EZ-BIST User Manual

v5.0

Contents

List of Figures

List of Tables

Type conversion in this document

Conversion	Meaning for use
Bold	Items in the user interface that you select or click and text that you type into the user interface
<Italic>	Variables in commands, code syntax, and path names
`Courier`	File name
“”	Emphasize the meaning
Color in blue	The outputs from EZ-BIST tool presenting in blue color
…	Omitted material in a line of code
⋮	Omitted lines in code and report examples
[ ]	Optional items in syntax descriptions to specify
( )	Explanations or to clarify meaning
{ }	Repeatable items in syntax descriptions
|	Separated the individual item in syntax descriptions

1. Introduction to EZ-BIST

EZ-BIST is an EDA tool that can generate the test circuit for MBIST (Memory Built-In Self-Test), providing total solutions including comprehensive test algorithms, auto-grouping mechanism, and auto-integration mechanism for MBIST circuits and the original circuit. It is easy for users to generate optimized MBIST circuits.

1.1. Features

As shown in Table 1-1, EZ-BIST supports several features. For more details, please refer to Application Notes.

Table 1-1 EZ-BIST Features

Feature		Description
POT	Power_On-Test	It is used to guarantee that memory can execute normally after powered on, EZ-BIST supports the POT function for users to implement the POT design.
ACT	Auto-Clock Tracing	ACT can trace the clock root to the clock source of memory modules and classify those memories into different clock domains. This mechanism not only saves time of connecting clock sources manually but also helps users to trace the clock in an easier way during creating MBIST.
BUF	Bottom-Up Flow	BUF is designed for IP/Harden implementation. Users can insert MBIST in an individual module. Then, integrate these individual modules in the top module.
AGC	Auto-Gating Clock Cell Insertion Flow	To reduce power consumption, EZ-BIST supports AGC for users to insert gate cells and MUX in front of MBIST automatically.
DIAG	Diagnosis Function	In general, MBIST only shows the results of pass or fail after MBIST executes memory testing. To analyze memory defects, EZ-BIST supports memory diagnosis to collect related information such as memory failure addresses, failure patterns, etc. In addition to collecting information, EZ-BIST diagnosis can also assign diagnosis buffer sizes and control the diagnosis timing.

1.2. Architecture

Figure 1-1 shows the operation flow of EZ-BIST.

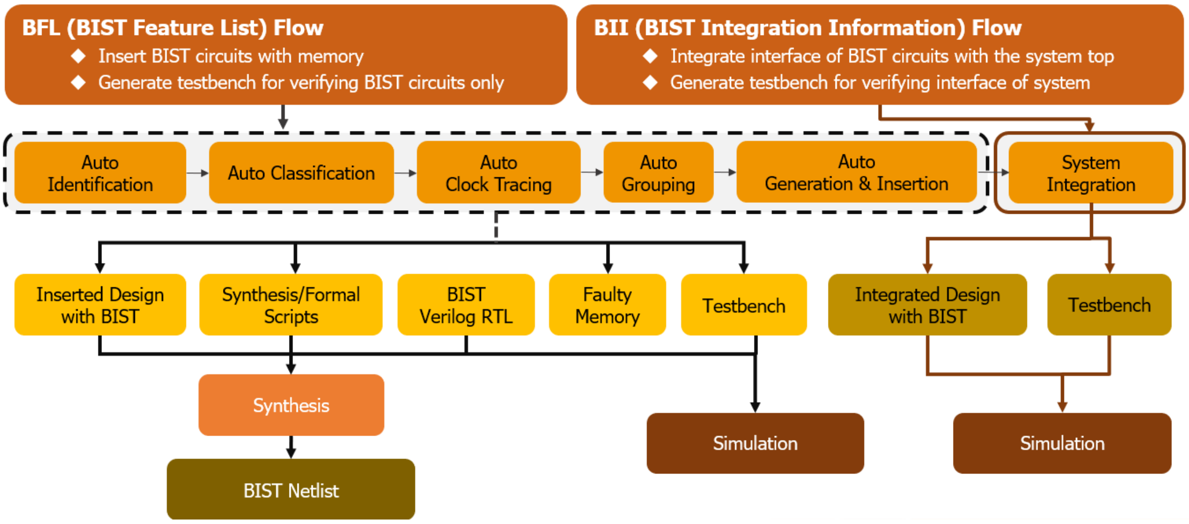


Figure 1-1 EZ-BIST Operation Flow Diagram

EZ-BIST input files include the files listed below:

Table 1-2 EZ-BIST Input Files

Top HDL Design	Top HDL design with memory models
Memory Module	Verilog files of memory models
UDM Files	User-defined memory files

EZ-BIST output files include the files listed below:

