable()

gpio_check_stable (data_exp, stable_req, msg, data_port, [alert_level, [scope, [msg_id_panel, [config]]]])

The gpio_check_stable() procedure reads the DUT register and compares the data with the expected data in 'data_exp', it also checks that the DUT register has been stable for the 'stable_req' time (see section 1.1). If the DUT data does not match 'data_exp' or is not stable, an alert with severity 'alert_level' will be triggered. If the DUT data matches 'data_exp' and is stable, a message with ID config.id_for_bfm will be logged.

- The default value of alert_level is ERROR.
- The default value of scope is C_SCOPE ("GPIO BFM")
- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
- A log message is written if ID_BFM is enabled for the specified message ID panel.
- Data_exp is normalised to data_port direction.

Example:
```
gpio_check_stable(x"3B", 100 us, "Check data from UART RX has been stable for 100 us", data_port, ERROR, C_SCOPE,
                  shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);
```
Suggested usage (requires local overload, see section 4):
```
gpio_check_stable(x"3B", 100 us, "Check data from UART RX has been stable for 100 us", data_port);
```

gpio_expect()

gpio_expect (data_exp, msg, data_port, [timeout, [alert_level, [scope, [msg_id_panel, [config]]]]])

The gpio_expect() procedure reads a register until the expected data, 'data_exp', is matched or until a timeout value is reached.
If the received data does not match 'data_exp' within the timeout delay, an alert with severity 'alert_level' will be triggered. If the DUT data matches 'data_exp', a message with ID config.id_for_bfm will be logged.

- The default timeout is -1 ns.
- The default value of alert_level is ERROR.
- The default value of scope is C_SCOPE ("GPIO BFM")
- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
- A log message is written if ID_BFM ID is enabled for the specified message ID panel.
- Data_exp is normalised to data_port direction.

Example:
```
gpio_expect(x"0B", "Read UART RX until CR is found", data_port, 10 ms, ERROR, C_SCOPE, shared_msg_id_panel,
            C_GPIO_BFM_CONFIG_DEFAULT);
```

Suggested usage (requires local overload, see section 4):

```
gpio_expect(x"0B", "Read UART RX until CR is found", data_port, 10 ms);
```

gpio_expect_stable()

gpio_expect_stable (data_exp, stable_req, stable_req_from, msg, data_port, [timeout, [alert_level, [scope, [msg_id_panel, [config]]]]])

The gpio_expect_stable() procedure reads a register until the expected data, 'data_exp', is matched or until a timeout value is reached. It also checks that the register remains stable for the 'stable_req' time, sampled after the 'stable_req_from' point in time (see section 1.1).
If the received data does not match 'data_exp' within the timeout delay or it doesn't remain stable, an alert with severity 'alert_level' will be triggered. If the DUT data matches 'data_exp' and is stable, a message with ID config.id_for_bfm will be logged.

- The default timeout is -1 ns.
- The default value of alert_level is ERROR.
- The default value of scope is C_SCOPE ("GPIO BFM")
- The default value of msg_id_panel is shared_msg_id_panel, defined in UVVM_Util.
- The default value of config is C_GPIO_BFM_CONFIG_DEFAULT.
- A log message is written if ID_BFM ID is enabled for the specified message ID panel.
- `Data_exp` is normalised to `data_port` direction.

Example:

```
gpio_expect_stable(x"0B", 100 us, FROM_NOW, "Read UART RX until CR is found and check it remains stable for 100 us",
                   data_port, 10 ms, ERROR, C_SCOPE, shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);
```

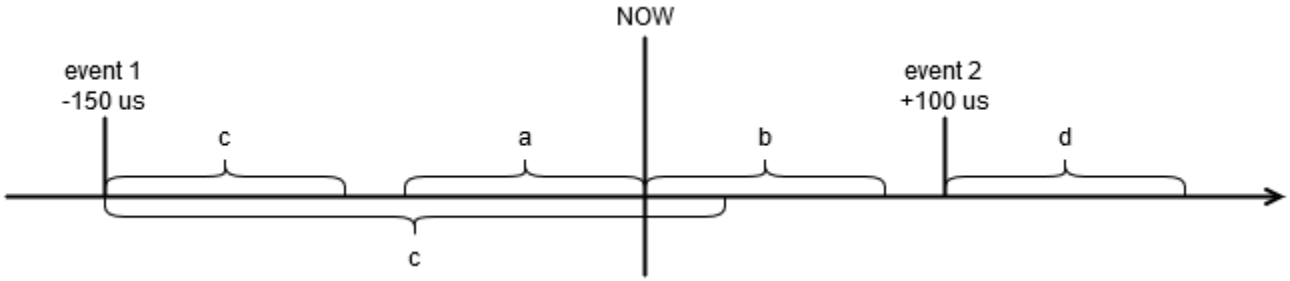
Suggested usage (requires local overload, see section 4):

```
gpio_expect_stable(x"0B", 100 us, FROM_LAST_EVENT, "Read UART RX and check it has been stable for 100 us since the last
                   event", data_port);
```

1.1 Checking stability

The procedures gpio_check_stable and gpio_expect_stable can check if the DUT data port is stable for a certain time. There are different scenarios where we could check stability:

a. To check that data has been stable for a certain time before now, use gpio_check_stable().
b. To check that data, which is already same as expected, remains stable for a certain time from now, use gpio_expect_stable(FROM_NOW).
c. To check that data, which is already same as expected, remains stable for a certain time from the last change, use gpio_expect_stable(FROM_LAST_EVENT).
d. To check that data remains stable after it is equal than expected, use gpio_expect_stable(FROM_NOW). Note that in this case the 'stable_req_from' parameter does not have any influence since the event has not occurred.



2 BFM Configuration record

Type name: t_gpio_bfm_config

Record element	Type	C_GPIO_BFM_CONFIG_DEFAULT	Description
clock_period	time	-1 ns	Specifies the clock period
match_strictness	t_match_strictness	MATCH_STD	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 GPIO BFM
id_for_bfm_wait	t_msg_id	ID_BFM_WAIT	The message ID used for logging waits in the GPIO BFM
timeout	time	-1 ns	Timeout value for the expect procedures. This is only used if no timeout parameter is given in the procedures.

3 Compilation

The GPIO 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 gpio_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.

3.1 Simulator compatibility and setup

See README.md for a list of supported simulators.
For required simulator setup see UVVM-Util Quick reference.

4 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.

```
gpio_expect(x"F5", "Read UART RX until 0xF5 is found", data_port, 2 ms);
```

rather than

```
gpio_expect(x"F5", "Read UART RX until 0xF5 is found", data_port, 2 ms, ERROR,
            C_SCOPE, shared_msg_id_panel, C_GPIO_BFM_CONFIG_DEFAULT);
```

By defining the local overload as e.g.:

```
procedure gpio_check(
   constant data_exp     : in std_logic_vector;
   constant msg          : in string;
   constant data_port    : in std_logic_vector;
   constant timeout      : in time) is
begin
   gpio_check(data_exp,             -- keep as is
              msg,                  -- keep as is
              data_port,            -- keep as is
              timeout,              -- keep as is
              error,                -- Just use the default
              C_SCOPE,              -- Just use the default
              shared_msg_id_panel,  -- Use global, shared msg id panel
              C_GPIO_CONFIG_LOCAL); -- Use locally defined configuration or C_GPIO_BFM_CONFIG_DEFAULT

end;
```

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

- 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

INTELLECTUAL PROPERTY

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.