UltraSky User Manual

Contents

1 Document Information

1.1 About Document

Release Date	12/16/2020
Copyright	Shanghai JiuXiao Intelligent Technology Co., Ltd.
Revision	1.0
Scope	Support JXiao1.0 and higher version
Restriction	No part of this document may be reproduced in any form or by any means with written permission of Shanghai JiuXiao Intelligent Technology Co., Ltd.

Table 1 - About Document

1.2 Revision history

Revision	Date	Description
0.1	Nov.02.2020	Initial draft
0.2	Dec.12.2020	Add all rules and descriptions
1.0	Dec.16.2020	Add Information about IDE
1.1	Janu.11.2021	Add AutoBench function description of IDE
1.5	Janu.14.2021	Update AutoBench class description
1.6	Janu.27.2021	Update AutoBench demon
1.7	Janu.28.2021	Update AutoBench demon
1.8	Mar.28.2021	Update AutoBench demon
1.9	Apr.28.2021	Add description for reduction operation
2.0	May.02.2021	Add description for Autobench
2.1	May.08.2021	Add description for Autobench
2.2	May.09.2021	Add rule for floating signals

Table 2 - Revision History

2 Basic Operations

JXiao1.0 IDE(Integrated Development Environment) provides three functionalities: HDL edit environment, HDL rule check(Lint) and verification assistant tool(AutoBench).
Here is the overview of IDE.

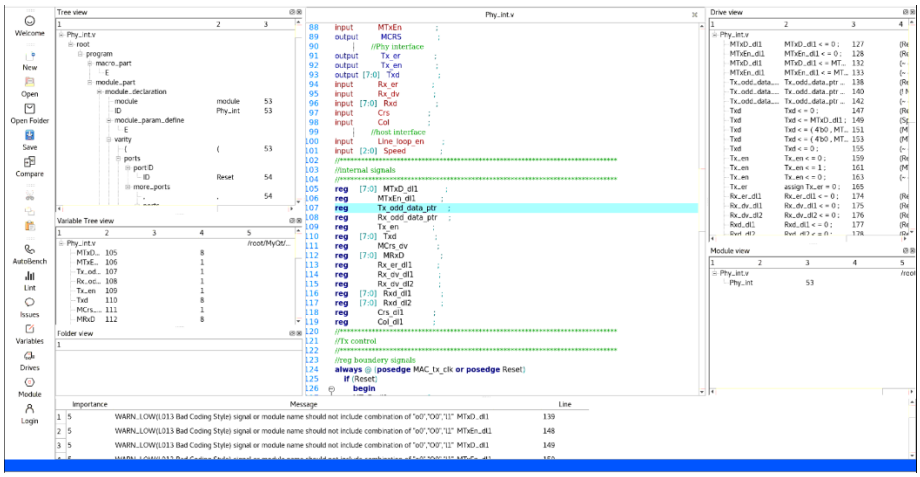


Figure 1 – IDE overview

As shown in *Figure 2,* there are three areas included inside IDE overview. On the very left of *Figure 3* is "**Menu Panal**", which contains function buttons the IDE supported. Bottom of the IDE is "**Message Box**", which displays Lint logs. "**Main Body**" of the IDE shows views of user files.

2.1 Menu Panel

Menu panel contains function buttons of the IDE listed in Table 3.

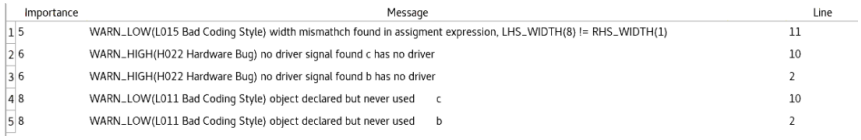
Button	Function Description
 Welcome	**Hotkey** : Not applicable **Function**: Switch between start view and current view **Use case**: Start view displays some historical files open, users can click this button any time to switch current code view to start view
 New	**Hotkey** : Ctrl + n **Function:** create new files **Use case:** create new files
 Open	**Hotkey** : Ctrl + o **Function:** open an existing single file **Use case:** open an existing single file for HDL check or further edit
 Open Folder	**Hotkey** : Not applicable **Function:** open an existing single file **Use case:** open an existing single file for HDL check or further edit
 Save	**Hotkey** : Ctrl + s **Function:** open an existing single file **Use case:** open an existing single file for HDL check or further edit
 Compare	**Hotkey** : Not applicable **Function:** compare content of two files **Use case:** compare content of two similar files, like different version of the same file
	**Hotkey** : cut(Ctrl + x), copy(Ctrl + c) and paste(Ctrl + v) **Function:** cut, copy and paste **Use case:** cut, copy and paste some content
 AutoBench	**Hotkey** : Not applicable **Function:** generate UVM testbench automatically **Use case:** when RTL top file is ready, this tool can generate UVM testbench automatically
 Lint	**Hotkey** : Not applicable **Function:** HDL language Lint check **Use case:** checks coding style issue, simulation issue and RTL bugs statically on RTL file. Rules supported can be found in other section of this document.
 Issues	**Hotkey** : Not applicable **Function:** fold/unfold “Message box” panel **Use case:** choose to display “Message box” or not
 Variables	**Hotkey** : Not applicable **Function:** open “Variables” panel **Use case:** display “Variables” panel
 Drives	**Hotkey** : Not applicable **Function:** open “Drives” panel **Use case:** display “Drives” panel

 Module	**Hotkey** : Not applicable **Function:** open “Module” panel **Use case:** display “Module” panel
 Login	**Hotkey** : Not applicable **Function:** Log in **Use case:** Log in

Table 4 - Function buttons

2.2 Message Box

Message box displays Lint rule check log.



	Importance	Message	Line
1	5	WARN_LOW(L015 Bad Coding Style) width mismathch found in assigment expression, LHS_WIDTH(8) != RHS_WIDTH(1)	11
2	6	WARN_HIGH(H022 Hardware Bug) no driver signal found c has no driver	10
3	6	WARN_HIGH(H022 Hardware Bug) no driver signal found b has no driver	2
4	8	WARN_LOW(L011 Bad Coding Style) object declared but never used c	10
5	8	WARN_LOW(L011 Bad Coding Style) object declared but never used b	2

Figure 4 – Message box

As shown in *Figure 5* , functions from left column to right in the Message box are Waring No, Importance, Warning Message and related line.

2.3 Main Body

Main Body of the IDE contains different views of user files. Functions of them are listed in Table 5 .

View Name	Description
Tree view	**Function** : Displays abstract syntax tree of current RTL files. **How to produce:** open an HDL file or save it after edit, Tree view will be refreshed
Variable Tree view	**Function** : Displays variables analyzed from HDL files. **How to produce:** open an HDL file or save it after edit, Tree view will be refreshed
Folder view	**Function** : Displays folders opened. **How to produce:** open a folder
Main file Text view	**Function** : Displays user files for editing or checking. **How to produce:** open a file of create a new file
Main file Text view	**Function** : Displays user files for editing or checking. **How to produce:** open a file of create a new file
Drive view	**Function** : Displays drive relationship of all variables in HDL file. **How to produce:** open an HDL file or save it after edit, Drive view will be refreshed
Module view	**Function** : Displays modules' hierarchy of current HDL file. **How to produce:** open an HDL file or save it after edit, Drive view will be refreshed

Table 6 - Different views in Main body

3 Lint Rules

Three types issues will be detected by Lint rule check, the severity from low to high are marked as "**WARN_LOW**", "**WARN_MEDIUM**" and "**WARN_HIGH**".

WARN_LOW

- Description: Warnings marked as "WARN_LOW" means lowest severity.
- Root cause: Mainly bad coding style. For example, bad naming convention, redundant code or mismatch of variable type or width.
- Damage: This type of issues may **not** be detected by simulation tools. Bad coding styles make the code hard to maintain.
- How to fix: Recommend fixing according to Lint logs.

WARN_MEDIUM

- Description: Warnings marked as "WARN_MEDIUM" means medium severity.
- Root cause: Simulation issues introduced by typos. For example, infinite loop, negative delay or functions without return value.
- Damage: Some of this type of issues may captured by simulation tools. Others will be explained by simulation tools, but different tools may explain in different ways. These issues introduce simulation problems and may not be found until very late stage of project.
- How to fix: Should be fixed before simulation according to Lint logs.

WARN_HIGH

- Description: Warnings marked as "WARN_HIGH" means highest severity.
- Root cause: Hardware bugs introduced by engineers in design stage. For example, buffered clock, asynchronous loop or multiple drivers.
- Damage: Most of these issues will introduce design bugs. Some of them can be directly captured by simulation and synthesis tools. Others will cause mismatch between pre- and post-simulation.
- How to fix: Should be fixed before simulation and synthesis according to Lint logs. For some special cases, need designers double confirm one by one before manually add "ignore check".

3.1 Bad Coding Style

3.1.1 L001: found undeclared variable

Warning Message

WARN_LOW(L001 Bad Coding Style) found undeclared variable

Severity

Low, recommend fixing, but can be ignored.

Description

Check undeclared variables

Example

```
module test(in1, in2, out1);
    input in1,in2;
    output out1;
    out1=in1&in2;
    assign tmp = in1&in2;    //Warning: found Undeclared variable “tmp”
endmodule
```

3.1.2 L002: vectors driven separately in different blocks.

Warning Message

WARN_LOW(L002 Bad Coding Style) vectors driven separately in different blocks

Severity

Low, recommend fixing, but can be ignored.