Table 1-3 EZ-BIST Output Files

Inserted Design	Integrated MBIST circuits with the top HDL design
Synthesis Scripts	Synthesis scripts for users to synthesize
MBIST Verilog Design	Generated MBIST circuits design
Fault Memory	Generated fault memory models This is used to verify functional correctness of MBIST and circuits with a pre-defined error bit memory.
Testbench	Testbench of MBIST circuits simulation

2. EZ-BIST Command Options and Parameters

Users can execute EZ-BIST commands with the options, *--help* or *-h*, to know all the options supported by EZ-BIST. Figure 2-1 shows an example of executing EZ-BIST with option *-h* and this chapter will introduce these options. The upper section is the command list. The lower section is the command descriptions.

```
usage: ezBist [-h] [-bii INTEGRATE_FILE] [-bfl BFL_FILE]
              [-f RUN_FILE [RUN_FILE ...]] [-v VERILOG_FILE [VERILOG_FILE ...]]
              [-W DIR] [-top MODULE] [-I] [--genmeminfo]
              [-integ FILE [FILE ...]] [-u FILE [FILE ...]] [-pm Verilog type]
              [--integrator] [--faultfree] [--ug UDM_FILE config_FILE]
              [--rcfg Addr_length Data_width output_FILE] [--tempgen]
              [--memchecker] [--memlib2udm MEMLIB_FILE]
              [--bflconfig [BFL_FILE]] [--biiconfig [BII_FILE]]
              [--pathconv work_path] [--STILloopformat work_path]
              [--latchgo_hier latchgo_data meminfo] [--udmgui [UDMGUI]]
              [--meminfogui [MEMINFO]]

optional arguments:

 -h, --help                              show this help message and exit
 -bii INTEGRATE_FILE                     input BII file
 -bfl BFL_FILE                           input BFL file
 -f RUN_FILE [RUN_FILE ...]              input run file(s)
 -v VERILOG_FILE [VERILOG_FILE ...]      input 4 verilog file(s)
 -W DIR                                  specify working path
 -top MODULE, -T MODULE                  specify top module
 -I, --insert                            insert BIST to design
(..........................................................................................................)
```

Figure 2-1 EZ-BIST Command Options

2.1. Invoke EZ-BIST with the GUI Mode

Usage: *--gui*

Description: This option is used to invoke EZ-BIST with the GUI mode.

Example: *$ ezBist --gui*

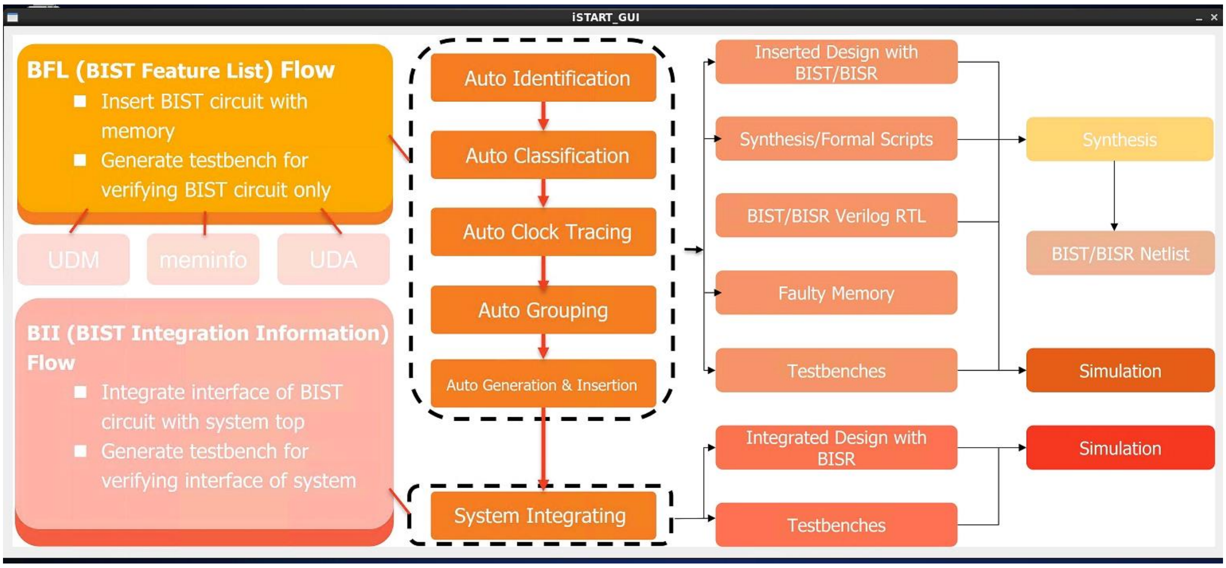
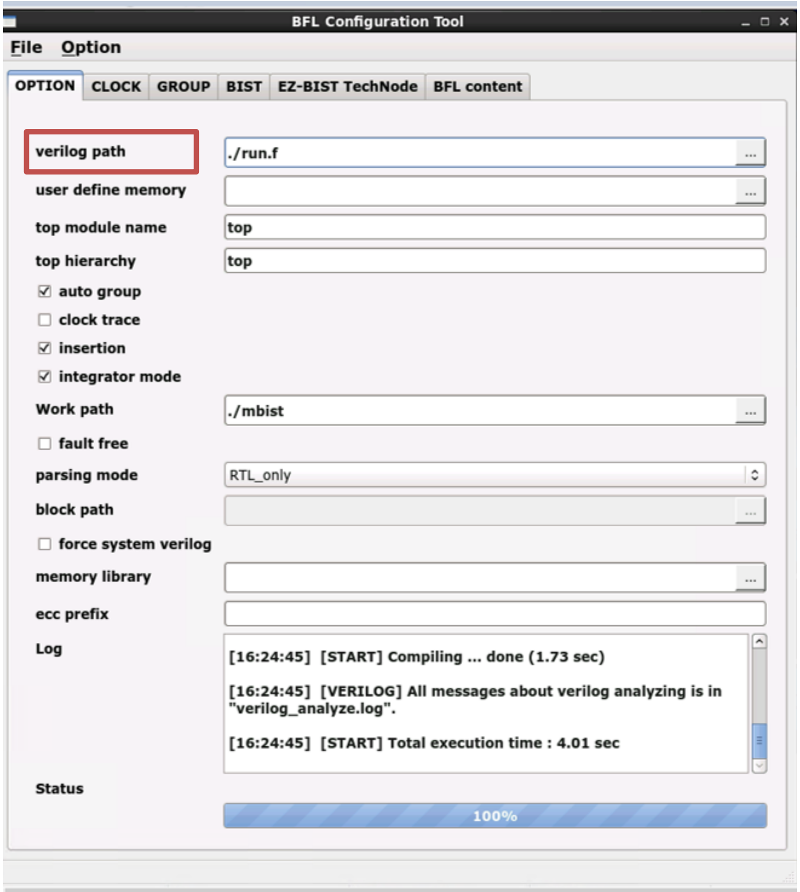


