Verigen

(verilog generator)

User Guide

version 1.03

November 2, 2023

Important Notice

Document Revision History

Doc Revision Number	Date	Description
1.03	November 2, 2023	fix module definition
1.02	June 12, 2023	add default clock, add vfunctions ($LOG2, $RANGE, $DEMUX_BY_EN, $MULTICYCLE)
1.01	June 8, 2023	add interface:set_top_uppercase function
1.00	May 8, 2023	Initial relase

Table of Contents

1. INTRODUCTION 9
   - 1.1 Main functions 9
   - 1.2 Verilog automation creation process 9
   - 1.3 How to run 9
   - 1.4 license grant 10
2. FAST FOLLOW 11
   - 2.1 Step #1 : Creation of module 11
   - 2.2 Step #2 : module interconnections 12
   - 2.3 Step #3 : Verilog code insertion 15
3. MACRO FUNCTIONS 23
   - 3.1 _V macro 24
   - 3.2 vfunction macro 25
     - 3.2.1 verigen_description 26
   - 3.3 $LOG2 function 27
   - 3.4 $DEMUX_BY_EN function 28
   - 3.5 $MULTICYCLE function 29
4. CLASS AND METHOD 31
   - 4.1 clock 32
     - 4.1.1 clock:new 33
     - 4.1.2 clock:set_reset 34
     - 4.1.3 clock:get_reset 35
     - 4.1.4 clock:set_speed 36
     - 4.1.5 clock:set_default 37
     - 4.1.6 clock.find 38
     - 4.1.7 clock.is_valid 39
     - 4.1.8 clock.get_default 40
   - 4.2 interface 41
     - 4.2.1 interface:new 43
     - 4.2.2 new_signal 44
     - 4.2.3 interface.find 45
     - 4.2.4 interface.is_valid 46
     - 4.2.5 interface:set_clock 47
     - 4.2.6 interface:get_clock 48
     - 4.2.7 interface:set_signal 49
     - 4.2.8 interface:signal_count 50
     - 4.2.9 interface:set_param 51
     - 4.2.10 interface:get_param 52
     - 4.2.11 interface:set_modport 53
     - 4.2.12 interface:add_modport 54
     - 4.2.13 interface:get_modport 55
     - 4.2.14 interface:set_prefix 56
     - 4.2.15 interface:set_bared 58
     - 4.2.16 interface:set_top_uppercase 59
     - 4.2.17 interface_i:set_port 60
     - 4.2.18 interface_i:set_desc 61
     - 4.2.19 interface_i:set_prefix 62
     - 4.2.20 interface_i:get_prefix 63

4.3 module 64
4.3.1 module:new 65
4.3.2 module:make_code 66
4.3.3 module:set_inception 67
4.3.4 module:set_document 69
4.3.5 module:get_inception 70
4.3.6 module:set_title 71
4.3.7 module:set_author 72
4.3.8 module:set_param 73
4.3.9 module:get_param 74
4.3.10 module:add_interface 75
4.3.11 module:add_clock 76
4.3.12 module:get_interface 77
4.3.13 module:get_port 78
4.3.14 module:add_module 79
4.3.15 module:get_module 80
4.3.16 module:add_code 81
4.3.17 module.find 82
4.3.18 module.is_valid 83
4.3.19 module.apply_code 84
4.3.20 module_i:set_param 85
4.3.21 module_i:get_param 86
4.3.22 module_i:set_port 87
4.3.23 module_i:get_port 88
4.3.24 module_i.is_valid 89

5. APPENDIX 90

5.1 Appendix : test_definition.lua 90

List of Tables

Table **Title** **Page**

Table 3-1. Macro function/data summary 23
Table 3-2. List of predefined vfunctions 23
Table 4-1. clock object summary 32
Table 4-2. interface object summary 41
Table 4-3. interface_i object summary 41
Table 4-4. module object summary 64
Table 4-5. module_i(sub module) object summary 64

List of Figures

Figure **Title** **Page**

Figure 2-1. Step #1 Hierarchy Diagram 12
Figure 2-2. Step #2 Hierarchy Diagram 14
Figure 2-3. Step #3 Hierarchy Diagram 17

List of Terms

List	Description
TestDrive	TestDrive Profiling Master (https://testdrive-profiling-master.github.io/)
Lua	Lua script language (Wiki, Homepage)

1. Introduction

"Performance can't beat convenience."

When designing with the verilog of a large-scale project, one of the most problematic parts is that it takes a lot of time and effort to configure the control path between modules. In addition, if you need to modify some of the control paths of a design that has been completed with a lot of time and effort, or if you need a major change, you have to be more careful. Otherwise, it may introduce new errors or require the same amount of effort as recreating the design from scratch.

Therefore, I made a verigen tool that creates control paths with minimal design. This tool makes it easy and quick to build a control path programmatically with minimal effort, and has a function that allows you to check the structured control path as a design hierarchy at a glance. It can also allow for faster design changes and sharing of designs with other team members.

NOTE: If you have a new feature to suggest, or find improvements or bugs, please contact me (clonextop@gmail.com).

1.1 Main functions

verigen was created using codegen of TestDrive Profiling Master. This tool runs code written in lua, builds a verilog design, includes all codegen functionality, and generates the following files.

- Automatically generate verilog design (.sv, .f)
- Automatic creation of constraint (.xdc)
- Automatic creation of hierarchy diagram (.svg), HTML highlited source code (.html)

1.2 Verilog automation creation process

Creating a project through verigen proceeds in the following steps.

1. Write Lua scripts
    1). Create modules
    2). Connect modules
    3). Declaring parameters and interface to the module (option)
2. Write verilog codes
    1). Declaring parameters and interface to the module (option)
    2). Write Verilog additional code or write Lua mixed code
3. Run verigen to generate verilog code

1.3 How to run

To run verigen, run the following command.

```
> verigen

Verilog Generator for TestDrive Profiling Master. v1.02
Usage: verigen  [--help] input_file [output_path]

      --help               display this help and exit
  input_file               input Lua file
```

```
output_path              output path
                         default : ./output
```

NOTE: Command : `verigen INPUT_LUA_FILE  OUTPUT_PATH`

A Lua script corresponding to INPUT_LUA_FILE is created and executed. If OUTPUT_PATH is not specified, the result is created in the default "./output" folder.

1.4 license grant

The source implemented in verigen complies with the BSD license, and the user's individual scripts used to create verilog or derivative works such as verilog are wholly owned by the user.

2. Fast follow

This section is a quick, example-oriented explanation. To check the class and method in dictionary format, see the next step 'Class and Method'.

The example below describes the implementation at github example. You can achieve the same result by running do_test.bat in that folder.

2.1 Step #1 : Creation of module

Generate and run the script code as shown below.

[main.lua file]

```
verigen_description("Test project")

-- modules
core_wrapper = module:new("test_wrapper")   -- top
core = {}
core.top = module:new("test_core")
core.slave_ctrl = module:new("slave_ctrl")
core.core_if = module:new("core_if")
core.core_ex = module:new("core_ex")
core.core_wb = module:new("core_wb")
core.mem_ctrl = module:new("mem_ctrl")
core.reg_ctrl = module:new("reg_ctrl")

-- make code
core_wrapper:make_code()
```

It was created by putting the core_wrapper module and core related modules in lua table.

[Run command]

```
> verigen main.lua
*I: Build TOP design : test_wrapper.sv
*W: Empty port module : 'test_wrapper' module
*I: Build contraint : test_wrapper_constraint.xdc
*I: Make design hierarchy : test_wrapper_hierarchy.svg
*I: Make common defines : test_wrapper_defines.vh
*I: Make design file list : test_wrapper.f
```

Briefly declare the modules to be used through the module:new method. And the last module:make_code method generates the actual verilog code, constraint files and hierarchy diagram.

Currently, we have declared several modules, but since there is no module associated with core_wrapper, verilog design only creates one test_wrapper.sv file.

The rest of the test_wrapper_constraint.xdc, test_wrapper_defines.vh, etc. are empty.

[Result : test_wrapper.sv]

```
`include "test_wrapper_defines.vh"
```

```
module test_wrapper ();

endmodule
```

The resulting design is literally an empty module file, and the hierarchy diagram(test_wrapper_hierarchy.svg) is also empty.

[Result : test_wrapper_hierarchy.svg]

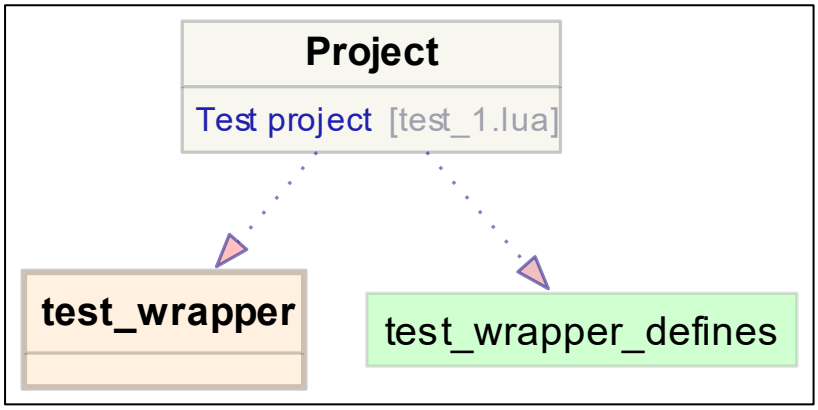

Figure 2-1. Step #1 Hierarchy Diagram

NOTE: You can view the actual verilog codes by clicking on the module name in this image.

