# 

The Error Injection VIP consists of two VHDL entities - a std\_logic entity and a std\_logic\_vector entity, and is used for inducing non protocol-dependent errors on single and vector signals. See full introduction to functionality on page 2.

Error Injection – Generics

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic element** | **Type** | **Default** | **Description** |
| GC\_START\_TIME | time | 0 ns | The error injector will wait the specified time after initialization before starting to monitor signal events, and injecting error if configured to. Note that default is 0 ns, i.e. not waiting after initialization. |
| GC\_INSTANCE\_IDX | natural | 1 | Error injection instance number. |
|  |  |  |  |

Error Injection – Configuration record ´**t\_error\_injection\_config´**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record element** | **Type** | **Example** | **C\_EI\_CONFIG\_DEFAULT** | **Description** |
| error\_type | t\_error\_injection | JITTER | BYPASS | Type of error to be injected to signal. No error is injected when error\_type is set to BYPASS. **1** |
| initial\_delay\_min | time | 5 ns | 0 ns | Error injection start relative to initial signal event. 1 |
| initial\_delay\_max | time | 0 ns | 0 ns | Setting the max parameter will generate a randomized initial timing parameter.2 |
| return\_delay\_min | time | 3 ns | 0 ns | Error injection end relative to next signal event. 1, 4 |
| return\_delay\_max | time | 0 ns | 0 ns | Setting the max parameter will generate a randomized return timing parameter.2, 4 |
| width\_min | time | 0 ns | 0 ns | The width of an error injected pulse if error type PULSE is selected. |
| width\_max | time | 0 ns | 0 ns | Setting the max parameter will generate a randomized timing parameter.2 |
| interval | positive | 3 | 1 | Errors will be injected in this interval, e.g. 1 for every signal event, 2 for every second etc.3, 5  Interval = 1: SL will experience error injection on every second signal event, e.g. every rising edge.  SLV will experience error injection on every signal event.  Interval = 2: SL will experience error injection on every fourth signal event, e.g. every second rising edge.  SLV will experience error injection on every second signal event. |
| base\_value | std\_logic | ‘1’ | ‘0’ | The initial edge in an SL error injection will start on the transition from base\_value to another value, e.g. from ‘0’ to ‘1’. The return edge in an SL error injection will be the transition back to base\_value. 6 Note that setting base\_value = ‘-‘will start SL error injection on first upcoming edge after configuration, regardless of input value |
| randomization\_seed1 | positive | 12 | 1 | Global seed 1 for randomized timing parameters. |
| randomization\_seed2 | positive | 14 | 2 | Global seed 2 for randomized timing parameters. |

Note 1: see Error injection types parameters on page 2 for required configuration parameters for each error type.  
Note 2: randomization is selected by setting the max parameter higher than the min parameter, and will be in the range of min parameter to max parameter.

Note 3: SL signal is handled with respect to a start and an end transition, and an error injection is activated at the start transition.

vector signal is handled with respect to a start transition, and the following transition will be treated as a new start transition.

Note 4: vector signal does not have a return\_delay\_min/max parameter 3

Note 5: error injection interval will always start injection on the first signal event for any interval setting, followed by the configured interval, e.g. every second for interval = 2.

Note 6: for initial and return edge definitions, see BYPASS example on page 3.

**Error Injection VIP -** Quick Reference  
UVVM Verification IP

Error Injection Types Parameters – Configure error injection type ´**t\_error\_injection\_types´**

Parameters marked with **R** is required, **O** is optional and **X** is ignored.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Error** | **Intial\_delay\_min** | **Intial\_delay\_max** | **Return\_delay\_min** | **Return\_delay\_max** | **Width\_min** | **Width\_max** |
| BYPASS | X | X | X | X | X | X |
| PULSE | **R** | **O** | X | X | **R** | **O** |
| DELAY | **R** | **O** | X | X | X | X |
| JITTER | **R** | **O** | **R** | **O** | X | X |
| INVERT | X | X | X | X | X | X |
| STUCK\_AT\_OLD | X | X | X | X | **R** | **O** |
| STUCK\_AT\_NEW | X | X | X | X | **R** | **O** |
|  |  |  |  |  |  |  |

**Note** that a randomized error injection is selected by setting optional parameter “\_max” ≠ 0 ns, and by setting “\_max” > “\_min”. The randomisation interval will be in the range of required parameter time to optional parameter time.