Description

Check drivers of vectors. Drive different vector bits in different blocks is a bad coding style.

Example

```
module test(clk, in, out);
    input in;
    output [1:0] out;
    always @(in)
```



```
        out[1]=in;
    always @(in)
        out[2]=!in;//vectors driven separately in different blocks
endmodule
```

3.1.3 L003: output is directly driven by input

Warning Message

WARN_LOW(L003 Bad Coding Style) output is directly driven by input

Severity

Low, recommend fixing, but can be ignored.

Description

Check drivers of output signals. Output signals directly driven by input is a bad coding style. These signals should be removed from target module.

Example

```
module test(in, out);
    input in;
    output out;
    out=in;//output is directly driven by input
endmodule
```

3.1.4 L004: FSM style not recommended

Warning Message

WARN_LOW(L004 Bad Coding Style) FSM style not recommended

Severity

Low, recommend fixing, but can be ignored.

Description

Check FSM coding style, two process style are recommended.

Example

```
module test(clk, rst, in, current_state, out);
    input in,clk,rst;
    input [1:0] current_state;
```



```
    output reg[1:0] out;
    parameter ST0=0;
    parameter ST1=1;
    parameter ST2=2;
    parameter ST3=3;
    reg[1:0] next_state=0;
    always @(posedge clk or posedge rst) //FSM in one process is not a recomended coding style
    begin
        if(rst)
            current_state = ST0;
        else
            begin //else_start
                case (current_state)
                    ST0:
                        begin
                            out = in + 2'b11;
                            next_state = ST1;
                        end
                    ST1:
                        begin
                            out = in + 2'b10;
                            next_state = ST2;
                        end
                    ST2:
                        begin
                            out = in + 2'b11;
                            next_state = ST3;
                        end
                    default:
                        begin
                            out = 2'b0;
                            next_state = ST0;
                        end
                endcase
                current_state = next_state;
            end //else_end
    end
endmodule
```


3.1.5 L005: only partial value of function returned.

Warning Message

WARN_LOW(L005 Bad Coding Style) only partial value of function returned.

Severity

Low, recommend fixing, but can be ignored.

Description

Check whether function return value is properly assigned.

Example

```
module test(clk,rst,in,out);
    output reg [7:0] out;
    input reg [7:0]    in;
    input clk,rst;

    function [7:0] function_val;
        function_val[5:2]=in[3:0];//only partial value of function returned.
    endfunction

    out=function_va;
endmodule
```

3.1.6 L006: reduction on signal bit signal is not recommended.

Warning Message

WARN_LOW(L006 Bad Coding Style) reduction on signal bit signal is not recommended.

Severity

Low, recommend fixing, but can be ignored.

Description

Check expressions on signal bit signal, reduction operations(&,|,^,~&,~|,~^) are not recommended

.

Example

```
module test(clk,in,out);
```



```
    output reg out;
    input   in;
    input clk;
    always @( posedge clk )
      out<=(&in); //reduction operation on signal bit signal
endmodule
```

3.1.7 L007: empty process detected!

Warning Message

WARN_LOW(L007 Bad Coding Style) empty process detected!

Severity

Low, recommend fixing, but can be ignored.

Description

Check empty process, including empty function, task and always block

Example

```
module test(clk,rst,in,out);
    output out;
    input reg in;
    input clk,rst;
    out=!in;
    always @( posedge clk or negedge rst ); //empty process
endmodule
```

3.1.8 L008: instance and module name should be different

Warning Message

WARN_LOW(L008 Bad Coding Style) instance and module name should be different

Severity

Low, recommend fixing, but can be ignored.

Description

Check to see whether instance and module name are different

Example

```
module test(in,out);
    input in;
    output out;
    or or(in,0,out);//instance and module name should be different
endmodule
```

3.1.9 L009: more than one module found in the same file

Warning Message

WARN_LOW(L009 Bad Coding Style) more than one module found in the same file

Severity

Low, recommend fixing, but can be ignored.

Description

Check whether there are more than one modules in the same file

Example

```
module   module1(in, out);//more than one modules found in the same file
    input in;
    output out;
    out=!in;
endmodule

module   module12(in, out);//more than one modules found in the same file
    input in;
    output out;
    out=!in;
endmodule
```

3.1.10 L010: more than one statement found on the same line

Warning Message

WARN_LOW(L010 Bad Coding Style) more than one statement found on the same line

Severity

Low, recommend fixing, but can be ignored.

Description

Check muiltiple statements on the same line, which is a bed coding style.

Example

```
module test(in,output);
    input in; output out;//more than one statement found on the same line
    out=!in;
endmodule
```

3.1.11 L011: object declared but never used

Warning Message

WARN_LOW(L011 Bad Coding Style) object declared but never used

Severity

Low, recommend fixing, but can be ignored.

Description

Check unused objects

Example

```
module test(clk,set,in,out,out2);
    output reg out;
    input reg in;
    input set;
    input clk;
    wire tmp;//Warning: object declared but never used
    always @( posedge clk or posedge set )
        if( set )
            out <= 1;
        else
            out <= in;
endmodule
```


3.1.12 L012: redundant case label found

Warning Message

WARN_LOW(L012 Bad Coding Style) redundant case label found

Severity

Low, recommend fixing, but can be ignored.

Description

Check to see if there is any case label containing two or more expressions that result in the same value.

Example

```
module test(in,out);
    output reg [2:0] out;
    input [1:0] in;
    wire [1:0] tmp;
    assign tmp=in;
    always @(in)
        case (in)
            0: out = 'd0;
            1: out = 'd1;
            2: out = 'd2;
            3: out = 'd3;
            3: out = 'd3;//Warning: redundant case label found
        endcase
endmodule
```

3.1.13 L013: bad naming convention

Warning Message

WARN_LOW(L013 Bad Coding Style) signal or module name should not include combination of "o0","O0","l1"

Severity

Low, recommend fixing, but can be ignored.

Description

Check not recommended naming style which is confusing, signal or module

name should not include combination of "o0","O0","l1"

Example

```
module test(l1, b, O0);
    input l1, b; //SemanticNamesEasyToGetConfused signal or module name
                 should not include combination of ""o0"","O0","l1""
    output O0; //SemanticNamesEasyToGetConfused signal or module name
                 should not include combination of ""o0"","O0","l1""
    and ando0(O0, l1, b); //SemanticNamesEasyToGetConfused signal or
                          module name should not include combination
                          of ""o0"","O0","l1""
endmodule
```

3.1.14 L014: suspicious semicolon found

Warning Message

WARN_LOW(L014 Bad Coding Style) suspicious semicolon found

Severity

Low, recommend fixing, but can be ignored.

Description

Check suspicious semicolons in file

Example

```
module test(clk,rst,in,out,out2);
    output reg out;
    input reg in;
    input rst;
    input clk;
    always @( posedge clk or posedge rst )
        if( rst );//Warning: suspicious semicolon found
            out <= 1'b0;
        else
            out <= !in;
endmodule
```


3.1.15 L015: width mismatch found in assignment expression

Warning Message

WARN_LOW(L015 Bad Coding Style) width mismatch found in assignment expression, LHS_WIDTH != RHS_WIDTH

Severity

Low, recommend fixing, but can be ignored.

Description

Check width mismatch in assignment statements

Example

```
module test(in1, out1);
    input in1;
    output [15:0] out1;
    assign out = in1;   //Warning: width mismatch found in assignment
                          expression, LHS_WIDTH(16)!= RHS_WIDTH(1)
endmodule
```

3.1.16 L016: width mismatch found in operation

Warning Message

WARN_LOW(L016 Bad Coding Style) width mismatch found in operation, OPERATION_LEFT_WIDTH != OPERATION_RIGHT_WIDTH in operation

Severity

Low, recommend fixing, but can be ignored.

Description

Check width mismatch in operations.

Example

```
module test(in1,in2, in3,in4,out1);
    input [7:0] in1,in2;
    output [15:0] in3,in4,out1;
  reg [15:0] out1;
    always @(in1 or in2 or in3 or in4)
      if((in1+in2)<(in3+in4))//Warning: width mismatch found in operation,
                              OPERATION_LEFT_WIDTH(9)        !=
                              OPERATION_RIGHT_WIDTH(16) in operation
```



```
            out1='b0;
    else
            out1='b1;
endmodule
```

3.1.17 L017: wire not explicitly declared

Warning Message

WARN_LOW(L017 Bad Coding Style) wire not explicitly declared

Severity

Low, recommend fixing, but can be ignored.

Description

Check undeclared wire signals.

Example

```
module test(in, out);
    input in;
    output [1:0] out;
    tmp=!in;//""tmp"" not declared explicitly
    out={tmp,in};
endmodule
```

3.1.18 L018: floating signal

Warning Message

WARN_LOW(L018 Bad Coding Style) floating signal

Severity

Low, recommend fixing, but can be ignored.

Description

Found floating signals while instancing a submodule.

Example

```
module test(in, out);
    input in;
```



```
    output [1:0] out;
    tmp=!in;//""tmp"" not declared explicitly
    SUBMODULE submodule1 (
    .Inport(in),
    .SPO(), //"floating signal" found
    .Outport(output));
endmodule
```


3.2 Simulation Issue

3.2.1 M001: found delay in non-blocking assignment

Warning Message

WARN_MEDIUM(M001 Simulation Issue) found delay in non-blocking assignment

Severity

Medium, should fix before start simulation

Description

Check non-blocking assignment, delay operation is not allowed.