2.2 Step #2 : module interconnections

Modify and run the Lua script as shown below.

[main.lua file]

```
 1: verigen_description("Test project")
 2:
 3: RunScript("test_definition.lua")
 4:
 5: -- modules
 6: core_wrapper = module:new("test_wrapper")    -- top
 7: core = {}
 8: core.top = module:new("test_core")
 9: core.slave_ctrl = module:new("slave_ctrl")
10: core.core_if = module:new("core_if")
11: core.core_ex = module:new("core_ex")
12: core.core_wb = module:new("core_wb")
13: core.mem_ctrl = module:new("mem_ctrl")
14: core.reg_ctrl = module:new("reg_ctrl")
15: core.busy_ctrl = module:new("busy_ctrl")
16:
17: -- module connection
18: core_wrapper:add_module(core.mem_ctrl)
19: core_wrapper:add_module(core.slave_ctrl)
20:
21: core.top:add_module(core.core_if)
22: core.top:add_module(core.core_ex)
```

```
23: core.top:add_module(core.core_wb)
24:
25: core.slave_ctrl:add_module(core.reg_ctrl)
26:
27: -- multi-core genration
28: for i = 1, config.core_size do
29:     core_wrapper:add_module(core.top)
30: end
31:
32: -- make code
33: core_wrapper:make_code()
```

Now, in the added lines 17 to 30, each module is connected with the module:add_module function, and four modules are also created and connected to the core. Include "Appendix : test_definition.lua" at the top (line #3) to use the predefined config.core_size value.

[Run command]

```
> verigen main.lua
*I: Build sub design : mem_ctrl.sv
*W: Empty port module : 'mem_ctrl' module
*I: Build sub design : reg_ctrl.sv
*W: Empty port module : 'reg_ctrl' module
*I: Build sub design : slave_ctrl.sv
*W: Empty port module : 'slave_ctrl' module
*I: Build sub design : core_ex.sv
*W: Empty port module : 'core_ex' module
*I: Build sub design : core_if.sv
*W: Empty port module : 'core_if' module
*I: Build sub design : core_wb.sv
*W: Empty port module : 'core_wb' module
*I: Build sub design : test_core.sv
*W: Empty port module : 'test_core' module
*I: Build TOP design : test_wrapper.sv
*W: Empty port module : 'test_wrapper' module
*I: Build contraint : test_wrapper_constraint.xdc
*I: Make common defines : test_wrapper_defines.vh
*I: Make design hierarchy : test_wrapper_hierarchy.svg
*I: Make design file list : test_wrapper.f
```

In the execution result, other files included in addition to the test_wrapper.sv file are automatically created, and if you look at the top design, only the module is added and the port is not described, so a warning is generated, but each submodule is automatically You can see what has been added.

[Result : test_wrapper.sv]

```
`include "test_wrapper_defines.vh"

module test_wrapper ();

/* no ports in module. (commented out for DRC.)
mem_ctrl mem_ctrl (
);*/

/* no ports in module. (commented out for DRC.)
```

```
slave_ctrl slave_ctrl (
);*/

/* no ports in module. (commented out for DRC.)
test_core test_core_0 (
);*/

/* no ports in module. (commented out for DRC.)
test_core test_core_1 (
);*/

/* no ports in module. (commented out for DRC.)
test_core test_core_2 (
);*/

/* no ports in module. (commented out for DRC.)
test_core test_core_3 (
);*/
endmodule
```

NOTE: In the code above, since the submodule has no input/output at all, it is commented out to avoid 'DRC (Design Rule Check)' errors.

[Result : test_wrapper_hierarchy.svg]

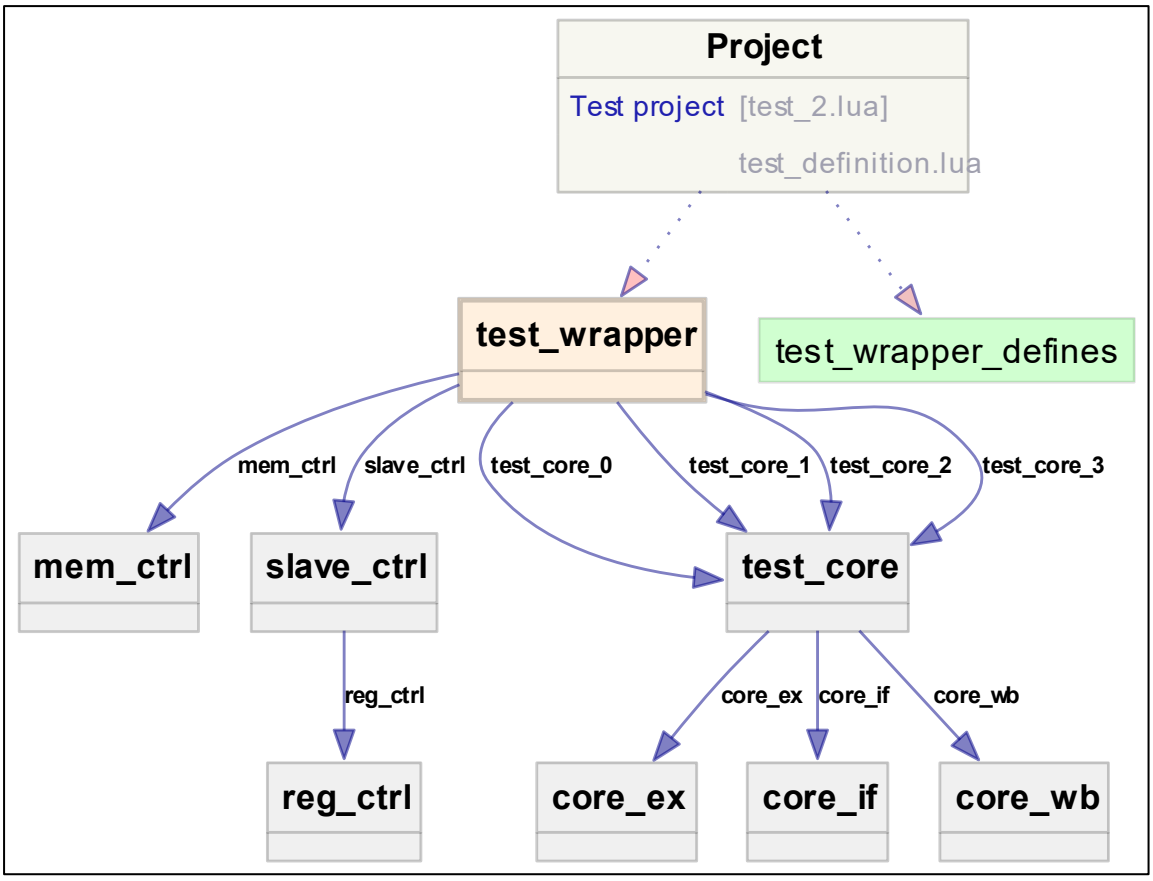

Figure 2-2. Step #2 Hierarchy Diagram

In addition to test_wrapper.sv, other slave_ctrl.sv and test_core.sv also contain submodules, as seen in Figure 2-2.

2.3 Step #3 : Verilog code insertion

Modify and run the Lua script as shown below.

[main.lua file]

```
verigen_description("Test project")

RunScript("test_definition.lua")

-- modules
core_wrapper = module:new("test_wrapper")    -- top
core = {}
core.top = module:new("test_core")
core.slave_ctrl = module:new("slave_ctrl")

core.core_if = module:new("core_if")
core.core_ex = module:new("core_ex")
core.core_wb = module:new("core_wb")
core.mem_ctrl = module:new("mem_ctrl")
core.reg_ctrl = module:new("reg_ctrl")
core.busy_ctrl = module:new("busy_ctrl")

-- add master bus
bus.maxi4:set_param("DATA_WIDTH", 512)
bus.maxi4:set_param("ADDR_WIDTH", 36)
bus.maxi4:set_prefix("M#")

-- add busy
core_busy = new_signal("core_busy")

-- module connection
core_wrapper:add_module(core.mem_ctrl)
core_wrapper:add_module(core.slave_ctrl)

core.top:add_module(core.core_if)
core.top:add_module(core.core_ex)
core.top:add_module(core.core_wb)

core.slave_ctrl:add_module(core.reg_ctrl)

-- multi-core genration
for i = 1, config.core_size do
    core_wrapper:add_module(core.top)
end

 -- add verilog codes
for entry in lfs.dir("src/") do
    local s = String(entry)
```

```
44:     if s:CompareBack(".sv") then
45:         module.apply_code("src/" .. entry)
46:     end
47: end
48:
49: -- make code
50: core_wrapper:make_code()
```

The code added to the existing Lua script is line #18~24 and #41~47.

The first changes the bit width of data and address of axi4, and the second one adds the two files below through the module.apply_code() function to all *.sv files in the subfolder "./src".

[src/__wrapper.sv]

```
1: //#--------------------------------------------------------------------
2: module test_core
3:
4: //#--------------------------------------------------------------------
5: module core_if
6: $set_param("CORE_ID", "0")
7: $add_interface(core_i.inst, "if_inst", "m")
8: $add_interface(core_busy, nil, "m")
9:
10: assign  core_busy   = 1'b0;
11:
12:
13: //#--------------------------------------------------------------------
14: module core_ex
15: $add_interface(core_i.inst, "if_inst", "s")
16: $add_interface(core_i.inst, "ex_inst", "m")
17:
18:
19: //#--------------------------------------------------------------------
20: module core_wb
21: $add_interface(core_i.inst, "ex_inst", "s")
```

NOTE: As above, by declaring 'modport' paired with the name of the same type of interface through the $add_interface() function, the same interfaces declared in two different modules are automatically connected.

