FUNCTIONAL COVERAGE GENERATOR

Version 1.0

This tool is developed using PERL. It can be used in two ways:

- GUI Mode
- Normal (Batch) Mode

GUI Mode

To run in GUI mode, "*-g*" argument shall be provided with the command line (Command: *perl coverage_generator.pl -g*). The GUI looks like as follow:

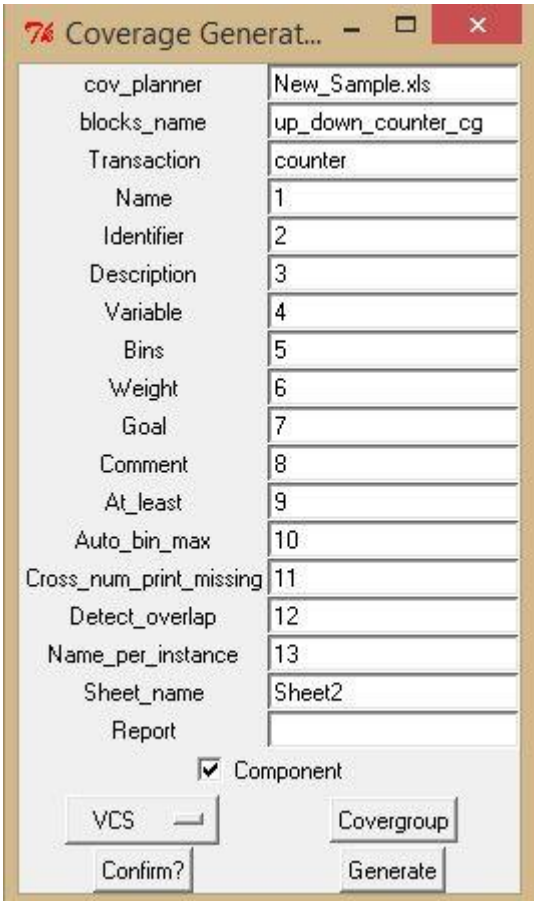

Figure 1 COVERAGE GENERATOR GUI

As shown in Figure 1 COVERAGE GENERATOR GUI, tool contains various text boxes to configure the tool and maps coverage plan with this tool. Multiple buttons are provided below text box to generate the functional coverage code or dump the coverage result. Below is the explanation of each input:

- **Text Box**:

Table 1 Text Box

Sr. Number	Text Box	Description
1	*cov_planner*	Name of the coverage plan document. With this tool, XLS and SpreadSheet XML format are supported. Relative path of the file is supported. Only single file shall be provided as input. Ex: coverage_planner.xls

