

How to activate Specman Profiler? How to get rid of automatic vr_ad coverage? Let's find out.

How to activate Specman Profiler?

```
{code}extend sys {
```

```
setup() is also {
```

```
specman("set profile");
```

```
};
```

```
};
```

```
{/code}
```

How to get rid of automatic vr_ad coverage?

```
{code}extend sys {
```

```
setup() is also {
```

```
set_cover("vr_ad_reg", FALSE);
```

```
};
```

```
};
```

```
{/code}
```

How to continue simulation regardless of DUT errors?

```
{code}extend sys {
```

```
setup() is also {
```

```
set_check( "...", ERROR_CONTINUE );
```

```
// set_check( "...", IGNORE ); -- use this to disable DUT errors completely
```

```
};
```

```
};
```

```
{/code}
```

How to instantiate a logger?

```
{code}extend your_unit {
```

```
logger: message_logger is instance;
```

```
keep soft logger.verbosity == MEDIUM;
```

```
keep soft logger.tags == {NORMAL};
```

```
short_name(): string is {
```

```
result = append("YOUR_UNIT_NAME"); // can also be a dedicated evc_name
```

```
};
```

```
// Constrain the output coloring for better visibility
```

```
short_name_style(): vt_style is {
```

```
result = BLUE;
```

```
};
```

```
};
```

```
}/code}
```

How to generate a random list of numbers?

```
{code}gen_random_list_of_numbers(num_of_items: uint, max_val: uint): list of uint is {
```

```
assert num_of_items <= max_val + 1;
```

```
var next_item: uint;
```

```
for i from 1 to num_of_items {
```

```
gen next_item keeping {
```

```
it <= max_val;
```

```
it not in result;
```

```
};
```

```
result.add(next_item);
```

```
};
```

```
result = result.sort(it);
```

```
};
```

```
{/code}
```

How to implement register side affect with vr_ad?

```
{code}extend MY_REG vr_ad_reg {
```

```
post_access(direction : vr_ad_rw_t) is {
```

```
if direction == WRITE then { // or READ
```

```
// get pointer to the enclosing reg file
```

```
var rgf := get_parents()[0].as_a(REG_FILE_TYPE vr_ad_reg_file);
```

```
// side affect
```

```
if bit0 == 1 then {
```

```
    rgf.another_register.another_bit = 1;
```

```
};
```

```
};
```

```
};
```

```
};
```

```
{/code}
```

How to control generation of Boolean variables elegantly?

```
{code}gen_true_or_false(true_prob: uint): bool is {
```

```
    assert that true_prob <= 100;
```

```
gen result keeping {
```

```
soft it == select {
```

```
true_prob : TRUE;
```

```
(100-true_prob): FALSE;
```

```
};
```

```
};
```

```
};
```

```
{/code}
```

How to declare a sequence?


```
{code}sequence my_prefix_seq_s using
```

```
item = my_prefix_frame_s,
```

```
created_kind = my_prefix_seq_kind_t,
```

```
created_driver = my_prefix_driver_u,
```

```
sequence_type = my_prefix_any_sequence_s, // optional
```

```
sequence_driver_type = my_prefix_any_sequence_driver_u; // optional
```

```
{/code}
```

How to use method ports?

```
{code}struct packet {
```

```
just_a_number: uint;
```

```
};
```

```
// define a method type  
method_type packet_method_t (p: packet);
```

```
// add an output port
```

```
unit tx_u {
```

```
broadcast_packet: out method_port of packet_method_t is instance;
```

```
send_packet_to_port() is {
```

```
broadcast_packet$(pkt);
```

```
};
```

```
};
```

```
// add an input port
```

```
unit rx_u {
```

receive_packet: in method_port of packet_method_t is instance;

```
receive_packet(p :packet) is {
```

```
message(LOW,"packet received");
```

```
print p using hex;
```

```
};
```

```
};
```

```
//make a one-to-many connection
```

```
extend sys {
```

```
tx: tx_u is instance;
```

```
rx_list[2] : list of rx_u is instance;
```

```
connect_ports() is also {
```

```
for each in rx_list {
```

```
do_bind(tx.broadcast_packet, it.receive_packet);
```

```
};
```

```
};  
};
```

```
{/code}
```