[src/__wrapper.sv]

```
1: //#--------------------------------------------------------------------
2: module test_wrapper
3: wire    $RANGE(config.core_size)    core_busy_all;
4:
5: ${  -- It's Lua codes
6:     module:set_title("Fast Follow")
7:
8:     for i = 0, (config.core_size-1) do
9:         local core  = sub_module["test_core_" .. i]
10:        core:set_param("CORE_ID", i)
11:        core:set_port("core_busy", "core_busy_all[" .. i .. "]")
12:    end
```

```
13:
14:     sub_module["slave_ctrl"]:set_port("core_busy", "|core_busy_all")
15: }
16:
17: //#------------------------------------------------------------------
18: module slave_ctrl
19: $set_param("BASE_ADDR", "32'h10000000")
20: $add_interface(bus.apb, "s_apb", "m")
21: $add_interface(bus.apb, "s_apb_0", "m")
22:
23: //#------------------------------------------------------------------
24: module mem_ctrl
25: $add_interface(bus.maxi4, "maxi", "m")
26:
27: //#------------------------------------------------------------------
28: module reg_ctrl
29: $add_interface(core_busy, nil, "s")
```

NOTE: You can connect directly through the "module_i:set_param()" and "module_i:set_port()" functions without automatically connecting interfaces or parameters.

After declaring "module <module_name>"(The use of "endmodule" can be omitted.) in the added .sv file, the file can be described using both Verilog and Lua grammars. You can either declare I/O via @set_param() function and @module:add_interface() function, respectively, or use Verilog syntax directly.

In addition, if you want to directly access a Lua variable or function, you can access it with $(*), or you can execute a Lua statement by describing it with ${*}.

[Result : test_wrapper_hierarchy.svg]

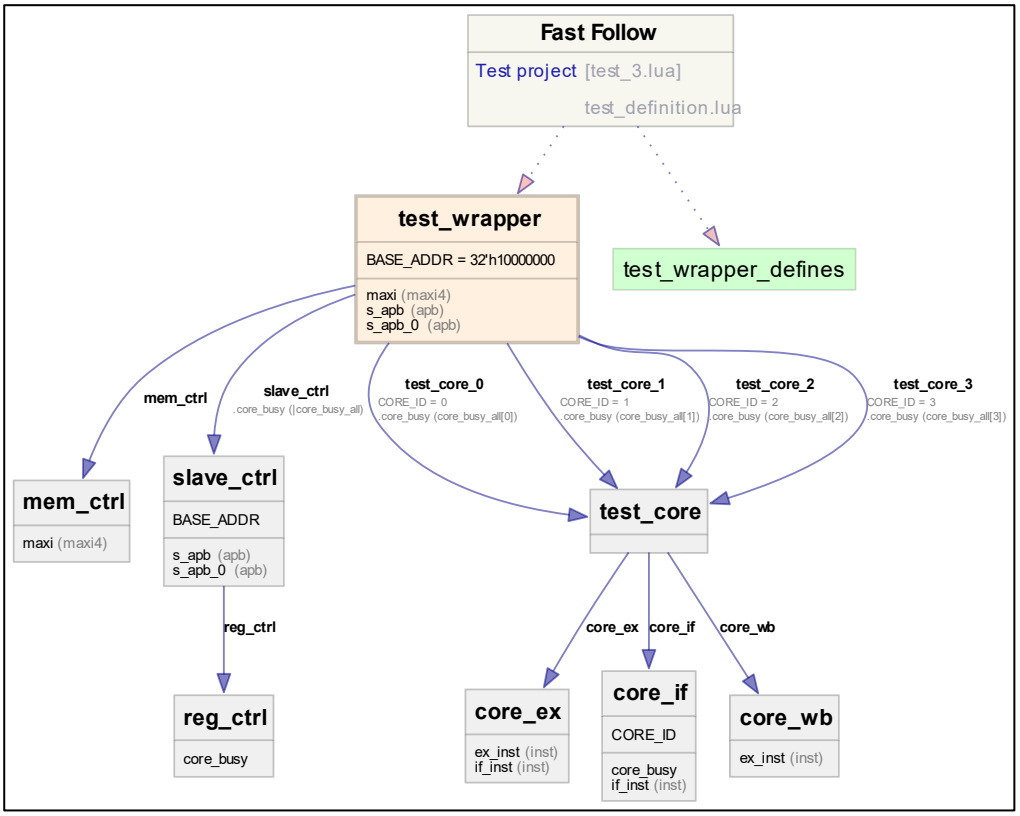

Figure 2-3. Step #3 Hierarchy Diagram

Below is the output of the top design.

[Final result : test_wrapper.sv]

```
`include "test_wrapper_defines.vh"

module test_wrapper #(
    parameter BASE_ADDR         = 32'h10000000
) (
    // clock & reset
    input                       ACLK,         // AXI clock
    input                       CLK,          // main clock
    input                       PCLK,         // APB clock
    input                       PRESETn,      // reset of 'PCLK' (active low)
    input                       nRST,         // default global reset (active low)

    // maxi
    input                       M_AWREADY,
    input                       M_WREADY,
    input                       M_BVALID,
    input  [1:0]                M_BRESP,
    input  [3:0]                M_BID,
    input                       M_ARREADY,
    input                       M_RVALID,
    input  [511:0]              M_RDATA,
    input  [1:0]                M_RRESP,
    input  [3:0]                M_RID,
    input  [3:0]                M_ARQOS,
    input  [3:0]                M_ARREGION,
    output                      M_AWVALID,
    output [35:0]               M_AWADDR,
    output [2:0]                M_AWSIZE,
    output [1:0]                M_AWBURST,
    output [3:0]                M_AWCACHE,
    output [2:0]                M_AWPROT,
    output [3:0]                M_AWID,
    output [7:0]                M_AWLEN,
    output                      M_AWLOCK,
    output                      M_WVALID,
    output                      M_WLAST,
    output [511:0]              M_WDATA,
    output [63:0]               M_WSTRB,
    output [3:0]                M_WID,
    output                      M_BREADY,
    output                      M_ARVALID,
    output [35:0]               M_ARADDR,
    output [2:0]                M_ARSIZE,
    output [1:0]                M_ARBURST,
    output [3:0]                M_ARCACHE,
    output [2:0]                M_ARPROT,
    output [3:0]                M_ARID,
    output [7:0]                M_ARLEN,
    output                      M_ARLOCK,
```

```
    output                      M_RREADY,
    output                      M_RLAST,
    output [3:0]                M_AWQOS,
    output [3:0]                M_AWREGION,

    // s_apb
    input                       S0_PREADY,
    input  [31:0]               S0_PRDATA,
    input                       S0_PSLVERR,
    output [1:0]                S0_PSEL,
    output                      S0_PENABLE,
    output                      S0_PWRITE,
    output [15:0]               S0_PADDR,
    output [31:0]               S0_PWDATA,

    // s_apb_0
    input                       S1_PREADY,
    input  [31:0]               S1_PRDATA,
    input                       S1_PSLVERR,
    output [1:0]                S1_PSEL,
    output                      S1_PENABLE,
    output                      S1_PWRITE,
    output [15:0]               S1_PADDR,
    output [31:0]               S1_PWDATA
);

// synopsys template
// interface : maxi
i_maxi4                 maxi();
assign maxi.AWREADY             = M_AWREADY;
assign maxi.WREADY              = M_WREADY;
assign maxi.BVALID              = M_BVALID;
assign maxi.BRESP               = M_BRESP;
assign maxi.BID                 = M_BID;
assign maxi.ARREADY             = M_ARREADY;
assign maxi.RVALID              = M_RVALID;
assign maxi.RDATA               = M_RDATA;
assign maxi.RRESP               = M_RRESP;
assign maxi.RID                 = M_RID;
assign maxi.ARQOS               = M_ARQOS;
assign maxi.ARREGION            = M_ARREGION;
assign M_AWVALID                = maxi.AWVALID;
assign M_AWADDR                 = maxi.AWADDR;
assign M_AWSIZE                 = maxi.AWSIZE;
assign M_AWBURST                = maxi.AWBURST;
assign M_AWCACHE                = maxi.AWCACHE;
assign M_AWPROT                 = maxi.AWPROT;
assign M_AWID                   = maxi.AWID;
assign M_AWLEN                  = maxi.AWLEN;
assign M_AWLOCK                 = maxi.AWLOCK;
assign M_WVALID                 = maxi.WVALID;
assign M_WLAST                  = maxi.WLAST;
```

```
assign M_WDATA                  = maxi.WDATA;
assign M_WSTRB                  = maxi.WSTRB;
assign M_WID                    = maxi.WID;
assign M_BREADY                 = maxi.BREADY;
assign M_ARVALID                = maxi.ARVALID;
assign M_ARADDR                 = maxi.ARADDR;
assign M_ARSIZE                 = maxi.ARSIZE;
assign M_ARBURST                = maxi.ARBURST;
assign M_ARCACHE                = maxi.ARCACHE;
assign M_ARPROT                 = maxi.ARPROT;
assign M_ARID                   = maxi.ARID;
assign M_ARLEN                  = maxi.ARLEN;
assign M_ARLOCK                 = maxi.ARLOCK;
assign M_RREADY                 = maxi.RREADY;
assign M_RLAST                  = maxi.RLAST;
assign M_AWQOS                  = maxi.AWQOS;
assign M_AWREGION               = maxi.AWREGION;

// interface : s_apb
i_apb                   s_apb();
assign s_apb.PREADY             = S0_PREADY;
assign s_apb.PRDATA             = S0_PRDATA;
assign s_apb.PSLVERR            = S0_PSLVERR;
assign S0_PSEL                  = s_apb.PSEL;
assign S0_PENABLE               = s_apb.PENABLE;
assign S0_PWRITE                = s_apb.PWRITE;
assign S0_PADDR                 = s_apb.PADDR;
assign S0_PWDATA                = s_apb.PWDATA;

// interface : s_apb_0
i_apb                   s_apb_0();
assign s_apb_0.PREADY           = S1_PREADY;
assign s_apb_0.PRDATA           = S1_PRDATA;
assign s_apb_0.PSLVERR          = S1_PSLVERR;
assign S1_PSEL                  = s_apb_0.PSEL;
assign S1_PENABLE               = s_apb_0.PENABLE;
assign S1_PWRITE                = s_apb_0.PWRITE;
assign S1_PADDR                 = s_apb_0.PADDR;
assign S1_PWDATA                = s_apb_0.PWDATA;

mem_ctrl mem_ctrl (
    .ACLK               (ACLK),
    .nRST               (nRST),
    .maxi               (maxi)
);

slave_ctrl #(
    .BASE_ADDR          (BASE_ADDR)
) slave_ctrl (
    .PCLK               (PCLK),
    .PRESETn            (PRESETn),
    .core_busy          (|core_busy_all),
```

```
    .s_apb              (s_apb),
    .s_apb_0            (s_apb_0)
);

test_core #(
    .CORE_ID            (1)
) test_core_0 (
    .CLK                (CLK),
    .nRST               (nRST),
    .core_busy          (core_busy_all[0])
);

test_core #(
    .CORE_ID            (2)
) test_core_1 (
    .CLK                (CLK),
    .nRST               (nRST),
    .core_busy          (core_busy_all[1])
);

test_core #(
    .CORE_ID            (3)
) test_core_2 (
    .CLK                (CLK),
    .nRST               (nRST),
    .core_busy          (core_busy_all[2])
);

test_core #(
    .CORE_ID            (4)
) test_core_3 (
    .CLK                (CLK),
    .nRST               (nRST),
    .core_busy          (core_busy_all[3])
);

wire    [3:0]   core_busy_all;


endmodule
```