2	*blocks_name*	Covergroups which are to be coded in the file. More than one covergroups can be provided separated by ",". These covergroups shall be mentioned in the coverage plan. Ex: i2c_covergroup, spi_covergroup
3	*Transaction*	Name of the transaction class. It should be provided properly. Name of the generated functional coverage file will be <*Transaction Name*>*_coverage.sv*. Coverage class name will be <*Tranaction Name*>*_coverage*. In case of coverage component, Transaction class's instantiated as "<Transaction Class>_pkt". Ex: i2c, generates "i2c_coverage.sv" file and coverage component name (if generated) will be "i2c_coverage".
4	*Name*	Column number in the coverage plan where name of the covergroup/coverpoint/cross are mentioned.
5	*Identifier*	Column number in the coverage plan where identifier information is mentioned. Identifier indicates whether row contains information for covergroup or coverpoint or cross.
6	*Description*	Column number in the coverage plan where comments for the covergroup/coverpoint/cross is mentioned. It will be printed in the code as single line comment i.e. // Description. In case of more than 80 characters, the line will be wrapped automatically.
7	*Variable*	Column number in the coverage plan where sampling variable for coverpoint or cross is mentioned. Sampling condition can be provided with the variable name. For covergroup, arguments or sampling detail can be provided in this column.
8	*Bins*	Column number in the coverage plan where bins are written with respective to coverpoint or cross. Each bin shall be written in new line in the cell. Tool is supporting macros to generate bins and shall be written in this column. The macros' details is provided in Table 2 Supported Macros.
9	*Weight*	Column number in the coverage plan where weight option is mentioned for the covergroup, coverpoint and cross. If the cell is empty then "*option.weight*" will not be dumped in the coverage code else weight number will be dumped with "*option.weight*".
10	*Goal*	Column number in the coverage plan where goal option is mentioned for the covergroup, coverpoint and cross. If the cell is empty then "*option.goal*" will not be dumped in the coverage code else goal number will be dumped with "*option.goal*".
11	*Comment*	Column number in the coverage plan where comment option is mentioned for the covergroup, coverpoint and cross. If the cell is empty then "*option.comment*" will not be dumped in the coverage code else the mentioned comment will be dumped with "*option.comment*".
12	*At_least*	Column number in the coverage plan where *at_least* option is mentioned for the covergroup, coverpoint and cross. If the cell is empty then "*option.at_least*" will not be dumped in the coverage code else the numer will be dumped with "*option.at_least*" .
13	*Auto_bin_max*	Column number in the coverage plan where *auto_bin_max* option is mentioned for the covergroup and coverpoint. If the cell is empty then "*option.auto_bin_max*" will not be dumped in the coverage code. For cross, this option is not supported. So cell shall be empty for it.
14	*Cross_num_print_missing*	Column number in the coverage plan where cross_num_print_missing option is mentioned for the covergroup and cross. If the cell is empty then "*option.cross_num_print_missing*" will not be dumped in the coverage code. For coverpoint, this option is not supported. So cell shall be empty for it.
15	*Detect_overlap*	Column number in the coverage plan where detect_overlap option is mentioned for the covergroup and coverpoint. If the cell is empty then "*option.detect_overlap*" will not be dumped in the coverage code. For cross, this option is not supported. So cell shall be empty for it.

16	*Name_per_instance*	Column number in the coverage plan where *name* and *per_instance* options are mentioned for the covergroup. It contains two options. So format is, first character shall be "*per_instance*" (it's type is bit), second character shall be a separator and then "name" (it's type is string). "*per_instance*" is a bit, so first character is used for it and value shall be 0 or 1. Separation can be any character/digit/special character and will be ignored always. If cell is empty then "*option.name*" and "*option.per_instance*" will not be dumped in the coverage code. For coverpoint and cross, this options are not supported. So, cell shall be empty for them.
17	*Sheet_name*	Sheet name where the coverage plan is written in XLS or Spreadsheet XML.
18	*Report*	This text box is used when user wants to dump coverage report in the coverage plan. The input format shall be *<column_number>,<Report File>*. Column number indicates where the report information will be dumped in the coverage plan. This box shall be empty when user wants to generate functional coverage code. If this input is not empty then tool consider as coverage report dumping functionality.

- **Tick Box:**
  - *Component*: A tick box is shown below the text box. If tick mark is provided in the box then coverage code will be generated with coverage component otherwise only covergroup will be dumped in the file while generating coverage code.

- **Option Menu:**

Below the tick box an OptionMenu is provided for dumping the coverage result in the coverage plan. User has to provide the coverage result file in the "*Report*" text box and has to select the tool name from OptionMenu by which the report file is generated. Mostly the coverage report text file format is different among various tool as well as same tool with different versions. So it is difficult to support all of them. To handle this problem, "USER" option is provided in the OptionMenu. With this option, user has to implement logic in "*coverage_generator_user_result.pl*" file to grab the coverage result and assign it to the "%report"(hash). This variable shall be declared as "our" as it is a global variable and shared between the tool and "*coverage_generator_user_result.pl*" file. The implemented logic shall be in "*user_result*" subroutine and it has a single input argument (Report text file) which can e accessible using $ARGV[0]. The result shall be stored as "$report{'<covergroup_name>'}{'<coverpoint/cross_name>'}.