Error injection functional details

Note 1: initial event is when a SL signal is at base\_value (see Configuration Record details on page 1) and changes signal level. The return event is when a SL signal is returning to its base\_value. Note that vector signals only have initial events. See Figure in BYPASS example on page 3.

|  |  |  |
| --- | --- | --- |
| **Error** | **Std\_logic** | **Std\_logic\_vector** |
| BYPASS | Signal is preserved and no error is injected. | Signal is preserved and no error is injected. |
| PULSE | Signal will experience a pulse of width **width** after a time of **initial\_delay**. 1 Note that the pulse value will be the value prior to initial event. | As for std\_logic.  Note that the pulse value will be the value prior to initial event. |
| DELAY | Signal will be skewed for a time of **initial\_delay** on initial edge and on return edge. 1 Any signal activity while error injection is active will override the error injection. | As for std\_logic. . |
| JITTER | As for DELAY, but with different values on edge delays. Note that only positive jitter is supported, and that reordering of positive and negative edges yield an invalid configuration and will not work. | Not applicable for vector signals. |
| INVERT | The signal is inverted as long as the error injection configuration is set to INVERT. Note that when error injection configuration is changed from INVERT, a new signal event is required for the new configuration to take effect. | As for std\_logic. |
| STUCK\_AT\_OLD | The signal will be stuck at the value it had prior to initial signal event, and released to the current signal value after the specified **width\_min** time. 1 Note that any signal event during STUCK\_AT\_OLD is ignored. | As for std\_logic. |
| STUCK\_AT\_NEW | The signal will be stuck at the value it had prior to initial signal event, and released to the current signal value after a specified **width\_min** time. 1 Note that any signal event during STUCK\_AT\_NEW is ignored. | As for std\_logic |

Error Injection examples for std\_logic and std\_logic\_vector signals

The section “Error injection type parameters” on page 2 show how error types are configured. Below follows a series of examples showing error configuration of a std\_logic and a std\_logic\_vector type signal, respectively.

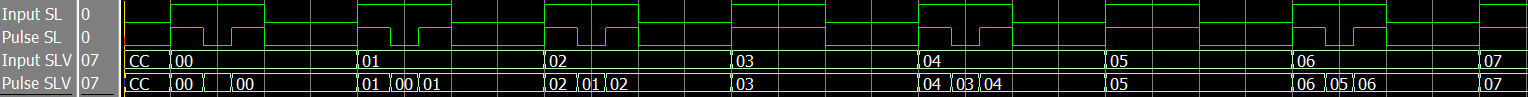
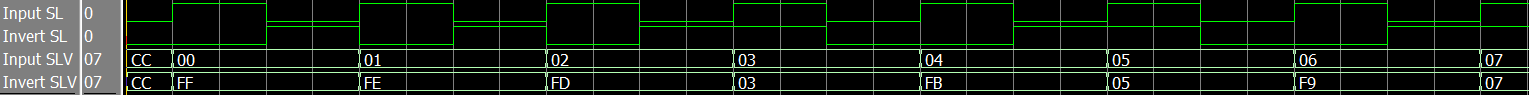
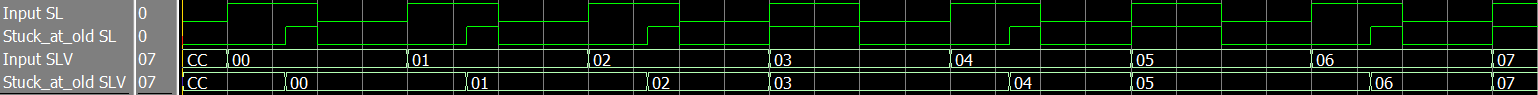
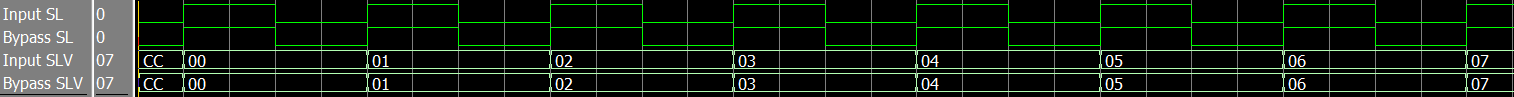
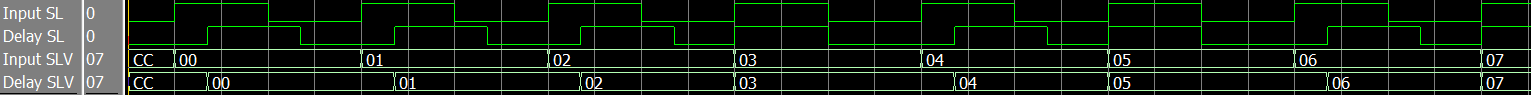
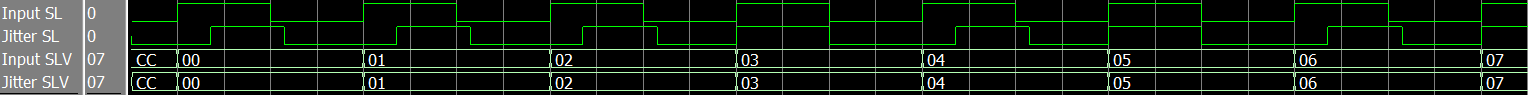
**Note** in these examples. the std\_logic\_signal is active high for 20 ns and active low for 20 ns, while the std\_logic\_vector signal is updated to a new value every 40 ns.  
**Note** that JITTER is an invalid error injection type for vector signals and that the signal is not error injected.