[Final result : test_core.sv]

```
`include "test_wrapper_defines.vh"

module test_core #(
    parameter CORE_ID          = 0
) (
    // clock & reset
    input                      CLK,         // main clock
    input                      nRST,        // default global reset (active low)
```

```
    // core_busy
    output logic                core_busy
);

// synopsys template
// interface : ex_inst
i_inst                  ex_inst();

// interface : if_inst
i_inst                  if_inst();

core_ex core_ex (
    .CLK                (CLK),
    .nRST               (nRST),
    .ex_inst            (ex_inst),
    .if_inst            (if_inst)
);

core_if #(
    .CORE_ID            (CORE_ID)
) core_if (
    .CLK                (CLK),
    .nRST               (nRST),
    .core_busy          (core_busy),
    .if_inst            (if_inst)
);

core_wb core_wb (
    .CLK                (CLK),
    .nRST               (nRST),
    .ex_inst            (ex_inst)
);
endmodule
```

3. Macro functions

Provides macro functions useful for organizing your code.

Table 3-1. Macro function/data summary

Macro	Type	Description
_V	function	string expansion manipulation
vfunction	function	Declaring a function for verilog
verigen_description	function	Set verigen source code description
sub_module	data	Sub-module instance list

The functions below are predefined functions as vfunction available in verilog. It can be used as "$function_name(~)".

Table 3-2. List of predefined vfunctions

vfunction	Type	Description
$LOG2	function	log2(X) function
$RANGE	function	verilog bitwidth range template
$DEMUX_BY_EN	function	demux design template
$MULTICYCLE	function	multicycle design template
$add_clock	function	Refer to 4.3.11. module:add_clock
$add_interface	function	Refer to 4.3.10. module:add_interface
$add_code	function	Refer to 4.3.16. module:add_code
$set_param	function	Refer to 4.3.8. module:set_param
$set_inception	function	Refer to 4.3.3. module:set_inception
$set_author	function	Refer to 4.3.7. module:set_author
$_V	function	Refer to 3.1. _V macro

3.1 _V macro

Type	Description
Prototype	function _V(s, [start], [end], [step])
Return value	string
Remarks	Extends a statement by incrementing it by a step from start to end. If there is a part of the statement implemented with $(...), only that part is expanded. If there is none, the entire sentence is expanded, and the '#' character in the sentence is assigned a repeated value from start to end.
start	start value
end	end value (If omitted, it is treated the same as the start value.)
step	increase value (If omitted, it increases or decreases by 1 or -1. according to the sign of the end-start value.)

ex) _V macro example

```
print(_V("assign A = {$(B[#],)};", 0, 3))
```

[Result]

```
assign A = {B[0], B[1], B[2], B[3]};
```

3.2 vfunction macro

Type	Description
Prototype	function vfunction(name, func)
Return value	-
Remarks	You can call lua functions from within verilog with "$function(...)".
name	Function name to use within verilog
func	lua function to use in verilog

NOTE: By default, the "_V" macro is declared as vfunction , so you can use the $_V(...) function equivalently within verilog.

ex) vfunction macro example

```
vfunction("RANGE", function(size,step)
    return ("[" .. ((size*(step+1))-1) .. ":" .. (size*step) .. "]")
end)
```

[Source input]

```
wire    $RANGE(32,1)    T;
```

[Result]

```
wire    [63:32]     T;
```

3.2.1 verigen_description

Type	Description
Prototype	function verigen_description(desc)
Return value	-
Remarks	Set verigen source's description in Lua file
desc	description of verigen source file

ex) verigen_description macro example

```
verigen_description("Your description")
end)
```

3.3 $LOG2 function

Type	Description
Prototype	$LOG2(val, [bOverflow])
Return value	number
Remarks	Returns log2(val) value.
val	log2 input value
bOverflow	val must be an integer equal to 2^N . If not, return an error. Set this value to true to force rounding up on the resulting value. If omitted, false is assumed.

ex) $LOG2 example

```
val_a = 16
```

[Source input]

```
localparam      BITS    = $LOG2(val_a);
```

[Result]

```
localparam      BITS    = 4;
```

3.4 $DEMUX_BY_EN function

Type	Description
Prototype	$DEMUX_BY_EN(width, channel_count, en, data_in, data_out)
Return value	string
Remarks	Implement demux using demux_by_enable module.
width	bitwidth per data
channel_count	Number of input channels
en	input enable signal (string)
data_in	Input data (as many as the total number of channel_count, string)
data_out	output data (string)

ex) $DEMUX_BY_EN example

[Source input]

```
wire    [31:0]      a,b,c,d;
wire    [3:0]       en;
wire    [31:0]      odata;

$DEMUX_BY_EN(32, 4, "en", "{a,b,c,d}", "odata")
```

[Result]

```
wire    [31:0]      a,b,c,d;
wire    [3:0]       en;
wire    [31:0]      odata;

demux_by_enable #(
    .WIDTH              (32),
    .CHANNELS           (4),
    .TRISTATE           (1)
) demux_en_pc (
    .EN_BUS             (en),
    .DIN_BUS            ({a,b,c,d}),
    .DOUT               (odata)
);
```

3.5 $MULTICYCLE function

Type	Description
Prototype	$MULTICYCLE(module_inst_name, if_name, cycle_count, [instance_count], [clk])
Return value	string
Remarks	Multicycle Implementation Using the template "MultiCyclePath" module or "MultiCyclePathEx", a module with one interface is implemented as multicycle.
module_inst_name	Module name included as a child of the current module
if_name	Specifies the interface instance name corresponding to the module of module_inst_name.
cycle_count	cycle count (2 ≤ cycle_count ≤ 12)
instance_count	Number of instances of submodules (1 ≤ instance_count ≤ cycle_count) If omitted, it is regarded as the same number as cycle_count.
clk	Clock to use for multicycle implementation If omitted, the default clock is used. (see clock:set_default() function)

ex) $MULTICYCLE example

[Source input]

```
$MULTICYCLE("MTSP_Synchronize", "mtsp_sync", 2, 1)
```

[Result]

```
genvar i;
// multicycle design for MTSP_Synchronize
i_mtsp_sync mtsp_sync();
wire    mtsp_sync_ie, mtsp_sync_oe, mtsp_sync_iready;
generate
wire    [7:0]   pipe_i;
wire    [1:0]   pipe_o;
wire    [1:0]   __o;

MultiCyclePathEx #(
    .IWIDTH     (8),
    .OWIDTH     (2),
    .CYCLE      (2),
    .COUNT      (1)
) multi_pipe (
    .CLK        (MCLK),
    .nRST       (nRST),
    .IE         (mtsp_sync_ie),
    .IDATA      ({mtsp_sync.sync, mtsp_sync.eop}),
    .IREADY     ({mtsp_sync_iready}),
    .PIPE_I     (pipe_i),
    .PIPE_O     (pipe_o),
    .OE         (mtsp_sync_oe),
    .ODATA      (__o)
);
```

```
assign  {mtsp_sync.awake, mtsp_sync.done}   = __o;

for(i=0; i<1; i=i+1) begin
    i_mtsp_sync     __temp;
    assign  {__temp.sync, __temp.eop}   = pipe_i[`BUS_RANGE(8, i)];
    assign  pipe_o[`BUS_RANGE(2,i)] = {__temp.awake, __temp.done};

    MTSP_Synchronize MTSP_Synchronize (
        .mtsp_sync          (__temp)
    );
end
endgenerate
```

4. Class and Method

There are three object types as shown below.