Ex: $report{'i2c_covergroup'}{'RW'} = 50.00;

Advantage of providing "USER" option is, user don't require to modify the code and can be saved in central data base.

- **Buttons:**

With OptionMenu, three buttons are provided.

  - *Covergroup*: It is very tedious task to write all covergroup name in the "*blocks_name*" text box. To avoid it, user can provide the coverage plan name in "*cov_planner*" text box and then click on this "*Covergroup*" button which a show a pop up which has all covergroups name (with tick box) written in the coverage plan. User can select the covergroups from it by doing tick mark against the covergroup name and then have to click on "*Selected*" button. Before clicking on this button, user has to make sure that all required text messages are filled properly.

Figure 2 Covergroup GUI

- *Confirm?*: After providing all required details, user has to click on this button to validate all inputs are proper and then user can generate coverage code or dump the coverage result in the coverage plan. When user click on this button the input details will be dumped in the "*coverage_generator_cfg.txt*" file. This tool doesn't require to do reconfiguration manually again. In case of reconfiguration by clicking on this "*Confirm?*" button, the last stored information is retrieved back into the GUI as a starting point and can generate the coverage code. User can also change the configuration in the text box. For reconfiguration, user has to make sure that, "*coverage_generator_cfg.txt*" file shall be in the same directory from where the coverage code was generated otherwise tool can't retrieve the old information. If "*coverage_generator_cfg.txt*" file is not present in the directory especially during first time usage, this file will be created automatically while clicking on "*Confirm?*" button.
- *Generate*: This button is used to generate the functional coverage code or dump the coverage result in the coverage plan. By default, this button is in disabled state. Once user click on the "*Confirm?*" button then the tool will validate the inputs and then this button will be enabled automatically.

Normal Mode

In case of Normal Mode, user don't need to provide "*-g*" option (Command: *perl coverage_generator.pl*). In this case, user has to update "*coverage_generator_cfg.txt*" file as per the given format and this file shall be in the same directory from where the perl command is applied. Information shown in the GUI format, same information shall be provided in the txt file and identification name shall be same. User input shall be separated by "=". Multiple covergroup names shall be separated with ",". For selecting "*Component*", user has to provide "*Component=1*" and in case of only covergroup code requires, user has to provide "*Component=0*" in the txt file. For generating functional coverage code, "*Report=*" shall be empty and for dumping coverage report, report file and column number shall be provided with it. Below is the snapshot of "*coverage_generator_cfg.txt*" file.

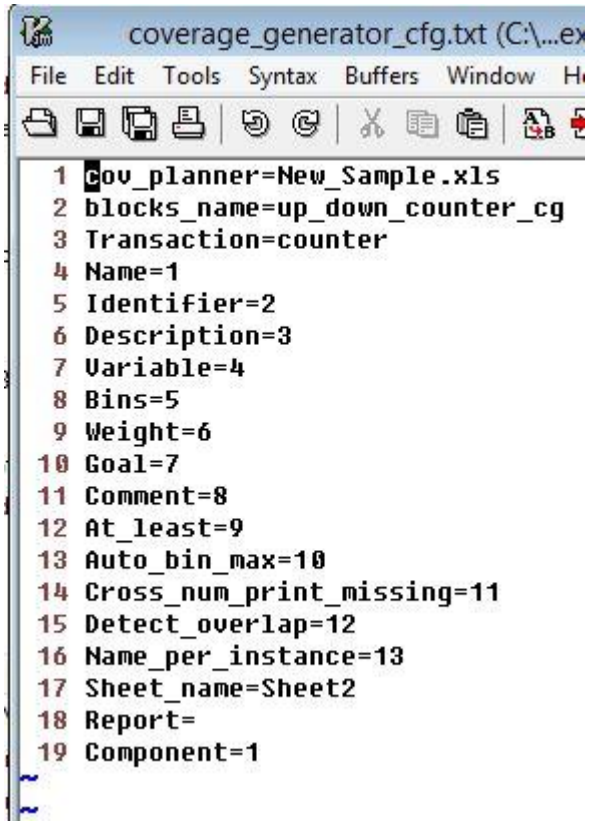

Figure 3 Coverage Generator CFG

To write the coverage bins easily, this tool is supporting in built macros. So with a single line, multiple bins can be generated and it requires very less time. These macros shall be written in *"Bins"* column. Below are the supported macros with this tool.