Example

```
module test(clk, rst, in, out);
    input clk, rst, in;
    output reg out;
    parameter DELAY = 10;
    always @(posedge clk or posedge rst)
        if (rst)
            #DELAY out <= 0;//warning delay in non-blocking assignment
        else
            #DELAY out <= in;//warning delay in non-blocking assignment
endmodule
```

3.2.2 M002: found infit loop

Warning Message

WARN_MEDIUM(M002 Simulation Issue) found infit loop

Severity

Medium, should fix before start simulation

Description

Check infinite loop. Infinite loop may not be detected by simulation tools, need to fix before simulation.

Example

```
module test(in, out);
    input in;
    output [7:0] reg c;
    integer i;
    always @(in)
        for (i=0; i<7; i=i-1) //infinite loop
            c = c+1;
endmodule
```

3.2.3 M003: input signal found assigned.

Warning Message

WARN_MEDIUM(M003 Simulation Issue) input signal found assigned.

Severity

Medium, should fix before start simulation

Description

Check driver of input signal which will result in multi driver issue.

Example

```
module test(in, out);
    input in;
    output out;
    assign out=!in;
    assign in=1;//warning,    driver on input signal ""in""
endmodule
```

3.2.4 M004: negative delay found

Warning Message

WARN_MEDIUM(M004 Simulation Issue) negative delay found

Severity

Medium, should fix before start simulation

Description

Check delay time, negative delay is not allowed.

Example

```
module test(in, out);
    input in;
    output reg out;
    parameter DELAY = -10;
    always @(in)
        out = #DELAY in; //negative delay of ""DELAY"" is not allowed
endmodule
```

3.2.5 M005: siganal are not allowed to assign to itself.

Warning Message

WARN_MEDIUM(M005 Simulation Issue) siganal are not allowed to assign to itself.

Severity

Medium, should fix before start simulation

Description

Check drivers of all signals, its not permited to assign signal to itself.

Example

```
module test(in, out);
    input in;
    output reg out;
    wire tmp=!in;
    assign tmp = tmp;//assign signal to itself is not allowed
    assign out=tmp;
endmodule
```


3.2.6 M006: siganl stuck at fixed value.

Warning Message

WARN_MEDIUM(M006 Simulation Issue) siganl stuck at fixed value.

Severity

Medium, should fix before start simulation

Description

Check all signals,siganls should not stuck at fixed value.

Example

```
module test(in, out);
    input in;
    output reg out;
    wire tmp=1'b1;//signal ""tmp"" stuck at value 1'b1
    assign out=tmp&in;
endmodule
```

3.2.7 M007: no return value found for function

Warning Message

WARN_MEDIUM(M007 Simulation Issue) no return value found for function

Severity

Medium, should fix before start simulation

Description

Check return value of functions, all functions should return values.

Example

```
module test ( in, out);
    input in;
    output [7:0] out;
    out = func1(in);
    function [7:0] func1;
        input   a;
        if(a) //when a=0, function does not have a return value
            func1=7'hff;
```



```
        endfunction
    endmodule
```

3.2.8 M008: case label should not larger than max value of case variable

Warning Message

WARN_MEDIUM(M008 Simulation Issue) case label should not larger than max value of case variable

Severity

Medium, should fix before start simulation

Description

Check case label larger the max value of case variable.

Example

```
module test(in1,out1);
      input [1:0] in1;
      output [15:0] out1;
  reg [15:0] out1;
      always @(in1 or in2 or in3 or in4)
        if((in1+in2)<(in3+in4))//width mismatch found in operation, OPERATION_LEFT_WIDTH(9) != OPERATION_RIGHT_WIDTH(16) in operation
          out1='b0;
    else
          out1='b1;
endmodule
```

3.2.9 M009: conditional expression should not all constant value

Warning Message

WARN_MEDIUM(M009 Simulation Issue) conditional expression should not all constant value

Severity

Medium, should fix before start simulation

Description

Check all constant value conditional expression

Example

```
module test(in, out);
    output reg out;
    input reg in;

    always @( in )
        for (i=0;1>=0;i=i+1) // conditional expression should not all constant
value
        out = in;
endmodule
```

3.2.10 M010: constant cannot be used as event control condition

Warning Message

WARN_MEDIUM(M010 Simulation Issue) constant cannot be used as event control condition

Severity

Medium, should fix before start simulation

Description

Check to find constant used as event control condition

Example

```
module test(in, out);
    output reg out;
    input reg in;
    parameter PARAM=8;

    always @( PARAM ) // constant cannot be used as event control condition
        out = in;
endmodule
```


3.2.11 M011: event expression should not use "|", "||", please replace it with "or"

Warning Message

WARN_MEDIUM(M011 Simulation Issue) event expression should not use "|", "||", please replace it with "or"

Severity

Medium, should fix before start simulation

Description

Check "|" , "||" used in an event expression. Should replace with 'or'.

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input clk,rst;
    always @( posedge clk | nedgedge rst ) //event expression should not use ""|"", ""||"", please replace it with ""or""
    if( !rst )
        out <= 1'b0;
    else
        out <= !in;
endmodule
```

3.2.12 M012: found shifted out signal

Warning Message

WARN_MEDIUM(M012 Simulation Issue) found shifted out signal

Severity

Medium, should fix before start simulation

Description

Check all bits shifted out in shift operation

Example

```
module test(in1, in2, out1);
```



```
    input in1,in2;
    output out1;
    reg[15:0] reg1,reg2;
    out1=in1&in2;
    reg2 = reg1 << 16;//Warning 16 bits of reg1 all shifted out
endmodule
```

3.2.13 M013: last case label is not default

Warning Message

WARN_MEDIUM(M013 Simulation Issue) last case label is not default

Severity

Medium, should fix before start simulation

Description

Check to see last case label is not default

Example

```
module test(in,out);
    output reg [2:0] out;
    input [2:0] in;
    always @(in)
        case (in)
            0: out = 'd0;
            1: out = 'd1;
            2: out = 'd2;
            default: out = 0;//last case label is not default
            3: out = 'd3;
        endcase
endmodule
```

3.2.14 M014: operation on value "X/Z" or assign directly by value "Z" are not recommended

Warning Message

WARN_MEDIUM(M014 Simulation Issue) operation on value "X/Z" or assign

directly by value "Z" are not recommended

Severity

Medium, should fix before start simulation

Description

Check operation performed on value "X/Z" or assign directly by value "Z", it will reduce simulation mismatch between pre-synthesis and post-synthesis.

Example

```
module test(in,out,); //Warning: null port detected in module definition
    input  in;
    output out;
    wire tmp1, tmp2;
    assign tmp1=in & 1'bx;//Warning(simulation mismatch): operation on
                          value ""X/Z"" or assign directly by value ""Z""
                          are not recommended
    assign tmp2=1'bz; //Warning(simulation mismatch): operation on value
                      ""X/Z"" or assign directly by value ""Z"" are not
                      recommended
    assign out=in & tmp1 & tmp2;
endmodule
```

3.2.15 M015: signal in sensitivity list changed in its block

Warning Message

WARN_MEDIUM(M015 Simulation Issue) signal in sensitivity list changed in its block

Severity

Medium, should fix before start simulation

Description

Check sensitivity signal to make sure its not changed in the same block

Example

```
module test(in1,in2,out);
    output reg out;
    input reg in1,in2;
```



```
always @( in1 or in2 )
    in1=0;//Warning: signal in sensitivity list changed in its block
    out=in2;
endmodule
```

3.2.16 M016: logical operators (&&, ||) used on vectors

Warning Message

WARN_MEDIUM(M016 Simulation Issue) logical operators (&&, ||) used on vectors

Severity

Medium, should fix before start simulation

Description

Check operands of single-bit logic operator (&&, ||), vectors are illegal operands. For expression "OP1 && OP2", recommended style is "(|OP1) && (|OP2).

Example

```
module test(in1,in2,out);
    output  out;
    input [1:0] in1,in2;
    always @( in1 or in2 )
        if (in1 && in2) //logical operators (&&, ||)  used on vectors
          out = 1;
endmodule
```


3.3 Hardware Bug

3.3.1 H001: buffered clock detected

Warning Message

WARN_HIGH(H001 Hardware Bug) buffered clock detected

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check design to find buffered clock.

Example

```
module test ( in, clk, rst, out );
    input clk, rst, in;
    output reg out;
    wire clk_buf;
    buf buf_instance(clk_buf, clk);//buffered clock detected
    always @( posedge clk_buf or posedge rst )
        if ( rst )
            out <= 1'b0;
        else
            out <= in;
endmodule
```

3.3.2 H002: combinational loop detected

Warning Message

WARN_HIGH(H002 Hardware Bug) combinational loop detected

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check combinational loop in design.

Example

```
module test;
    wire a,b,c;
    assign b = !c;
    assign c = a & b; //warning on ""c->c->c""
endmodule
```

3.3.3 H003: edge trigger logic is not allowed inside task

Warning Message

WARN_HIGH(H003 Hardware Bug) edge trigger logic is not allowed inside task

Severity

High, must fix, some may not found by simulation, but will introduce hardware bugs.

Description

Check tasks, edge trigger is not allowed inside tasks.

Example