- clock
  - You can generate clocks and assign them to interfaces. When the corresponding interface is used, the automatically assigned clock and reset matching the clock are declared to the port. If reset is not declared, the default reset nRST signal is automatically generated.
- interface
  - Create an interface to be used in the module. Interfaces can be created by inheriting from other interfaces. Instances requested by add_interface to a module can only call interface_i:* functions.
- module
  - You can create a module, include other submodules via the module:add_module function, or call module:add_interface to create an interface instance.

4.1 clock

Generates or manages clocks to be assigned to interfaces. The assigned clock is automatically declared according to the port of the module when the corresponding interface is used.

In addition, the speed of the corresponding clock is defined in the contraint, and false_path is automatically designated for registers between heterogeneous clocks and reset set for the clock.

Table 4-1. clock object summary

Member	Type	Description
.name	string	clock name
:new	function	clock creation
:set_reset	function	setup reset of clock
:get_reset	function	return reset of clock
:set_speed	function	set clock speed
:set_default	function	set default clock
.find	function	find clock
.is_valid	function	check valid clock object
.get_default	function	get defulat clock

4.1.1 clock:new

Type	Description
Prototype	function clock:new(name, [desc])
Return value	clock
Remarks	Create clock with name.
name	clock name.
desc	Used for clock descriptions and comments. (can skip.)

ex) creation example

```
new_clock    = clock:new("CLK")          -- generated from the base clock
new_clock:set_reset("GRSTn")

new_clock2   = new_clock:new("ACLK")     -- Clock cloned from new_clock. It inherits reset and
speed.
```

4.1.2 clock:set_reset

Type	Description
Prototype	function clock:set_reset(name)
Return value	-
Remarks	Create a reset on the clock. If reset is not generated using this function, the default reset 'nRST' signal is automatically used.
name	reset name. (active low)

ex) Example of specifying reset

```
aclock  = clock:new("ACLK")
aclock:set_reset("ARSTn")                -- Assign reset ARSTn
```

4.1.3 clock:get_reset

Type	Description
Prototype	function clock:get_reset()
Return value	string
Remarks	Returns the reset assigned to clock. If no reset is assigned, the default reset signal is returned.

4.1.4 clock:set_speed

Type	Description
Prototype	function clock:set_speed(mhz)
Return value	-
Remarks	Specifies the operating speed of the clock. If not specified using this function, the default clock 100MHz is set.
mhz	speed value of the clock. (MHz)

ex) Example of motion speed designation

```
aclock  = clock:new("ACLK")
aclock:set_speed(1000)                    -- Set 1GHz to ACK
```

4.1.5 clock:set_default

Type	Description
Prototype	function clock:set_default()
Return value	-
Remarks	Set the current clock as the default clock. The clock that is created first is set as the default clock, and it is used when a specific clock is explicitly set as the clock separately.

ex) Basic clock setting example

```
aclock  = clock:new("MCLK")
aclock:set_default()
```

4.1.6 clock.find

Type	Description
Prototype	function clock.find(name)
Return value	clock
Remarks	Find the clock. Returns 'nil' if not found.
name	The clock name to find.

ex) clock find example

```
aclock  = clock.find("ACLK")            -- Find ACLK

if aclock ~= nil then
    LOGI("ACLK is found.")              -- found.
end
```

4.1.7 clock.is_valid

Type	Description
Prototype	function clock.is_valid(obj)
Return value	boolean
Remarks	Check that the clock object is correct.
obj	A clock object to check.

ex) Example of checking the clock object

```
aclock  = clock:new("ACLK")

if clock.is_valid(aclock) then
    LOGI("aclock is clock object.")     -- clock object is correct.
end
```

4.1.8 clock.get_default

Type	Description
Prototype	function clock.get_default()
Return value	clock
Remarks	Returns the default clock.

4.2 interface

The interface object works identically to systemverilog's interface technology. If you look at the interface syntax of systemverilog, it is as follows.

[systemverilog interface 선언]

```
interface my_intface;
    logic          a;
    logic   [3:0]  b;

    // modport example
    modport s (input a, output b);      // slave modport
    modport m (input a, input  b);      // master modport
endinterface
```

NOTE: A detailed description of the systemverilog interface can be found in external links. See systemverilog modport description.

Among them, port configuration is attempted using the modport function that can be synthesized, and it is largely divided into the interface_i object created through add_interface to the interface object, which is the basic object, and the module object.

Table 4-2. interface object summary

Member	Type	Description
.name	string	Interface name
:new	function	Create interface
new_signal	function	Creating a single signal interface
.find	function	Find interface
.is_valid	function	Check if object is valid interface
:set_clock	function	Assign clock
:get_clock	function	Get clock
:set_signal	function	add signal
:signal_count	function	get total signal count
:set_param	function	Add parameter
:get_param	function	Get parameter
:set_modport	function	setup modport
:add_modport	function	Add modport
:get_modport	function	Get modport
:set_prefix	function	Designate prefix for port export
:set_bared	function	Apply into a bared signal

Table 4-3. interface_i object summary

Member	Type	Description
:set_port	function	set instance to port
:set_desc	function	Add instance description

Member	Type	Description
:set_prefix	function	Specify prefix of instance
:get_prefix	function	Get instance prefix

4.2.1 interface:new

Type	Description
Prototype	function interface:new(name)
Return value	interface
Remarks	Creates an interface with name. When creating, the default prefix is specified as ('name uppercase' + '#').
name	interface name.

ex) Example of interface creation

```
i_apb       = interface:new("APB")      -- APB interface creation
i_apb:set_signal("RARDDR", 32)
```

4.2.2 new_signal

Type	Description
Prototype	function new_signal(name, [width])
Return value	interface
Remarks	Creates a bared interface with name.
name	signal name.
width	signal bitwidth. If omitted, it is set to 1.

The actual implementation inside creates a bared interface as shown below, setting modport 's' to input and modport 'm' to output. Also, because it is a bared interface, it is not even logged as an interface in the [top_module]_include.vh header.

```
function new_signal(name, width)
    local   signal  = interface:new(name)

    if width == nil then
        width   = 1
    end

    signal:set_param("WIDTH", width)
    signal:set_signal(name, "WIDTH")
    signal:set_modport("s", {["input" ] = {name}})
    signal:set_modport("m", {["output"] = {name}})
    signal:set_prefix()     -- none prefix
    signal:set_bared()      -- bared signals
    return signal
end
```

ex) signal creation example

```
s_BUSY      = new_signal("BUSY_ALL", 4)
```

4.2.3 interface.find

Type	Description
Prototype	function interface.find(name)
Return value	interface
Remarks	Find the created interface.
name	interface name to find

ex) Example of finding an interface

```
i_APB       = interface:new("APB")

if interface.find("APB") ~= nil then
    LOGI("APB interface is existed.")
end
```

4.2.4 interface.is_valid

Type	Description
Prototype	function interface.is_valid(obj)
Return value	boolean
Remarks	Check that the interface object is correct.
obj	Clock object to check.

ex) Example of checking interface object

```
i_APB   = interface:new("APB")

if interface.is_valid(i_APB) then
    LOGI("i_APB is interface object.")      -- interface object is correct.
end
```

4.2.5 interface:set_clock

Type	Description
Prototype	function interface:set_clock(clk)
Return value	-
Remarks	Assign clock to interface.
clk	clock object

ex) Example of setting clock on interface object

```
i_APB   = interface:new("APB")

PCLK    = clock:new("PCLK", "APB's clock")
PCLK:set_reset("PRSTn")

i_APB:set_clock(PCLK)           -- PCLK setting
```

4.2.6 interface:get_clock

Type	Description
Prototype	function interface:get_clock()
Return value	clock
Remarks	Returns the clock assigned by interface.

ex) Example of setting clock on interface object

```
i_APB   = interface:new("APB")

PCLK    = clock:new("PCLK", "APB's clock")
PCLK:set_reset("PRSTn")

i_APB:set_clock(PCLK)

LOGI("APB's clock is " .. i_APB:get_clock().name)    -- print clock name
```

4.2.7 interface:set_signal

Type	Description
Prototype	function interface:set_signal(name, [bit_width])
Return value	-
Remarks	Set or change signal on interface.
name	signal name to set
bit_width	The bit width of the signal. If not set, it is regarded as 1. Also, if explicitly set to 0, the corresponding signal is not used. (In addition to constants, parameter values or formulas can be used.)

ex) Example of adding signal to interface object

```
i_axi3  = interface:new("AXI3")

-- parameter setting
i_axi3:set_param("ADDR_WIDTH", 16)
i_axi3:set_param("DATA_WIDTH", 128)

-- signal setting
i_axi3:set_signal("AWVALID")
i_axi3:set_signal("AWREADY")
i_axi3:set_signal("AWADDR", "ADDR_WIDTH")
i_axi3:set_signal("AWSIZE", 3)
i_axi3:set_signal("AWBURST", 2)
i_axi3:set_signal("WSTRB", "DATA_WIDTH/8")
```

4.2.8 interface:signal_count

Type	Description
Prototype	function interface:signal_count()
Return value	number
Remarks	Returns the number of signals defined in the interface.