↓return event

initial event ↓

**BYPASS**

shared\_ei\_config(C\_EI\_1):= C\_EI\_CONFIG\_DEFAULT;



↓ interval=2

↓ interval=1

↓ interval=2

↓ interval=1

initial event ↑

initial event ↑

**JITTER**

shared\_ei\_config(C\_EI\_1).error\_type := JITTER;

shared\_ei\_config(C\_EI\_1).initial\_delay\_min := 7 ns;

shared\_ei\_config(C\_EI\_1).return\_delay := 3 ns;

**DELAY**

shared\_ei\_config(C\_EI\_1).error\_type := DELAY;

shared\_ei\_config(C\_EI\_1).initial\_delay\_min := 7 ns;

↓ interval=2

↓ interval=1

↓ interval=2

↓ interval=1

↓ interval=2

↓ interval=1

**STUCK\_AT\_OLD**

shared\_ei\_config(C\_EI\_1).error\_type := STUCK\_AT\_OLD;

shared\_ei\_config(C\_EI\_1).width\_min := 13 ns;

**INVERT**

shared\_ei\_config(C\_EI\_1).error\_type := INVERT;

**PULSE**

shared\_ei\_config(C\_EI\_1).error\_type := PULSE;

shared\_ei\_config(C\_EI\_1).initial\_delay\_min := 7 ns;

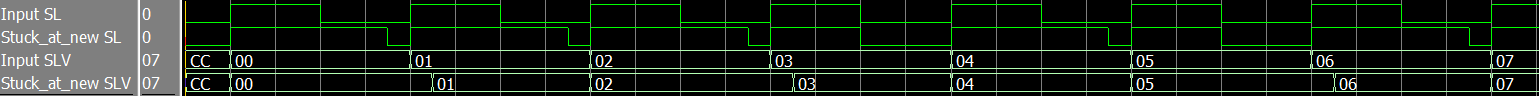
shared\_ei\_config(C\_EI\_1).width\_min := 6 ns;

**STUCK\_AT\_NEW**

shared\_ei\_config(C\_EI\_1).error\_type := STUCK\_AT\_NEW;