```
task task1;
    input clk,in;
    output reg out;
    always @( posedge clk ) //edge trigger logic is not allowed in task process.
        out <= !in;
endtask
```

3.3.4 H004: non-constant shift amount is not allowed

Warning Message

WARN_HIGH(H004 Hardware Bug) non-constant shift amount is not allowed

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check non-constant shift amount.

Example

```
module test ( in1,in2, clk, out );
    input clk;
    input [7:0] in1;
    input [2:0] in2;
    output reg [7:0] out;
    always @( posedge clk)
        out = in1<< in2; //""in2"" is not a static constant
endmodule
```

3.3.5 H005: recursive function/task call found

Warning Message

WARN_HIGH(H005 Hardware Bug) recursive function/task call found

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check recursive function or task call

Example

```
module test ( );
    function [7:0] func1;
        input [7:0] a;
        input [7:0] b;
        func1 = func2(a,b);//recursive function call func1-func2-func1-...
    endfunction

    function [7:0] func2;
        input [7:0] a;
        input [7:0] b;
        func2 = func1(a,b);//recursive function call func1-func2-func1-...
    endfunction
endmodule
```


3.3.6 H006: reset is not allowed driven by combinational logic

Warning Message

WARN_HIGH(H006 Hardware Bug) reset is not allowed driven by combinational logic

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check whether reset signal driven by combinational logic.

Example

```
module test ( in, in_rst, in_en, clk, out );
    input clk, in_rst, in_en,in;
    output reg out;
    wire rst;
    assign rst=in_rst & in_en;//reset is not allowed driven by combinational logic
    always @( posedge clk or posedge rst )
        if ( rst )
            out <= 1'b0;
        else
            out <= in;
endmodule
```

3.3.7 H007: set is not allowed driven by combinational logic

Warning Message

WARN_HIGH(H007 Hardware Bug) set is not allowed driven by combinational logic

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check whether set signal driven by combinational logic.

Example

```
module test ( in, in_set, in_en, clk, out );
    input clk, in_set, in_en,in;
    output reg out;
    wire set;
    assign set=in_set & in_en;//set is not allowed driven by combinational logic
    always @( posedge clk or posedge set )
        if ( set )
            out <= 1'b1;
        else
            out <= in;
endmodule
```

3.3.8 H008: set/reset signal found both active high and low

Warning Message

WARN_HIGH(H008 Hardware Bug) set/reset signal found both active high and low

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check set/reset signal active level, its not allowed to be both active high and low.

Example

```
module test ( in, rst, clk, out1,out2 );
    input clk, rst,in;
    output reg out1, out2;
    always @( posedge clk or posedge rst )
        if ( rst )
            out1 <= 1'b0;
        else
            out1 <= in;
```



```
always @( posedge clk or nedgedge rst ) //set/reset signal found both active high and low
    if ( !rst )
        out2 <= 1'b0;
    else
        out2 <= !in;
endmodule
```

3.3.9 H009: heavy fanout found

Warning Message

WARN_HIGH(H009 Hardware Bug) heavy fanout found

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check fanout for all siganls.

Example

```
module test ( in, out );
    input in;
    output out;
    wire tmp1,tmp2,tmp3,tmp4,tmp5,tmp6,tmp7;
    assign tmp1=in;
    assign tmp2=in;
    assign tmp3=in;
    assign tmp4=in;
    assign tmp5=in;
    assign tmp6=in;
    assign tmp7=in;//heavy fanout for signal ""in""
    assign out=func_test(tmp1,tmp2,tmp3,tmp4,tmp5,tmp6,tmp7)
endmodule
```


3.3.10 H010: no load found for current signal

Warning Message

WARN_HIGH(H010 Hardware Bug) no load found for current signal

Description

Check load for all internal signals. Internal siganl without load should be removed.

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Example

```
module test ( in, out );
    input in;
    output out;
    wire tmp;
    assign tmp=!in;//""tmp"" has no load
    assign out=!in;
endmodule
```

3.3.11 H011: variable not initialized in combinational process

Warning Message

WARN_HIGH(H011 Hardware Bug) variable not initialized in combinational process

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check to see if there is any variable which is referenced before initialized in combinational process, which will result in additional storage after synthesis

Example

```
module test( in, out );
    input in;
```



```
    output reg out;
    reg tmp;
    always @ ( in )
    begin
        if ( !in )
            out = tmp;
        out = in;
    end
endmodule
```

3.3.12 H012: variable not initialized in edge-trigger process

Warning Message

WARN_HIGH(H012 Hardware Bug) variable not initialized in edge-trigger process

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check to see if there is any variable which is referenced before initialized in edge-trigger process, which will result in additional storage after synthesis

Example

```
module test ( in, set, clk, out );
    input clk, set;
    input reg [1:0] in;
    output reg [1:0] out;
    reg [1:0] tmp;
    always @( posedge clk or posedge set )
        if ( set )
            out <= tmp; //""tmp"" is not initialized
        else
            out <= in;
endmodule
```


3.3.13 H013: multi driver found for some siganls

Warning Message

WARN_HIGH(H013 Hardware Bug) multi driver found for some siganls

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Scan all siganls in design under all possible conditions. Detect all possible multi-driver for all signals.

Example

```
module test( in1, in2, out );
    input in1, in2;
    output out;
    assign out=in1;
    assign out=in2;//multi driver found for ""out""
endmodule
```

3.3.14 H014: asynchronous loop detected

Warning Message

WARN_HIGH(H014 Hardware Bug) asynchronous loop detected

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check asynchronous loop

Example

```
module test(clk,in1, in2, out1);
    input clk;
    input reg[15:0] in1;
    output reg[15:0] out1;
    reg[15:0] in1,out1;
```



```
    reg sync_rst;
    always @( out1 )//out1->sync_rst->out1
        if ( out1 == in1 )
            sync_rst = 1;
        else
            sync_rst = 0;

    always @(posedge clk or posedge sync_rst)
        if ( sync_rst )
            out1 < = 0;
        else
            out1 < = out1 + 1;
endmodule
```

3.3.15 H015: bit range signal should not be used in sensitivity list

Warning Message

WARN_HIGH(H015 Hardware Bug) bit range signal should not be used in sensitivity list

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check bit range signal used in sensitivity list

Example

```
module test(in,out);
    output reg [2:0] out;
    input [2:0] in;
    always @(in[1]) //bit range signal should not be used in sensitivity list
        out=in;
endmodule
```

3.3.16 H016: blocking assignment cannot be used in edge triggered

blocks

Warning Message

WARN_HIGH(H016 Hardware Bug) blocking assignment cannot be used in edge triggered blocks

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check blocking assignment cannot be used in edge trigger blocks. This coding style may cause mismatch between pre and post-synthesis simulation

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input clk,rst;
    always @( posedge clk or nedgedge rst )
    if( !rst )
        out <= 1'b0;
    else
        out = in;//blocking assignment cannot be used in edge triggered
                 blocks
endmodule
```

3.3.17 H017: clock signal used as data input or reset detected

Warning Message

WARN_HIGH(H017 Hardware Bug) clock signal used as data input or reset detected

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check clock used as a data input or reset

Example

```
module test(clock_in, out);
    input clock_in;
    output out;

    always @(posedge clock_in)
        y1 = clock2;//clock signal used as data input detected
endmodule
```

3.3.18 H018: constant value found connected to Instance port

Warning Message

WARN_HIGH(H018 Hardware Bug) constant value found connected to Instance port

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check to find constant value connected to instance port

Example

```
module test(in,out);
    input in;
    output out;
    or or_instance(in,0,out);//constant value found connected to Instance port
endmodule
```

3.3.19 H019: divisor in division or modulo must be a constant

Warning Message

WARN_HIGH(H019 Hardware Bug) divisor in division or modulo must be a constant

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check divisor(second operand) in division or modulo, it must be a constant, otherwise cannot be synthesized.

Example

```
module test(in1, in2, out1,out2);
    input [1:0] in1, in2;
    output reg [2:0] out1,out2;
    always @(in1 or in2)
        out1 = in1 / in2 ; // divisor in division or modulo must be a constant
        out1 = in1 % in2 ; //divisor in division or modulo must be a constant
endmodule
```

3.3.20 H020: incomplete sensitivity list found

Warning Message

WARN_HIGH(H020 Hardware Bug) incomplete sensitivity list found

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check signals missed in sensitivity list.

Example

```
module test(in1,in2,out);
    output out;
    input in1,in2;
    always @(in1) //incomplete sensitivity list found, in2 should also be
included here
        out=in2+in2;
endmodule
```


3.3.21 H021: module without output found

Warning Message

WARN_HIGH(H021 Hardware Bug) module without output found

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check whether there is output for a moudle

Example

```
module test(in);//module without output found
    input in;
    wire tmp;
    tmp=!in;
endmodule
```

3.3.22 H022: no driver signal found

Warning Message

WARN_HIGH(H022 Hardware Bug) no driver found for signal

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check internal and output signals to make sure drivers found for them.

Example

```
module test(clk,set,in,out,out2);
    output reg out;
    input reg in;
    input set;
    input clk;
```



```
    wire tmp;//Warning: no driver signal found
    always @( posedge clk or posedge set )
        if( set )
            out <= in;
        else
            out <= tmp;
endmodule
```

3.3.23 H023: non-constant value found for loop count

Warning Message

WARN_HIGH(H023 Hardware Bug) non-constant value found for loop count

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Non-constant loop count are not recommended.