4.2.9 interface:set_param

Type	Description
Prototype	function interface:set_param(name, default_value)
Return value	-
Remarks	Add or change parameters to the interface.
name	parameter name
default_value	parameter default value. (Constant or formula may be included.)

ex) Example of adding parameter to interface object

```
i_axi3  = interface:new("AXI3")

-- parameter setting
i_axi3:set_param("ADDR_WIDTH", 16)
i_axi3:set_param("DATA_WIDTH", 128)

-- Modify parameter
i_axi3:set_param("DATA_WIDTH", 256)
```

4.2.10 interface:get_param

Type	Description
Prototype	function interface:get_param(name)
Return value	number or string
Remarks	Return parameters to the interface.
name	parameter name

ex) An example of getting parameters to an interface object

```
i_axi3  = interface:new("AXI3")

-- parameter setting
i_axi3:set_param("ADDR_WIDTH", 16)
i_axi3:set_param("DATA_WIDTH", 128)

-- get parameter and print
LOGI("i_axi3's data width = " .. tostring(i_axi3:get_param("DATA_WIDTH")))
```

4.2.11 interface:set_modport

Type	Description
Prototype	function interface:set_modport(name, modport)
Return value	-
Remarks	Add modport to interface.
name	modport name
modport	It is a modport configuration table structure, and is described in the form below. {["input"]={"", ...}, ["output"]={"", ...}, ["inout"]={"", ...}}

ex) set_modport example

```
-- APB bus
bus_apb     = interface:new("apb")
bus_apb:set_param("ADDR_WIDTH", 16)
bus_apb:set_param("DATA_WIDTH", 32)
bus_apb:set_param("SEL_WIDTH", 2)
bus_apb:set_signal("PADDR", "ADDR_WIDTH")
bus_apb:set_signal("PSEL", "SEL_WIDTH")
bus_apb:set_signal("PENABLE")
bus_apb:set_signal("PWRITE")
bus_apb:set_signal("PWDATA", "DATA_WIDTH")
bus_apb:set_signal("PREADY")
bus_apb:set_signal("PRDATA", "DATA_WIDTH")
bus_apb:set_signal("PSLVERR")

bus_apb:set_modport("s", {["input" ]={"PSEL", "PENABLE", "PWRITE", "PADDR", "PWDATA"},
["output"]={"PREADY", "PRDATA", "PSLVERR"}})
bus_apb:set_modport("m", {["output"]={"PSEL", "PENABLE", "PWRITE", "PADDR", "PWDATA"},
["input" ]={"PREADY", "PRDATA", "PSLVERR"}})
```

4.2.12 interface:add_modport

Type	Description
Prototype	function interface:add_modport(name, modport)
Return value	-
Remarks	Adds a signal to an existing modport of an interface.
name	modport name
modport	It is a modport configuration table structure, and is described in the form below. {["input"]={"", ...}, ["output"]={"", ...}, ["inout"]={"", ...}}

ex) add_modport example

```
-- APB bus
bus_apb     = interface:new("apb")
bus_apb:set_param("ADDR_WIDTH", 16)
bus_apb:set_param("DATA_WIDTH", 32)
bus_apb:set_param("SEL_WIDTH", 2)
bus_apb:set_signal("PADDR", "ADDR_WIDTH")
bus_apb:set_signal("PSEL", "SEL_WIDTH")
bus_apb:set_signal("PENABLE")
bus_apb:set_signal("PWRITE")
bus_apb:set_signal("PWDATA", "DATA_WIDTH")
bus_apb:set_signal("PREADY")
bus_apb:set_signal("PRDATA", "DATA_WIDTH")
bus_apb:set_signal("PSLVERR")

bus_apb:set_modport("s", {["input" ]={"PSEL", "PENABLE", "PWRITE"}, ["output"]={"PREADY",
"PRDATA", "PSLVERR"}})
bus_apb:set_modport("m", {["output"]={"PSEL", "PENABLE", "PWRITE"}, ["input" ]={"PREADY",
"PRDATA", "PSLVERR"}})

bus_apb:add_modport('s', {["input" ]={"PADDR", "PWDATA"}})
bus_apb:add_modport('m', {["output"]={"PADDR", "PWDATA"}})
```

4.2.13 interface:get_modport

Type	Description
Prototype	function interface:get_modport(name, modport)
Return value	table
Remarks	Return a table of the interface's existing modports.
name	modport name

ex) add_modport example

```
-- APB bus
bus_apb     = interface:new("apb")
bus_apb:set_param("ADDR_WIDTH", 16)
bus_apb:set_param("DATA_WIDTH", 32)
bus_apb:set_param("SEL_WIDTH", 2)
bus_apb:set_signal("PADDR", "ADDR_WIDTH")
bus_apb:set_signal("PSEL", "SEL_WIDTH")
bus_apb:set_signal("PENABLE")
bus_apb:set_signal("PWRITE")
bus_apb:set_signal("PWDATA", "DATA_WIDTH")
bus_apb:set_signal("PREADY")
bus_apb:set_signal("PRDATA", "DATA_WIDTH")
bus_apb:set_signal("PSLVERR")

bus_apb:set_modport("s", {["input" ]={"PSEL", "PENABLE", "PWRITE"}, ["output"]={"PREADY",
"PRDATA", "PSLVERR"}})
bus_apb:set_modport("m", {["output"]={"PSEL", "PENABLE", "PWRITE"}, ["input" ]={"PREADY",
"PRDATA", "PSLVERR"}})

-- List 'input' of modeport 's'
for i, signal_name in ipairs(bus_apb:get_modport("s").input) do
    LOGI("modport 's' input : " .. signal_name)
end
```

4.2.14 interface:set_prefix

Type	Description
Prototype	function interface:set_prefix(prefix)
Return value	-
Remarks	Specify prefix string when displaying port of interface. If several identical interfaces are outputting ports in the same module at the same time, '#' characters are included in the prefix, and the number is changed to increase by 1 from 0. If it is a single interface, the character '#' is removed.
prefix	prefix string

ex) set_prefix example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

inst:set_prefix("I#")   -- Signals start with I#_*.

m       = module:new("top")
m:add_interface(inst, "inst_0", "m")
m:add_interface(inst, "inst_1", "m")

m:make_code()
```

[execution result : top_defines.vh]

```
 1: `ifndef __TOP_DEFINES_VH__
 2: `define __TOP_DEFINES_VH__
 3: `include "testdrive_system.vh"      // default system defines
 4:
 5: //-----------------------------------------
 6: // interfaces
 7: //-----------------------------------------
 8: interface i_inst;
 9:     logic                EN;
10:     logic [31:0]         INST;
11:     modport m (
12:         output  EN, INST
13:     );
14:     modport s (
15:         input   EN, INST
16:     );
17: endinterface
18:
19: `endif//__TOP_DEFINES_VH__
```

[execution result : top.sv]

```
 1: `include "top_defines.vh"
 2:
 3: module top (
 4:     // inst_0
 5:     output                  IO_EN,
 6:     output [31:0]           IO_INST,
 7:
 8:     // inst_1
 9:     output                  I1_EN,
10:     output [31:0]           I1_INST
11: );
12:
13: // interface : inst_0
14: i_inst                inst_0;
15: assign IO_EN                  = inst_0.EN;
16: assign IO_INST                = inst_0.INST;
17:
18: // interface : inst_1
19: i_inst                inst_1;
20: assign I1_EN                  = inst_1.EN;
21: assign I1_INST                = inst_1.INST;
22:
23:
24: endmodule
```

4.2.15 interface:set_bared

Type	Description
Prototype	function interface:set_bared(bared)
Return value	-
Remarks	The structure of the interface is unpacked and applied.
bared	Whether the boolean value is bared or not, if not specified, it is set to true .

Used when configuring bared signals.

ex) set_bared example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

inst:set_bared()            -- bared interface setting
```

4.2.16 interface:set_top_uppercase

Type	Description
Prototype	function interface:set_top_uppercase(en)
Return value	-
Remarks	When outputting the port of the top of the interface, it is forced to be a forced uppercase or lowercase name.
en	Uppercase or not, true(uppercase), false(lowercase), nil(original)

4.2.17 interface_i:set_port

Type	Description
Prototype	function interface_i:set_port(modport_name)
Return value	-
Remarks	Specifies the interface instance added with the module:add_interface function as the port output.
modport_name	modport name

When a basic interface is added to a module, the port output (input, output, inout) is determined through this function.

ex) interface_i:set_port example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

top         = module:new("top")

top:add_interface(inst):set_port("m")    -- Set inst interface to top output as modport 'm'
```

4.2.18 interface_i:set_desc

Type	Description
Prototype	function interface_i:set_desc(desc)
Return value	-
Remarks	Adds comments to be used as comments for interface instances added with the module:add_interface function.
desc	additional descriptive string

ex) interface_i:set_desc example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

top         = module:new("top")

i_int = top:add_interface(inst)
i_int:set_port("m") -- Set inst interface to top output as modport 'm'
i_int:set_desc("main instruction")  -- comment description
```