Table 2 Supported Macros

Macro Name	**Usage**
`GBINS_SIMPLE(BIN_1; BIN_2; …;BIN_N)	Generates simple bins using "bins" keyword. Multiple bins can be provided using ";". In single bin, multiple values can be provided using ",". Range can be provided by "..". Output of `GBINS_SIMPLE(0,4..7; 8;11..14) is: bins _sbin_1 = {0, [4:7]}; bins _sbin_2 = {8}; bins _sbin_3 = {[11:14]};
`GBINS_ARRAY(ARRAY_SIZE,BIN_1; ARRAY_SIZE, BIN_2; …;ARRAY_SIZE, BIN_N)	Generates array of bins. Each bin shall be separated by ";". For each bin, size shall be first argument and other arguments shall be bins values. Range can be provided by "..". Output of `GBINS_ARRAY(,1,3..5;4,2,6..12;3, 14..19;) is: bins _abin_1[] = {1, [3:5]}; bins _abin_2[4] = {2, [6:12]}; bins _abin_3[3] = {[14:19]};
`GBINS_WILDCARD(BIN_1; BIN_2; …;BIN_N)	Generates wildcard bins using "wildcard bins" keyword. Multiple bins can be provided using ";". In single bin, multiple values can be provided using ",". Output of `GBINS_WILDCARD(4'b11??); is: wildcard bins _wbin_1 = {4'b11??};
`GBINS_IGNORE(BIN_1; BIN_2; …;BIN_N)	Generates ignore bins using "ignore_bins" keyword. Multiple bins can be provided using ";". In single bin, multiple values can be provided using ",".Range can be provided by "..". Output of `GBINS_IGNORE(1,3; 8..11) is:

	ignore_bins _ibin_1 = {1, 3}; ignore_bins _ibin_2 = {[8:11]};
`GBINS_ILLEGAL(BIN_1; BIN_2; …;BIN_N)	Generates illegal bins using "illegal_bins" keyword. In single bin, multiple values can be provided using ",".Range can be provided by "..". Output of `GBINS_ILLEGAL(1,3; 8..11) is: illegal_bins _lbin_1 = {1, 3}; illegal_bins _lbin_2 = {[8:11]};
`GBINS_WALK_ONE(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates bins of walk one pattern. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ONE(12; 2;4..11) is: bins _pbin_1 = {12'b0000_0000_0001}; bins _pbin_3 = {12'b0000_0000_0100}; bins _pbin_12 = {12'b1000_0000_0000}; As shown above, first argument "12" indicates the size of the variable i.e. 12 bits. So total 12 bins shall be generated. But other argument 2 & 4..11 (4 to 11) bit position doesn't require. So for bit position 1, 3 and 12, bins are generated.
`GBINS_WALK_ONE_LW(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates wildcard bins of walk one pattern with ignore bit locations on LSB. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ONE_LW(12; 2;4..11) is: wildcard bins _pwbin_1 = {12'b0000_0000_0001}; wildcard bins _pwbin_3 = {12'b0000_0000_01??}; wildcard bins _pwbin_12 = {12'b1???_????_????};
`GBINS_WALK_ONE_MW(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates wildcard bins of walk one pattern with ignore bit locations on MSB. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ONE_MW(12; 2;4..11) is: wildcard bins _pwbin_1 = {12'b????_????_???1}; wildcard bins _pwbin_3 = {12'b????_????_?100}; wildcard bins _pwbin_12 = {12'b1000_0000_0000};
`GBINS_WALK_ZERO(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates bins of walk zero pattern. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ZERO (12; 2;4..11) is: bins _pbin_1 = {12'b1111_1111_1110}; bins _pbin_3 = {12'b1111_1111_1011}; bins _pbin_12 = {12'b0111_1111_1111};
`GBINS_WALK_ZERO_LW(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates wildcard bins of walk zero pattern with ignore bit locations on LSB. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ZERO_LW (12; 2;4..11) is: wildcard bins _pwbin_1 = {12'b1111_1111_1110};