Example

```
module test(clk,in, out);
    input clk;
    input reg [1:0] in;
    output reg [7:0] out;
    integer i;
    always @(posedge clk)
        for(i=0;i<in;i++)//Warning(Bad-style): non-constant value found for
loop count
            out[i]<=out[i+1]
endmodule
```

3.3.24 H024: none-blocking assignment cannot be used in combinational blocks

Warning Message

WARN_HIGH(H024 Hardware Bug) none-blocking assignment cannot be used

in combinational blocks

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check none-blocking assignment cannot be used in combinational blocks. This coding style may cause mismatch between pre and post-synthesis simulation

Example

```
module test(in,out);
    output out;
    input in;
    always @(in)
        out<=in; //none-blocking assignment cannot be used in combinational blocks
endmodule
```

3.3.25 H025: none-constant case label detected

Warning Message

WARN_HIGH(H025 Hardware Bug) none-constant case label detected

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check all case labels and detect none-constant ones.

Example

```
module test(in,out);
    output reg [2:0] out;
    input [1:0] in;
    wire [1:0] tmp;
    assign tmp=in;
    always @(in)
        case (in)
```



```
            0: out = 'd0;
            1: out = 'd1;
            2: out = 'd2;
            tmp: out = 'd3;//Warning: none-constant case label detected
        endcase
endmodule
```

3.3.26 H026: null port detected in module definition

Warning Message

WARN_HIGH(H026 Hardware Bug) null port detected in module definition

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check null port used in module definition.

Example

```
module test(in,out,); //Warning: null port detected in module definition
    input  in;
    output out;
    assign out=!in;
endmodule
```

3.3.27 H027: only single bit clock is allowed

Warning Message

WARN_HIGH(H027 Hardware Bug) only single bit clock is allowed

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check multi-bit signal used as clock.

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input rst;
    input [1:0]clk;
    always @( posedge clk or nedgedge rst ) //only single bit clock is allowed
    if( !rst )
        out <= 1'b0;
    else
        out <= !in;
endmodule
```

3.3.28 H028: polarity of asynchronous set/reset signal mismatch in condition and sensitive list

Warning Message

WARN_HIGH(H028 Hardware Bug) polarity of asynchronous set/reset signal mismatch in condition and sensitive list

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check polarity of asynchronous set/reset signal, makre sure no mismatch found in condition and sensitive list

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input rst;
    input clk;
    always @( posedge clk or nedgedge rst )
    if( rst1 ) //Warning: polarity of asynchronous set/reset signal mismatch in condition and sensitive list
        out <= 1'b0;
```



```
    else
        out <= !in;
endmodule
```

3.3.29 H029: redundant asynchronous signal found in sensitive list

Warning Message

WARN_HIGH(H029 Hardware Bug) redundant asynchronous signal found in sensitive list

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check redundant asynchronous signal in sensitive list.

Example

```
module test(clk,rst1,rst2,in,out);
    output reg out;
    input reg in;
    input rst1,rst2;
    input clk;
    always @( posedge clk or nedgedge rst1 or negedge rst2 )
//Warning(nonsynthesizable): redundant asynchronous signal found in
sensitive list
    if( !rst1 )
        out <= 1'b0;
    else
        out <= !in;
endmodule
```

3.3.30 H030: set or reset detected used as data signal

Warning Message

WARN_HIGH(H030 Hardware Bug) set or reset detected used as data signal

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check set or reset used as a data signal.

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input rst;
    input clk;
    wire tmp;
    tmp=rst;//Warning: set or reset detected used as data signal
    always @( posedge clk or nedgedge rst )
    if( !rst1 )
        out <= 1'b0;
    else
        out <= !in;
endmodule
```

3.3.31 H031: set/reset should not be used as synchronous and asynchronous simultaneously in the same module

Warning Message

WARN_HIGH(H031 Hardware Bug) set/reset should not be used as synchronous and asynchronous simultaneously in the same module

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check set and reset signal to make sure they are not used as synchronous and asynchronous simultaneously in the same module

Example

```
module test(clk,rst,in,out,out2);
```



```
    output reg out;
    input reg in;
    input rst;
    input clk;
    always @( posedge clk or posedge rst )//Warning: set/reset should not be
used as synchronous and asynchronous simultaneously in the same module
        if( rst )
            out <= 1'b0;
        else
            out <= !in;

    always @( posedge clk )//Warning: set/reset should not be used as
synchronous and asynchronous simultaneously in the same module
        if( rst )
            out2 <= 1'b0;
        else
            out2 <= in;
endmodule
```

3.3.32 H032: signal assigned with both blocking and non-blocking expression

Warning Message

WARN_HIGH(H032 Hardware Bug) signal assigned with both blocking and non-blocking expression

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check signals assigned with both in blocking and nonblocking expressioin, it will reduce simulation mismatch between pre-synthesis and post-synthesis.

Example

```
module test(in1,in2,out);
    output reg out;
    input reg in1,in2;
    always @( in1 or in2 )
        if( in1 )
```



```
            out = 1'b0; //signal assigned with both blocking and non-
blocking
        else
            out <= in2; //signal assigned with both blocking and non-
blocking
endmodule
```

3.3.33 H033: synthesizable issue found, edge and non-edge expression mixed in sensitive list

Warning Message

WARN_HIGH(H033 Hardware Bug) synthesizable issue found, edge and non-edge expression mixed in sensitive list

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check synthesizable issue, edge and non-edge expression mixed in sensitive list, it will result in synthesis issue.

Example

```
module test(clk,rst,in,out);
    output reg out;
    input reg in;
    input clk,rst;
    always @( posedge clk or rst ) //edge and non-edge expression mixed in
sensitive list
    if( rst )
        out <= 1'b0;
    else
        out <= in;
endmodule
```


3.3.34 H034: Non-constant delay found

Warning Message

WARN_HIGH(H034 Hardware Bug) Non-constant delay found

Severity

High, must fix, some may not be found by simulation, but will introduce hardware bugs.

Description

Check to see if there is any delay value that is not a constant.

Example