4.2.19 interface_i:set_prefix

Type	Description
Prototype	function interface_i:set_prefix(prefix)
Return value	-
Remarks	Specifies the prefix of the interface instance added with the module:add_interface function. If not specified, the prefix of the original interface is used.
prefix	prefix string

ex) interface_i:set_prefix example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

top         = module:new("top")

i_int = top:add_interface(inst)
i_int:set_port("m")     -- Set inst interface to top output as modport 'm'
i_int:set_prefix("IF")  -- Specify prefix
```

This interface instance is converted to IF_EN, IF_INST and output as port.

4.2.20 interface_i:get_prefix

Type	Description
Prototype	function interface_i:get_prefix()
Return value	string
Remarks	Returns the prefix of the interface instance added with the module:add_interface function. If no prefix is specified in the interface instance, the prefix of the original interface is returned.

ex) interface_i:get_prefix example

```
-- interface example
inst    = interface:new("inst")
inst:set_signal("EN")
inst:set_signal("INST", 32)
inst:set_modport("s", {["input" ]={"EN", "INST"}})
inst:set_modport("m", {["output"]={"EN", "INST"}})

top         = module:new("top")

i_int = top:add_interface(inst)
i_int:set_port("m")     -- Set inst interface to top output as modport 'm'

-- default prefix output
LOGI("PREFIX : " .. i_int:get_prefix())
```

4.3 module

An object that matches a verilog module declaration. It is created with the module:new function, and the final result source is output through the module:make_code function. At this time, the declaration of sub modules included in the lower level and interfaces used at least once are also made.

The port of the top module is converted into single signals in the form of input/output, not the systemverilog interface syntax, and the internal sub modules are described according to the interface syntax.

The sub module object added with the module:add_module function is used as the module_i interface.

Table 4-4. module object summary

Member	Type	Description
.name	string	module name
:new	function	create module
:set_inception	function	Specifies the code inception file.
:get_inception	function	Return code inception.
:set_title	function	Specifies the title of the code inception.
:set_author	function	Specifies the author of the code inception.
:set_param	function	Specify parameters.
:get_param	function	Search parameter.
:add_interface	function	Add interface.
:add_clock	function	Add clock.
:get_interface	function	Search for added interfaces.
:get_port	function	Search ports among the added interfaces.
:add_module	function	Add sub module.
:get_module	function	Search for sub modules.
:add_code	function	Add a user code statement.
.find	function	Find the module object.
.is_valid	function	Check whether the module exists.
.apply_code	function	Apply the code file to the module.
.code	String	Added code string object from module

Table 4-5. module_i(sub module) object summary

Member	Type	Description
.name	string	Returns the sub module name.
:set_param	function	Specifies the value of the parameter.
:get_param	function	Returns the value of parameter.
:set_port	function	Specifies the value of port.
:get_port	function	Returns the value of port.
.is_valid	function	Determines whether object is a valid module_i object.

4.3.1 module:new

Type	Description
Prototype	function module:new(name)
Return value	module
Remarks	Create a module.
name	module name

ex) Module creation example

```
top         = module:new("top")      -- Module creation example
```

4.3.2 module:make_code

Type	Description
Prototype	function module:make_code()
Return value	-
Remarks	Generates a module into a final output file. Here is the creation list: [top_name]_defines.vh (define and interface declarations) [all used module names].sv (result systemverilog source files) [top_name].f (all reference source names) [top_name]_constraint.xdc (constraint declaration) [top_name]_hierarchy.svg (design hierarchy diagram)

4.3.3 module:set_inception

Type	Description
Prototype	function module:set_inception(filename)
Return value	-
Remarks	Specifies the file in which code inception is described. You can use the meta sentences below for your code inception technique. __YEAR__ : The current year. Ex) 2023 __DATE__ : The current date. Example) May/08/2023 Mon __TITME__ : The current time. Ex) 19:34:21 __AUTHOR__ : Author specified by the module:set_author function. Default: "testdrive profiling master - verigen" __TITLE__ : Title specified with the module:set_title function. Default: "no_title"
filename	File name where code inception is described
bit_width	The bit width of the signal. If not set, it is regarded as 1. Also, if explicitly set to 0, the corresponding signal is not used. (In addition to constants, parameter values or formulas can be used.)

This inception text will be placed at the top of each .sv source. You can separately insert sentences such as license by specifying :set_inception, :set_title, :set_author functions for each module separately.

If you call it with module:set_inception, all generated modules will use the code inception of the base module unless otherwise specified.

ex) set_inception example

```
-- code inception setting
module:set_inception("code_inception.txt")
module:set_title("some title")
module:set_author("me")

top         = module:new("top")

top:make_code()
```

[code_inception.txt]

```
//==============================================================================
// Copyright (c) 2013 ~ __YEAR__. HyungKi Jeong(clonextop@gmail.com)
// Freely available under the terms of the 3-Clause BSD License
// (https://opensource.org/licenses/BSD-3-Clause)
//
// Title : __TITLE__
// Rev.  : __DATE__ __TIME__ (__AUTHOR__)
//==============================================================================
```

execution result

[result file : top.sv]

```
//================================================================================
// Copyright (c) 2013 ~ 2023. HyungKi Jeong(clonextop@gmail.com)
// Freely available under the terms of the 3-Clause BSD License
// (https://opensource.org/licenses/BSD-3-Clause)
//
// Title : some title
// Rev.  : May/09/2023 Tue 14:10:16 (me)
//================================================================================
`include "top_defines.vh"

module top ();

endmodule
```

4.3.4 module:set_document

Type	Description
Prototype	function module:set_document(name, filename)
Return value	-
Remarks	add module's document
name	document diplay name
filename	document file name if set Excel file with this function, you can use this with "worksheet@filename" ex) "sheet1@a.xlsx"

This function only affects "Design map". Display it as a hyperlink in "Design map" so that users can click to edit it.

ex) set_document example

```
top = module:new("top")      -- Module creation example
top:set_document("Configuration", "configuration@config.xlsx")
```

4.3.5 module:get_inception

Type	Description
Prototype	function module:get_inception()
Return value	-
Remarks	Returns the result of applying all meta sentences to the code inception phrase set with module:set_inception .

4.3.6 module:set_title

Type	Description
Prototype	function module:set_title(title)
Return value	-
Remarks	Specifies the title of the code inception. __TITLE__ meta sentences in code inception are converted to the specified title.
title	title string

4.3.7 module:set_author

Type	Description
Prototype	function module:set_author(name)
Return value	-
Remarks	Specifies the author of the code inception. The __AUTHOR__ meta-sentence in code inception translates to the specified author.
name	author string

4.3.8 module:set_param

Type	Description
Prototype	function module:set_param(name, value, [is_local])
Return value	-
Remarks	Add module parametes.
name	parameter name
value	parameter default value
is_local	If it is true, it is implemented as a localparam, otherwise it is implemented as a port parameter. Defaults to false if omitted.

ex) module:set_param example

```
top = module:new("top")      -- Module creation example

-- port parameter setting
top:set_param("DATA_WIDTH", 32)

-- local parameter setting
top:set_param("BYTE_WIDTH", "DATA_WIDTH/8", true)
```

4.3.9 module:get_param

Type	Description
Prototype	function module:get_param(name)
Return value	integer or string
Remarks	Returns the default values of the module's parameters.
name	parameter name

4.3.10 module:add_interface

Type	Description
Prototype	function module:add_interface(i, [name], [modport])
Return value	interface_i
Remarks	Adds an interface instance object to the module.
i	interface object
name	interface instance name If the name is not specified, the interface name is followed.
modport	The modport name, if used for internal declarations other than ports. Do not specify this value.

4.3.11 module:add_clock

Type	Description
Prototype	function module:add_clock(clk)
Return value	-
Remarks	Add a clock to the module.
clk	clock object

4.3.12 module:get_interface

Type	Description
Prototype	function module:get_interface(name)
Return value	interface_i
Remarks	Retrieves and returns the interface instance object added to the module.
name	interface instance object name

4.3.13 module:get_port

Type	Description
Prototype	function module:get_port(name)
Return value	interface_i
Remarks	Among the interface instance objects added to the module, the object set as the port is searched and returned.
name	interface instance object name

4.3.14 module:add_module

Type	Description
Prototype	function module:add_module(m, [name])
Return value	-
Remarks	Add sub module.
m	Module source to be a sub module
name	Sub module name. If omitted, it is named the same as the original module name. If there are several omitted sub modules with the same name, put a number in the form of "_#" after each name to avoid duplication.

4.3.15 module:get_module

Type	Description
Prototype	function module:get_module(name)
Return value	module
Remarks	Searches for and returns sub modules.
name	sub module name

4.3.16 module:add_code

Type	Description
Prototype	function module:add_code(s)
Return value	-
Remarks	Add user code. These codes are included as statements inserted at the end of each module source. It is appended to the module.code(String) object, and can be usefully used with the _V() macro function. The last character of the added code is ';' If it ends with , the enter code is automatically inserted.
s	user add code

4.3.17 module.find

Type	Description
Prototype	function module.find(name)
Return value	module
Remarks	Find the created module.
name	module name to find

4.3.18 module.is_valid

Type	Description
Prototype	function module.is_valid(obj)
Return value	boolean
Remarks	Check that the object is valid module.
obj	Module object to check.

4.3.19 module.apply_code

Type	Description
Prototype	function module.apply_code(filename)
Return value	-
Remarks	Code is read from the code description file and inserted into each module as code. After starting with ":module name (option)" in the code description file, the code in the module from the next line is inserted when the result of the option is true. Option is a Lua script with a Boolean result indicating whether sub-specified codes are inserted. This option can be omitted. (default value: true)
filename	code description file name

ex) module.apply_code example (When you want to add code to Core and ALU modules.)