	wildcard bins _pwbin_3 = {12'b1111_1111_10??}; wildcard bins _pwbin_12 = {12'b0???_????_????};
`GBINS_WALK_ZERO_MW(SIZE_IN_BITS; LIST_OF_BINS_TO_IGNORE)	Generates wildcard bins of walk zero pattern with ignore bit locations on MSB. Number of bins are selected based on argument. First argument indicates the size of the variable in bits. Other arguments indicates the bit location to be ignored. Range can be provided by "..". Location 0 indicates position 1. Only decimal values are allowed in this macro. Output of `GBINS_WALK_ZERO_MW (12; 2;4..11) is: wildcard bins _pwbin_1 = {12'b????_????_???0}; wildcard bins _pwbin_3 = {12'b????_????_?011}; wildcard bins _pwbin_12 = {12'b0111_1111_1111};

This tool is tested on "Windows8" and "Linux" machine. Perl 5.020001 is used to create and test this tool. This tool is also used on PERL version "<TOOL VERSION>".

To use this tool, following CPAN modules shall be installed in the system:

(1) Spreadsheet::ParseExcel
(2) Spreadsheet::WriteExcel
(3) Text::Wrap
(4) Tk
(5) XML::Simple
(6) Data::Dumper

If above modules are not installed in the system then they can be downloaded from https://www.cpan.org/ website.

XLS Format:

Sr. No	Name	Identifier	Description	Variable	Bins	Weight	Goal	Comment	name & per_instance
1	up_down_counter_cg	Covergroup	This covergroup contains information abour up-down counter block.			1	100	"This covergroup is for up_down_counter block."	1, "Up Down Counter Coverage"
1.01	values	Coverpoint	This coverpoint contains information of output values.	w_out	`GBINS_SIMPLE(0;1..9;10	1	50		
1.02	walk_one	Coverpoint	This coverpoint contains information of walk one pattern is covered or not on output signal.	w_out	`GBINS_WALK_ONE(8)	0	25		
1.03	walk_zero	Coverpoint	This coverpoint contains information of walk zero pattern is covered or not on output signal.	w_out	`GBINS_WALK_ZERO_LW	1	75		
1.04	up_down	Coverpoint	This coverpoint contains information of up and down functionality is covered or not.	w_up_down	`GBINS_SIMPLE(0; 1)	0			
1.05	reset	Coverpoint	This coverpoint contains information whether reset is applied or not.	w_reset	`GBINS_SIMPLE(0 ; 1)	0	0	"This coverpoint indeicates reset is applied or not"	

1.06	transition_lsb	Coverpoint	This coverpoint contains information of transition values in 4 LSB bits of output.	w_out[3:0]	bins lsb_1 = (0=>1=>2);
					bins lsb_2 = (3 => 4);
					bins lsb_3 = (5=>6=>7=>8);
					bins lsb_4 = (14=>15);
							80		

1.07	valuesXreset	Cross	This cross provides information of output value with reset.	values, reset		1	70		
1.08	transition_lsbXup_down	Cross	This cross provided information of transition bins covered when up counter is enabled.	transition_lsb, up_down		0	100		
2	i2c_cg	Covergroup	This covergroup contains information about i2c block.			0			

Sheet1 Sheet2

Figure 4 XLS Format

This tool supports very flexible format and any input details can be written in any column. This tool requires couple of mandatory inputs like Name, Identifier, Variable and Bins. Other details are optional. So that, compiled version of code can be generated properly. All coverpoints and crosses shall be written just below the related covergroup and before starting of a new covergroup.