```
module test(clk, in, out);
    input clk, in;
    output reg out;
    reg [7:0] delay;
    initial delay = 0;
    always #4 delay = delay+1;
    always @(posedge clk)
        q = #delay q_nxt;       //""delay"" is not static constant
endmodule
```


4 AutoBench

4.1 AutoBench Introduction

4.1.1 What is AutoBench

AutoBench is used to generate an UVM testbench automatically. It provides the base UVM class needed to quickly develop well-constructed and reusable UVM verification components and test environments.

4.1.2 Who are target users

Systemverilog is the most popular ASIC verification language, and UVM is a systemverilog class library which provides verification framework. UVM is very powerful but also hard to learn. Normally, DV owners with less than 5 years working experience, cannot handle UVM freely. These engineers are target users for “Autobench” generator.

4.1.3 How to use AutoBench

The only input of the tools is DUT top file. AutoBench provides following items automatically for UVM testbench:

- interface definition
- agents and a stimulus generation handle
- ready to run sanity test
- handles for virtual interface, clock and reset BFM
- demon Makefile for simulation

DV owners can configure single or multiple clocks and period of each clock separately. They can also add stimulus inside the sanity test generated. DV owners can add more tests after sanity test works.

The primary audience for AutoBench is DE or new DV owners who cannot build UVM testbench from scratch in a very short time.

4.2 Generate UVM testbench step by step

DV owners can start generating UVM testbench once RTL top is ready. To unblock DV owners’ work, DE can provide the very initial version of DUT top file, it only needs to include interface definitions.

Example, DUT top file is “adder.v”, users can following the step by step demon to generate an UVM testbench and simulation environment.

✧ Step 1. **Start AutoBench**: Click “AutoBench” button in IDE left panal.

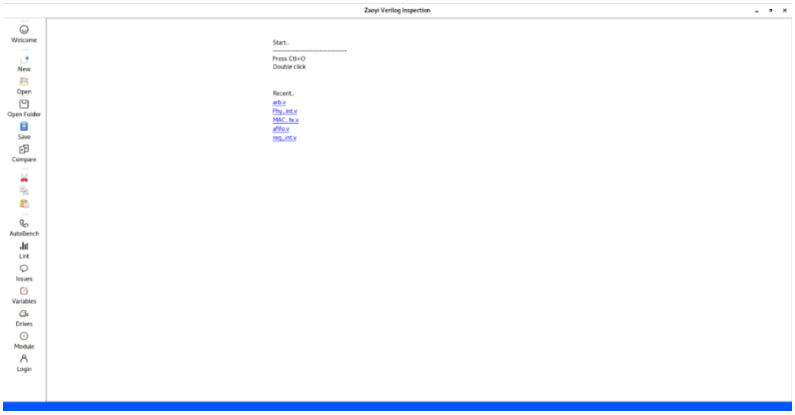


Figure 6 – Start AutoBench

✧ Step 2. **Select RTL**: Click “AutoBench” button in IDE view and choose RTL top file as expected then click “Open” button.

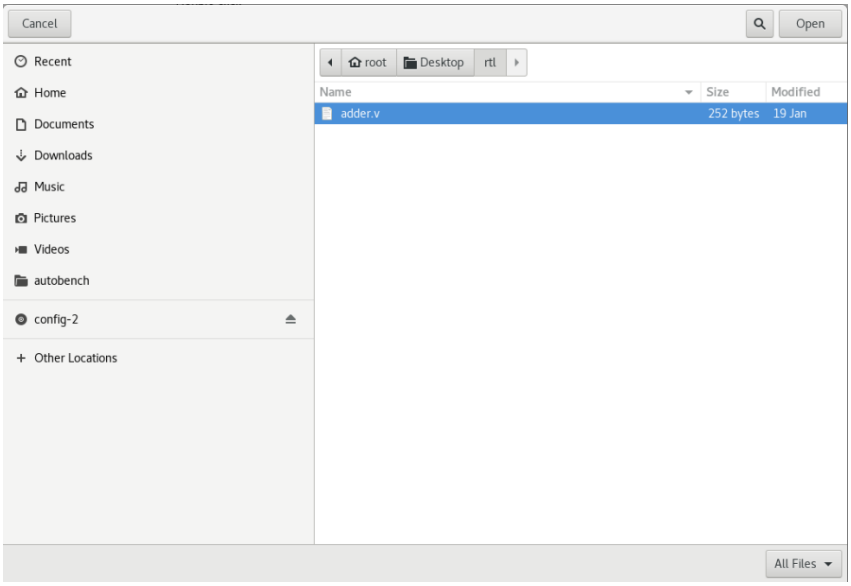


Figure 7 – Select RTL

✧ Step 3. **Generate UVM testbench**: If no RTL issues found in interface definition part, testbench, bringup test and regression will be generated.

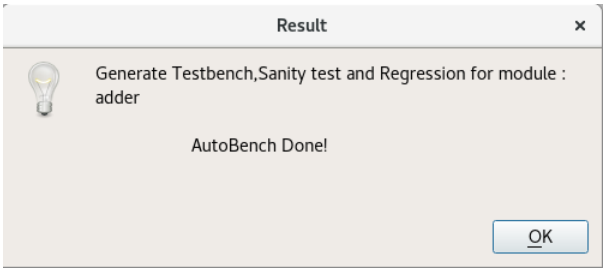


Figure 8 – Generate Testbench

✧ Step 4. **Testbenc Files**: Related testbench and test files can be found under your software

install director ($INSTAL_DIR):

- //$INSTAL_DIR/adder/tests/README.txt
- //$INSTAL_DIR/adder/tests/Makefile
- //$INSTAL_DIR/adder/tests/test_lib.sv
- //$INSTAL_DIR/adder/tests/filelist
- //$INSTAL_DIR/adder/tests/bringup.sv
- //$INSTAL_DIR/adder/tb/adder_wr_seq.sv
- //$INSTAL_DIR/adder/tb/adder_wr_agent.sv
- //$INSTAL_DIR/adder/tb/adder_rd_monitor.sv
- //$INSTAL_DIR/adder/tb/README.txt
- //$INSTAL_DIR/adder/tb/adder_rd_driver.sv
- //$INSTAL_DIR/adder/tb/adder_params.svh
- //$INSTAL_DIR/adder/tb/adder_intf.sv
- //$INSTAL_DIR/adder/tb/Makefile
- //$INSTAL_DIR/adder/tb/adder_env.sv
- //$INSTAL_DIR/adder/tb/adder_test.sv
- //$INSTAL_DIR/adder/tb/adder_rd_agent.sv
- //$INSTAL_DIR/adder/tb/adder_env_config.sv
- //$INSTAL_DIR/adder/tb/adder_test_defines.svh
- //$INSTAL_DIR/adder/tb/adder_wr_agent_config.sv
- //$INSTAL_DIR/adder/tb/adder_fc_subscriber.sv
- //$INSTAL_DIR/adder/tb/adder_intf_signals.sv
- //$INSTAL_DIR/adder/tb/adder_scoreboard_config.sv
- //$INSTAL_DIR/adder/tb/top_tb.sv
- //$INSTAL_DIR/adder/tb/adder_rd_trans.sv
- //$INSTAL_DIR/adder/tb/adder_seq_lib.sv
- //$INSTAL_DIR/adder/tb/adder_rd_agent_config.sv
- //$INSTAL_DIR/adder/tb/adder_scoreboard.sv
- //$INSTAL_DIR/adder/tb/adder_wr_monitor.sv
- //$INSTAL_DIR/adder/tb/adder_wr_trans.sv
- //$INSTAL_DIR/adder/tb/adder_pkg.sv
- //$INSTAL_DIR/adder/tb/adder_clk_reset.sv
- //$INSTAL_DIR/adder/tb/adder_rd_seq.sv
- //$INSTAL_DIR/adder/tb/adder_wr_driver.sv

✧ Step 5. **Compile**: There is a demon “Makefile” for simulation using synopsys VCS tool. When users run “Autobench”, it will generate Make file under Test directory.

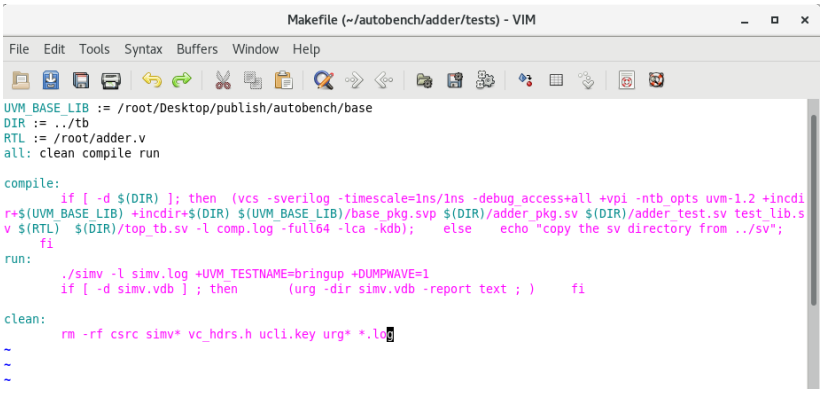


Figure 9 – Makefile Generated

✧ Step 6. **First Test**: After compile pass, UVM testbench is ready for simulation. Users do not need care about details of the testbench, only need to focus on add stimuli and compare data.

4.3 Simulation Demon

Demon RTL "adder.v" and testbench can be found under install directory. RTL is a very simple for "adder" function "result=i_a+i_b", showed in picture below.

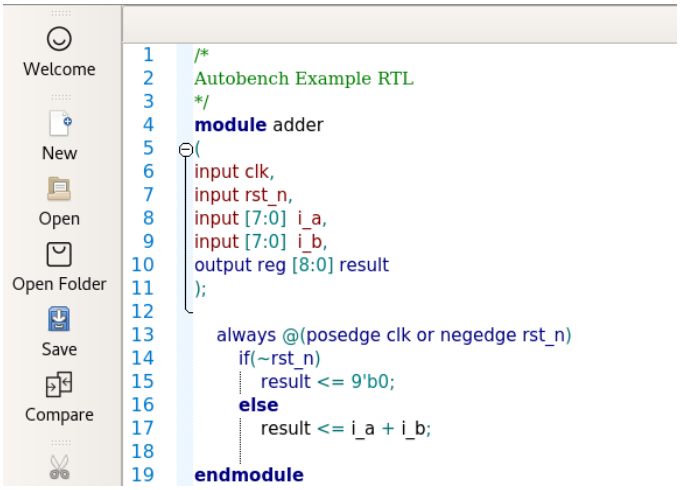


Figure 10 –Demon RTL "adder.v"

4.3.1 "TODO" items for simulation

After generating testbench followed guide of previous section, users can find a directory under Install directory named as "RTL_TOP" name. In this example, the testbench directory generated for "adder.v" is "adder".

Change directory to adder, and search "FIXME_TODO". After adding this information in testbench, users can get a regular testbench.