shared\_ei\_config(C\_EI\_1).width\_min := 35 ns; -- SL

shared\_ei\_config(C\_EI\_1).width\_min := 45 ns; -- SLV



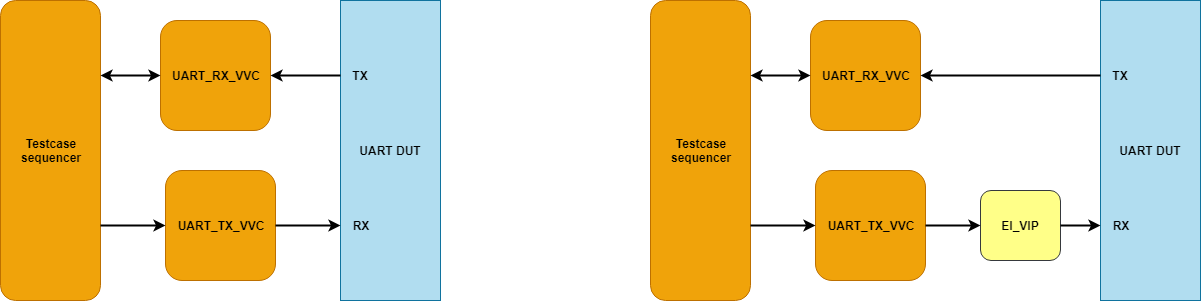
↓ interval=2

↓ interval=1

Error Injection VIP details

The error injection VIP can be used as a single instance or with multiple instances, and can be used with std\_logic or vector signals. 1 A generic GC\_WIDTH is needed for vector signals to constrain the vector width in the error injector port map. The default error injector behaviour is in BYPASS mode, i.e. without any error injected to the signal. The error injection is controlled using a dedicated shared variable array, shared\_ei\_config(index), of type t\_error\_injection\_config, where each error injector has its own dedicated array index. Error injection is started by setting the error injection VIPs index in the shared\_ei\_config array as listed in the Error Injection configuration record table on page 1 and Error Injection Types Parameters on page 2. The error injection is disabled and set to bypass mode by setting the configuration to C\_EI\_CONFIG\_DEFAULT.

The provided example below shows how a typical UART error injection setup could be done, using a error\_injection\_sl entity for the UART RX signal line.



Note 1: std\_logic signals require the error\_injection\_sl entity, while vector signals require the error\_injection\_slv entity.

# Example usage

The error injection VIP comes with two entities, a std\_logic version for single signals and a std\_logic\_vector version for vector signals. Setting up an error injector VIP is fast and only has a few mandatory steps:

* Include the error injection package.
* Define the needed error injection configuration signal(s).
* Instantiate the error injector(s) needed.
* Set the error injection configurations.

**In testbench**:

Include the Bitvis VIP Error Injection package

library bitvis\_vip\_error\_injection;

use bitvis\_vip\_error\_injection.error\_injection\_pkg.all;

...

Define the Error Injection instance index.

constant C\_UART\_TX\_EI : natural := 1;

...

Uart\_tx\_error\_injector: entity work.error\_injection\_sl

Instantiate the Error Injector VIP(s).

Note that component instances normally would be located in the test harness and not in the testbench.

generic map (

GC\_START\_TIME => 10 ns,

GC\_INSTANCE\_IDX => C\_UART\_TX\_EI

)

port map (

ei\_in => uart\_tx,

ei\_out => ei\_uart\_tx

);

....

Use the UART VVC to send data to DUT.

Configure the Error Injector VIP using the shared\_ei\_config from the error\_injection\_pkg.

Set the Error Injection config back to signal BYPASS (default) when done.

uart\_transmit(UART\_VVCT, 1, TX, x”AA”, “Transmitting data”);

shared\_ei\_config(C\_UART\_TX\_EI).error\_type := DELAY;

shared\_ei\_config(C\_UART\_TX\_EI).delay\_min := 2 ns;

shared\_ei\_config(C\_UART\_TX\_EI).base\_value := ‘0’;

wait for 10 ns;

shared\_ei\_config(C\_UART\_TX\_EI) := C\_EI\_CONFIG\_DEFAULT;

For more detailed examples see VHDL example ei\_demo\_tb.vhd, located in the tb folder.

# Additional documentation

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

# Compilation

The Error Injection VIP must be compiled with VHDL 2008.

It is dependent on the following libraries

* ***UVVM Utility Library (UVVM-Util), version 2.5.0 and up***

**Compile order for the Error Injection VIP:**

|  |  |  |
| --- | --- | --- |
| **Compile to library** | **File** | **Comment** |
| bitvis\_vip\_error\_injection | error\_injection\_pkg. | Configuration declaration. |
| bitvis\_vip\_error\_injection | error\_injection\_slv | Vector entity and error injection functionality. |
| bitvis\_vip\_error\_injection | error\_injection\_sl | Std\_logic entity, only neede when using error injection with std\_logic signals. |

# Simulator compatibility and setup

See README.md for a list of supported simulators.

For required simulator setup see UVVM-Util Quick reference.

**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.