```
module.apply_code("__core.sv")
```

[__core.sv]

```
:Core
assign  A = B;      // Core's code
assign  C = D;      // Core's code

:ALU (config.core_size > 4)
assign  E = F;      // ALU's code
assign  G = H;      // ALU's code
wire    [15:0]  CORE_SIZE   = $(config.core_size);
```

NOTE: You can execute lua code by writing '$(*)' or '${*}' in the middle of verilog code. '$(*)' is a string or number returned code, and '${*}' can describe lua code execution without return.

4.3.20 module_i:set_param

Type	Description
Prototype	function module_i:set_param(name, val)
Return value	-
Remarks	Specifies the parameter value of the sub module.
name	parameter name
val	parameter setting value

4.3.21 module_i:get_param

Type	Description
Prototype	function module_i:get_param(name)
Return value	integer or string
Remarks	Returns the value specified as the parameter of the sub module.
name	parameter name

4.3.22 module_i:set_port

Type	Description
Prototype	function module_i:module_i:set_port(name, val)
Return value	-
Remarks	Specifies the port value of the sub module.
name	port name
val	port setting value

4.3.23 module_i:get_port

Type	Description
Prototype	function module_i:get_port(name)
Return value	interger or string
Remarks	Returns the port value of the sub module.
name	port name

4.3.24 module_i.is_valid

Type	Description
Prototype	function module_i.is_valid(obj)
Return value	boolean
Remarks	Returns whether object is a valid module_i(sub module) object.
obj	module_i object

5. Appendix

5.1 Appendix : test_definition.lua

```
-------------------------------------------------------
-- clock definition
-------------------------------------------------------
clk     = {}

clk.MCLK    = clock:new("CLK", "main clock")    -- for core
clk.MCLK:set_speed(1000)

clk.PCLK    = clock:new("PCLK", "APB clock")
clk.PCLK:set_speed(100)
clk.PCLK:set_reset("PRESETn", "low")

clk.BCLK    = clock:new("ICLK", "interconnection clock")
clk.BCLK:set_speed(1500)

clk.ACLK    = clock:new("ACLK", "AXI clock")
clk.ACLK:set_speed(1000)

-------------------------------------------------------
-- bus interface
-------------------------------------------------------
bus     = {}

-- APB bus
bus.apb     = interface:new("apb")
bus.apb:set_clock(clk.PCLK)
bus.apb:set_param("ADDR_WIDTH", 16)
bus.apb:set_param("DATA_WIDTH", 32)
bus.apb:set_param("SEL_WIDTH", 2)
bus.apb:set_signal("PADDR", "ADDR_WIDTH")
bus.apb:set_signal("PSEL", "SEL_WIDTH")
bus.apb:set_signal("PENABLE")
bus.apb:set_signal("PWRITE")
bus.apb:set_signal("PWDATA", "DATA_WIDTH")
bus.apb:set_signal("PREADY")
bus.apb:set_signal("PRDATA", "DATA_WIDTH")
bus.apb:set_signal("PSLVERR")

bus.apb:set_modport("s", {["input" ]={"PSEL", "PENABLE", "PWRITE", "PADDR", "PWDATA"},
["output"]={"PREADY", "PRDATA", "PSLVERR"}})
bus.apb:set_modport("m", {["output"]={"PSEL", "PENABLE", "PWRITE", "PADDR", "PWDATA"},
["input" ]={"PREADY", "PRDATA", "PSLVERR"}})

bus.apb:set_prefix("S#")

-- AXI3 master bus
```

```
bus.maxi3   = interface:new("maxi3")
bus.maxi3:set_clock(clk.ACLK)
bus.maxi3:set_param("DATA_WIDTH", 128)
bus.maxi3:set_param("ADDR_WIDTH", 32)
bus.maxi3:set_param("ID_WIDTH", 4)
-- write address
bus.maxi3:set_signal("AWVALID")
bus.maxi3:set_signal("AWREADY")
bus.maxi3:set_signal("AWADDR", "ADDR_WIDTH")
bus.maxi3:set_signal("AWSIZE", 3)
bus.maxi3:set_signal("AWBURST", 2)
bus.maxi3:set_signal("AWCACHE", 4)
bus.maxi3:set_signal("AWPROT", 3)
bus.maxi3:set_signal("AWID", "ID_WIDTH")
bus.maxi3:set_signal("AWLEN", 4)
bus.maxi3:set_signal("AWLOCK", 2)
-- write data
bus.maxi3:set_signal("WVALID")
bus.maxi3:set_signal("WREADY")
bus.maxi3:set_signal("WLAST")
bus.maxi3:set_signal("WDATA", "DATA_WIDTH")
bus.maxi3:set_signal("WSTRB", "DATA_WIDTH/8")
bus.maxi3:set_signal("WID", "ID_WIDTH")
-- write response
bus.maxi3:set_signal("BVALID")
bus.maxi3:set_signal("BREADY")
bus.maxi3:set_signal("BRESP", 2)
bus.maxi3:set_signal("BID", "ID_WIDTH")
-- read address
bus.maxi3:set_signal("ARVALID")
bus.maxi3:set_signal("ARREADY")
bus.maxi3:set_signal("ARADDR", "ADDR_WIDTH")
bus.maxi3:set_signal("ARSIZE", 3)
bus.maxi3:set_signal("ARBURST", 2)
bus.maxi3:set_signal("ARCACHE", 4)
bus.maxi3:set_signal("ARPROT", 3)
bus.maxi3:set_signal("ARID", "ID_WIDTH")
bus.maxi3:set_signal("ARLEN", 4)
bus.maxi3:set_signal("ARLOCK", 2)
-- read data
bus.maxi3:set_signal("RVALID")
bus.maxi3:set_signal("RREADY")
bus.maxi3:set_signal("RLAST")
bus.maxi3:set_signal("RDATA", "DATA_WIDTH")
bus.maxi3:set_signal("RRESP", 2)
bus.maxi3:set_signal("RID", "ID_WIDTH")

bus.maxi3:set_modport("s", {
    ["input"]={
            "AWVALID", "AWADDR", "AWSIZE", "AWBURST", "AWCACHE", "AWPROT", "AWID", "AWLEN",
"AWLOCK",
            "WVALID", "WLAST", "WDATA", "WSTRB", "WID",
```

```
            "BREADY",
            "ARVALID", "ARADDR", "ARSIZE", "ARBURST", "ARCACHE", "ARPROT", "ARID", "ARLEN",
"ARLOCK",
            "RREADY", "RLAST"
            },
    ["output"]={
            "AWREADY", "WREADY", "BVALID", "BRESP", "BID", "ARREADY", "RVALID", "RDATA", "RRESP",
"RID"
            }
})
bus.maxi3:set_modport("m", {
    ["output"]={
            "AWVALID", "AWADDR", "AWSIZE", "AWBURST", "AWCACHE", "AWPROT", "AWID", "AWLEN",
"AWLOCK",
            "WVALID", "WLAST", "WDATA", "WSTRB", "WID",
            "BREADY",
            "ARVALID", "ARADDR", "ARSIZE", "ARBURST", "ARCACHE", "ARPROT", "ARID", "ARLEN",
"ARLOCK",
            "RREADY", "RLAST"
            },
    ["input"]={
            "AWREADY", "WREADY", "BVALID", "BRESP", "BID", "ARREADY", "RVALID", "RDATA", "RRESP",
"RID"
            }
})

bus.maxi3:set_prefix("M#")

-- AXI4 master bus
bus.maxi4   = bus.maxi3:new("maxi4")
bus.maxi4:set_signal("AWLOCK")          -- modified 2bit to 1bit
bus.maxi4:set_signal("ARLOCK")          -- modified 2bit to 1bit
bus.maxi4:set_signal("AWLEN", 8)        -- modified 4bit to 8bit
bus.maxi4:set_signal("ARLEN", 8)        -- modified 4bit to 8bit
bus.maxi4:set_signal("AWQOS", 4)        -- new on AXI4
bus.maxi4:set_signal("AWREGION", 4)     -- new on AXI4
bus.maxi4:set_signal("ARQOS", 4)        -- new on AXI4
bus.maxi4:set_signal("ARREGION", 4)     -- new on AXI4

bus.maxi4:add_modport("s", {
    ["output"]={"ARQOS", "ARREGION"},
    ["input" ]={"AWQOS", "AWREGION"}
})
bus.maxi4:add_modport("m", {
    ["input" ]={"ARQOS", "ARREGION"},
    ["output"]={"AWQOS", "AWREGION"}
})

bus.maxi4:set_prefix("M#")

--------------------------------------------------------
-- core interface
--------------------------------------------------------
```

```
core_i  = {}
core_i.inst     = interface:new("inst")
core_i.inst:set_signal("EN")
core_i.inst:set_signal("INST", 32)
core_i.inst:set_signal("READY")
core_i.inst:set_modport("m", {
    ["output"]={"EN", "INST"},
    ["input"] ={"READY"}
})
core_i.inst:set_modport("s", {
    ["input"]={"EN", "INST"},
    ["output"] ={"READY"}
})
core_i.inst:set_clock(clk.MCLK)

------------------------------------------------------
-- configuration
------------------------------------------------------
config              = {}
config.core_size    = 4
```