```
[root@VM_0_7_centos adder]# ls
tb  tests
[root@VM_0_7_centos adder]# grep -R FIXME_TODO *
tb/adder_rd_monitor.sv:            //FIXME_TODO: Collect result and send to scoreboard
tb/adder_rd_driver.sv:        //FIXME_TODO: set reset value of input signals
tb/adder_rd_driver.sv:    //FIXME_TODO: add stimuli to driver
tb/adder_intf.sv://FIXME_TODO: add interface definition here
tb/adder_fc_subscriber.sv:        //FIXME_TODO: add coverpoint definition here
tb/adder_intf_signals.sv:       //FIXME_TODO: Add the necessary data fields in trsactions to SB
tb/top_tb.sv:        adder    dut ();//FIXME_TODO: connect the interface
tb/adder_rd_trans.sv:    //FIXME_TODO: Add the necessary data fields in trsactions to DUT
tb/adder_scoreboard.sv:        //FIXME_TODO: option 1, for input monitor (default is agt1) only need to push tx to model
tb/adder_scoreboard.sv:        //FIXME_TODO: option 2, for output monitor (default is agt0) only need compare result
tb/adder_scoreboard.sv:        //FIXME_TODO: option 1, for input monitor (default is agt1) only need to push tx to model
tb/adder_scoreboard.sv:        //FIXME_TODO: option 2, for output monitor (default is agt0) only need compare result
tb/adder_scoreboard.sv:        //FIXME_TODO: put reference module in a serperate function or caculate here
tb/adder_wr_monitor.sv:            //FIXME_TODO: Collect result and send to scoreboard
tb/adder_wr_trans.sv:    //FIXME_TODO: Add the necessary data fields in trsactions to DUT
tb/adder_clk_reset.sv:            //FIXME_TODO: generate reset and clock signal signal
tb/adder_wr_driver.sv:        //FIXME_TODO: set reset value of input signals
tb/adder_wr_driver.sv:    //FIXME_TODO: add stimuli to driver
```

Figure 11 –Demon RTL adder.v

Here are files need to be updated.

File name	Useage	To do item in Demon Testbench
adder_rd_monitor.sv	Collect output at signal level, pack and send to scoreboard	Add signal names and transaction fields name.
adder_rd_driver.sv	Always used in read driver from passive ports, in this demon testbench it can be **ignored**.	No
adder_intf.sv	Define interface signals	add interface signals
adder_fc_subscriber.sv	Collect functional coverage	Add user defined cover points
adder_intf_signals.sv	Define transactions collected by monitors	Add transaction fields names
top_tb.sv	Testbench top	DUT instance signals
adder_rd_trans.sv	Read driver used transaction	No
adder_scoreboard.sv	1. Receive transaction and calculate expected output. 2. Receive output transaction and compare with expected output	Fill “compare” function
adder_wr_monitor.sv	Collect input at signal level, pack and send to scoreboard	Add signal names and transaction fields name.
adder_wr_trans.sv	Define transactions used by drivers	Add transaction fields names
adder_clk_reset.sv	Configure clock and reset	Add clock and reset name
adder_wr_driver.sv	Receive transaction and add stimuli to DUT	Add signal names and transaction fields name.

Table 7 - Testbench files

4.3.2 Simulation result

The picture below shows simulation result.

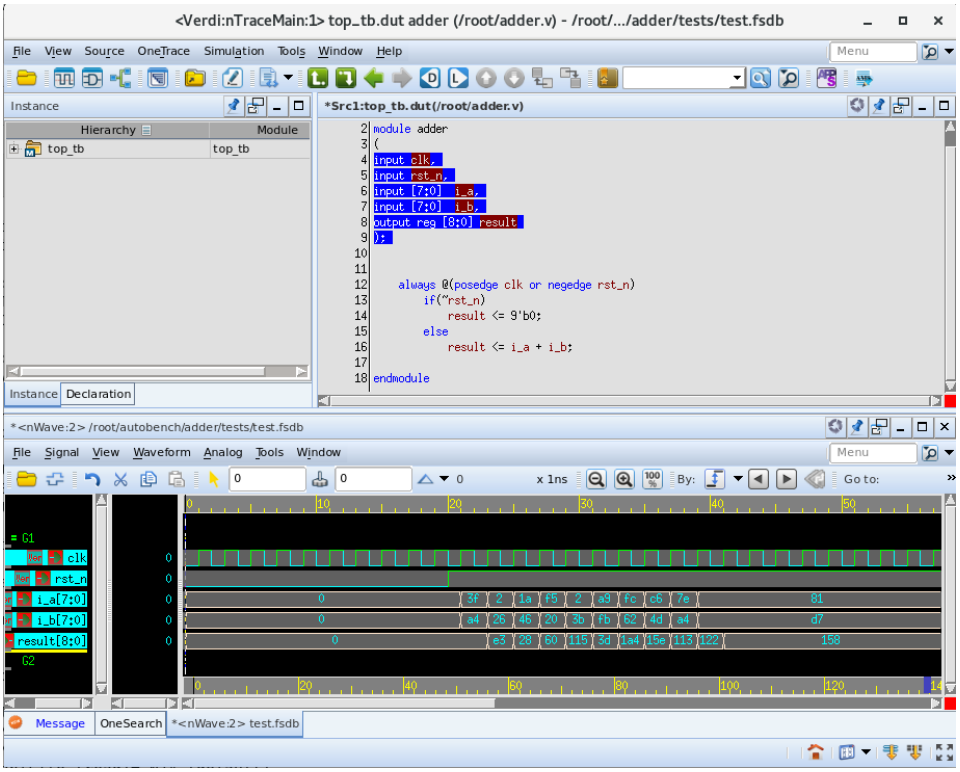


Figure 12 –Demon RTL adder.v

Ten transactions send by the demon test after reset. It can be found result match expected(i_a+i_b).

4.3.3 Coverage report

Figure below shows the functional coverage result in the demon RTL.

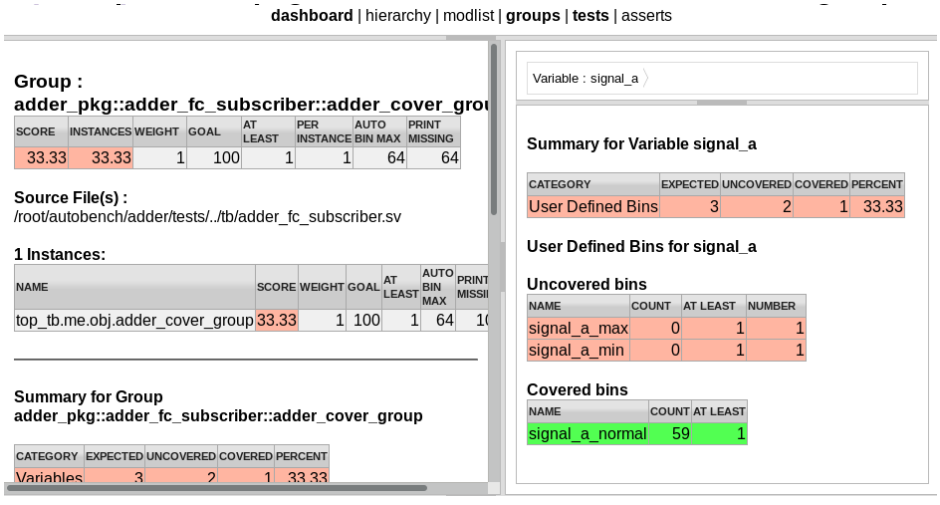


Figure 13 –Demon RTL adder.v

Three bins defined for input i_a(signal_a):

- siangal_a_min : i_a=0;
- signal_a_normal : 1<i_a<255;
- signal_a_max : i_a=255;

Fully random input "i_a" does not touch "siangal_a_min" and "signal_a_max" scenario, so result is 33.33%, which match expectation.

4.4 Class reference

"AutoBench" generate testbench on base classes($INSTAL_DIR/autobench/base) provided by the software. Normally, DV owners will use this tool to verify different modules at the same time, base class should be included in each simulation. Users can also set up environment variable as "*set AUTOBENCH_BASE_LIB=$INSTAL_DIR/autobench/base*".

Base class on this layer is not visible to users. DV owners can modify subclass generated from base class and RTL top to get all flexibility they want. Normally, DV owners do not need to understand how base class works and how it connects to subclass generated. So, we only list important classes on this layer for better understanding of the "AutoBench" methodology.

Following sections list contents (important Variables and functions) of base classes.

- $AutoBench./base/README.txt
- $AutoBench./base/Makefile
- $AutoBench./base/base_env_config.svp
- $AutoBench./base/base_agent_config.svp
- $AutoBench./base/base_driver.svp
- $AutoBench./base/base_env.svp
- $AutoBench./base/base_scoreboard.svp
- $AutoBench./base/base_scoreboard_config.svp
- $AutoBench./base/base_macros.svh
- $AutoBench./base/base_agent.svp
- $AutoBench./base/base_fc_subscriber_config.svp
- $AutoBench./base/base_transaction.svp
- $AutoBench./base/base_fc_subscriber.svp
- $AutoBench./base/base_test.svp
- $AutoBench./base/base_pkg.svp
- $AutoBench./base/base_clk_reset.svp
- $AutoBench./base/base_clk_reset_config.svp
- $AutoBench./base/base_sequencer.svp
- $AutoBench./base/base_monitor.svp

4.4.1 base_test

4.4.1.1 summary

This class is the base class for the AutoBench generated tests.

The base_test class is used as the base for generated tests. After UVM testbench generated, AutoBench tool will generate another test "RTLNAME_test" based on this. RTLNAME_test keeps all flexibility users needed to configure the testbench environment. Then, based on RTLNAME_test, users can develop their own tests for different scenarios.

Deriving from base_test provides the ability to configure everything the test may need but hide lower-level details. base_test is provided with the installation package, and not allowed to be modified.

4.4.1.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_test
base_test
RTLNAME_test
bringup

4.4.1.3 Methods

```
function new(string name="base_test",uvm_component parent=null);
```

Creates a new instance of this class using the normal constructor arguments for uvm_test. Parent is the handle of the hierarchical parent.

```
extern virtual function void build_phase(uvm_phase phase);
```

Creates env_config, env and report catcher inside build_phase.

```
extern virtual function void end_of_elaboration_phase(uvm_phase phase);
```

Set up timeout and drain time.

```
extern virtual function void report_phase(uvm_phase phase);
```

Process all messages after simulation finished, including all kinds of warnings, and errors.

```
function void set_timeout(time t);
```

Set simulation time out.

4.4.2 base_env_config

4.4.2.1 summary

This class is used for environment configuration. Following items are supported:

- num_of_agents: number of agents needed for current testbench;
- has_scoreboard: whether scoreboard is needed for current testbench;
- has_functional_coverage : whether needs to collect functional coverage;
- has_clk_rst: whether clk_reset is needed.

4.4.2.2 Class Hierarchy

uvm_void
uvm_object
base_env_config
RTLNAME_env_config

4.4.2.3 Methods

function new(string name="base_env_config")

Creates a new instance of this class.

function void set_num_of_agents(int unsigned a)

Set number of agents needed for current testbench.

function void set_has_scoreboard(bit a=1)

Set whether scoreboard is needed for current testbench. Default is adopting scoreboard.

function void set_has_functional_coverage(bit a=1)

Set whether needs to collect functional coverage. Default is collecting functional coverage.

function void set_has_clk_rst(bit a=1)

Set whether clk_reset is needed. Default is need clock and reset.

4.4.1 base_env

4.4.1.1 summary

This class is actually a hierarchical container of other components. It can be reused while building higher level testbench environment.

4.4.1.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_env
base_env
RTLNAME_env

4.4.1.3 Methods

function new(string name="base_env",uvm_component parent = null);

Creates a new instance of this class and initialize it inside this function.
name: name registered;
uvm_component: named following this instance, parent is the handle to the hierarchical parent.

function void build_phase(uvm_phase phase);

Creates **agents**, **scoreboard**, **subscriber**(collect functional coverage) and clk_reset according to the setting of **base_env_config**.

function void connect_phase(uvm_phase phase);

Connects **agents** to **scoredboard** and **agents** to **subscriber** ccording to the setting of **base_env_config**.

4.4.2 base_scoreboard_config

4.4.2.1 summary

This class is used for scoreboard configuration. Following items are supported:

- use_fifo: whether or not using FIFOs for transaction transfer.
- check_fifo_empty: when simulation finished, whether need to make sure all FIFOs are empty.
- compare_tx_cnt : number of transactions need to compare.
- num_of_analysis_exports_groups: number of analysis_export group used for data compare and collection functional coverage.
- expected_tx_cnt[]: expected transaction count for each analysis_export group.

4.4.2.2 Class Hierarchy

uvm_void
uvm_object
base_scoreboard_config

RTLNAME_scoreboard_config

4.4.2.3 Methods

function new(string name="base_scoreboard_config")

Creates a new instance of this class.

function void set_check_fifo_empty(bit a=1)

Set "check_fifo_empty=1" as default, before simulation finish. There are several transaction FIFOs inside scoredboard comes from driver, monitors. Setting "check_fifo_empty=1" means it needs make sure all data inside these FIFOs are cleared.

function void set_use_fifo(bit a =1)

Set "use_fifo=1" as default. Users could modify it in "*RTLNAME_scoreboard_config*" if no FIFO needed.

function void set_num_of_analysis_exports_groups(int a)

Set number of analysis_export group used for data compare and collection functional coverage.

function void set_expected_tx_cnt(int a, int b)

Set expected transaction count for each analysis_export group.

function void set_compare_tx_cnt(bit a=1)

Set number of transactions need to compare. Default is 1.

4.4.3 base_scoreboard

4.4.3.1 summary

This class is used to compare Data integrity between expected and real DUT output.

4.4.3.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_scoreboard
base_scoreboard
RTLNAME_scoreboard_config

4.4.3.3 Methods

function new(string name = "base_scoreboard",uvm_component parent);

Creates a new instance of this class.

function void build_phase(uvm_phase phase);

Create analyse_export and transaction count. Create FIFOs if used.

function void connect_phase(uvm_phase phase);

If adopting FIFOs, connect analyse_export to FIFO inside this function.

function void main_phase(uvm_phase phase);

Users need to do compare in main_phase. But should keep "base_scoreboard" main_phase clean and modify "*RTLNAME_scoreboard" main_phase.*

function void extract_phase(uvm_phase phase);

Compare transaction count and check whether FIFO is empty according to settings inside scoreboard_config.

4.4.4 base_agent_config

4.4.4.1 summary

This class is used for agent configuration. Following items are supported:

- has_monitor: whether monitor is needed for current agent.
- mode: master or slave mode
- is_active: active or passive.
- has_clk_rst: whether clk_reset is needed.

4.4.4.2 Class Hierarchy

uvm_void
uvm_object
base_agent_config
RTLNAME_agent_config

4.4.4.3 Methods

function void set_mode(MODE a =MASTER);

Set mode, default is MASTER mode.

function void set_is_active(uvm_active_passive_enum a = UVM_ACTIVE);

Set active or passive, default is active.

function void set_has_clk_rst(bit a=1);

Set whether clk_reset is needed. Default is need clock and reset.

4.4.1 base_agent

4.4.1.1 summary

This class is used as containers of four subcomponents: clk_reset, base_driver, base_sequencer, and base_monitor. There are two modes of this class: active and passive. For an active agent, it should contain all four types of subcomponents. For a passive agent it should only contain a base_monitor.

4.4.1.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_agent
base_agent
RTLNAME_agent

4.4.1.3 Methods

function new(string name="base_agent",uvm_component parent=null);

Creates a new instance of this class and initialize it inside this function.
name: name registered.
uvm_component: named following this instance, parent is the handle to the hierarchical parent.

function void build_phase(uvm_phase phase);

Creates analysis_ports for testbench.
Creates base_driver, base_sequencer and base_clk_reset for active agent.
Creates base_monitor according to configuration of **base_agent_config.has_monitor**.

function void connect_phase(uvm_phase phase);

For active agent, connect driver port to sequencer.
Connect base_monitor port according to configuration of **base_agent_config.has_monitor**.

4.4.2 base_fc_subscriber_config

4.4.2.1 summary

This class is used for subscriber configuration.

4.4.2.2 Class Hierarchy

uvm_void
uvm_object
base_fc_subscriber_config
RTLNAME_fc_subscriber_config

4.4.2.3 Methods

function new(string name="base_fc_subscriber_config");

Creates a new instance of this class.

function void set_num_of_fc_tx_fifos(int unsigned a=1);

Set number of transaction FIFOs for functional coverage collection, default is 1.

4.4.3 base_fc_subscriber

4.4.3.1 summary

This class is mainly used for collecting functional coverage. It provides an export for receiving transactions from a connected analysis export via a transaction FIFO.

4.4.3.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_subscriber
base_fc_subscriber
RTLNAME_fc_subscriber

4.4.3.3 Methods

function new(string name="base_fc_subscriber",uvm_component parent=null);

Creates a new instance of this class and initialize it inside this function.
“name”: name registered.

"uvm_component": named following this instance, parent is the handle to the hierarchical parent.

function void build_phase(uvm_phase phase);

Create analyse_export and FIFOs. Numbers of export and FIFOs are defined inside "base_fc_subscriber_config.num_of_fc_tx_fifos".

function void connect_phase(uvm_phase phase);

Connect analyse_export to FIFO export.

function void write(base_transaction t);

Write transaction action, should be redefined inside "*RTLNAME_scoreboard.write".*

function void main_phase(uvm_phase phase);

Collect functional coverage, should be redefined inside "*RTLNAME_scoreboard.main_phase".*

4.4.4 base_driver

4.4.4.1 summary

This class get new transactions from uvm_sequencer and translate to signal level stimuli to DUT.

4.4.4.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_driver#(REQ,RSP)
base_driver
RTLNAME_driver

4.4.4.3 Methods

function new(string name="base_driver",uvm_component parent);

Creates a new instance of this class and initialize it inside this function.
name: name registered.
uvm_component: named following this instance, parent is the handle to the hierarchical parent.

function void main_phase(uvm_phase phase);

Call tx_driver() task to receive transaction and send it out to DUT at signal level.

4.4.1 base_monitor

4.4.1.1 summary

This class get collect signal level signal from DUT and package to transaction then send to scoreboard via base_agent.

4.4.1.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_monitor
base_monitor
RTLNAME_monitor

4.4.1.3 Methods

function new(string name="base_monitor",uvm_component parent);

Creates a new instance of this class and initialize it inside this function.
name: name registered.
uvm_component: named following this instance, parent is the handle to the hierarchical parent.

function void main_phase(uvm_phase phase);

Call send_tx(), send_intf_signals() and send_fc_tx() tasks to collect signal level output and packet to transaction for compare purpose.

4.4.2 base_clk_reset_config

4.4.2.1 summary

This class is used for base_clk_reset configuration. Following items are supported:

- num_of_clks: clock number
- half_period: clock half period
- num_of_rsts : number of resets
- reset_active: reset active type.
- reset_hold_delay: hold delay of reset.

4.4.2.2 Class Hierarchy

uvm_void
uvm_object
base_clk_reset_config
RTLNAME_clk_reset_config

4.4.2.3 Methods

function new(string name="base_clk_reset_config")
Creates a new instance of this class.

function void set_num_of_clks(int a)
Set clock number.

function void set_num_of_resets(int a)
Set reset number.

function void set_clk_half_period(int a, real b)
Set clock half period.

function void set_reset_hold_delay(int a, int b)
Set hold delay of reset.

function void set_reset_active(int a, ACTIVE_TYPE b)
Set reset active type.

4.4.3 base_clk_reset

4.4.3.1 summary

This class is used to generate clock and reset according to configuration of the **base_clk_reset_config**.

4.4.3.2 Class Hierarchy

uvm_void
uvm_object
uvm_report_object
uvm_component
uvm_driver
base_clk_reset
RTLNAME_clk_reset

4.4.3.3 Methods

function new(string name="base_clk_reset",uvm_component parent);

Creates a new instance of this class and initialize it inside this function.

name: name registered.

uvm_component: named following this instance, parent is the handle to the hierarchical parent.

task generate_reset(int index , ref logic a);

Generate reset signal according to setting of **base_clk_reset_config**.reset_active and base_clk_reset_config.reset_hold_delay.

task generate_clock(int index,ref logic a);

Generate clock signal according to setting of **base_clk_reset_config**. half_period.

5 Appendix

NA