Figure 2-2 EZ-BIST GUI Mode

2.2. Input Verilog Files



verilog path: ./run.f
user define memory:
top module name: top
top hierarchy: top
☑ auto group
☐ clock trace
☑ insertion
☑ integrator mode
Work path: ./mbist
☐ fault free
parsing mode: RTL_only
block path:
☐ force system verilog
memory library:
ecc prefix:
Log:

```
[16:24:45]  [START] Compiling ... done (1.73 sec)

[16:24:45]  [VERILOG] All messages about verilog analyzing is in
"verilog_analyze.log".

[16:24:45]  [START] Total execution time : 4.01 sec
```

Status: 100%

Figure 2-3 Verilog File Path

Usage: *-v [VERILOG_PATH]*

Description: This option specifies the paths of Verilog design files. The design files here include “system design files” and “memory models”. EZ-BIST provides an auto-insertion function to integrate MBIST circuits into the original system design. For this reason, users need to provide the whole design files rather than the memory files only.

This option supports either reading one Verilog file or reading all files in the working directory. It also supports the file-list file format `*.f`. Users can integrate all design files into a single file-list file and read it through EZ-BIST commands. EZ-BIST will read design files automatically. The file-list file also supports *+define+, +incdir+,* and the *-y* options.

Example 1: *$ ezBist -v vlog_1*

EZ-BIST will read the Verilog files in vlog_1 directory.

Example 2: $ *ezBist -v vlog_1/file1.v vlog_2/file4.v*

EZ-BIST will read the `file1.v` in vlog_1 directory and `file4.v` in vlog_2 directory.

Example 3: *$ ezBist -v filelist.f*

EZ-BIST will read the designs in `fielist.f`. Figure 2-4 is an example of the file-list file.

```
-v ./memory/rf_2p_24x28.v
-v ./memory/sram_sp_4096x64.v
-v ./memory/rom_6144_64.v
-v ./memory/rf_sp_128x22.v
-v ./memory/sram_dp_1024x64.v
-v ./memory/rf_2p_24x56.v
-v ./memory/sram_sp_2048x64.v
-v ./memory/sram_sp_640x32.v
-v ./memory/rf_2p_64x64.v
-v ./memory/rf_2p_72x14.v
-v ./memory/sram_sp_1024x32.v
-v ./memory/RA1RW_D2048_W128_BE_RE.v
-v ./memory/RA1RW_D2048_W140_BE_RE.v
-v ./memory/RA1RW_D1024_W128_BE_RE.v

./top.v
```

Figure 2-4 File-list File Example

2.3. Specify the Working Path

Usage: *-W [WORK_PATH]*

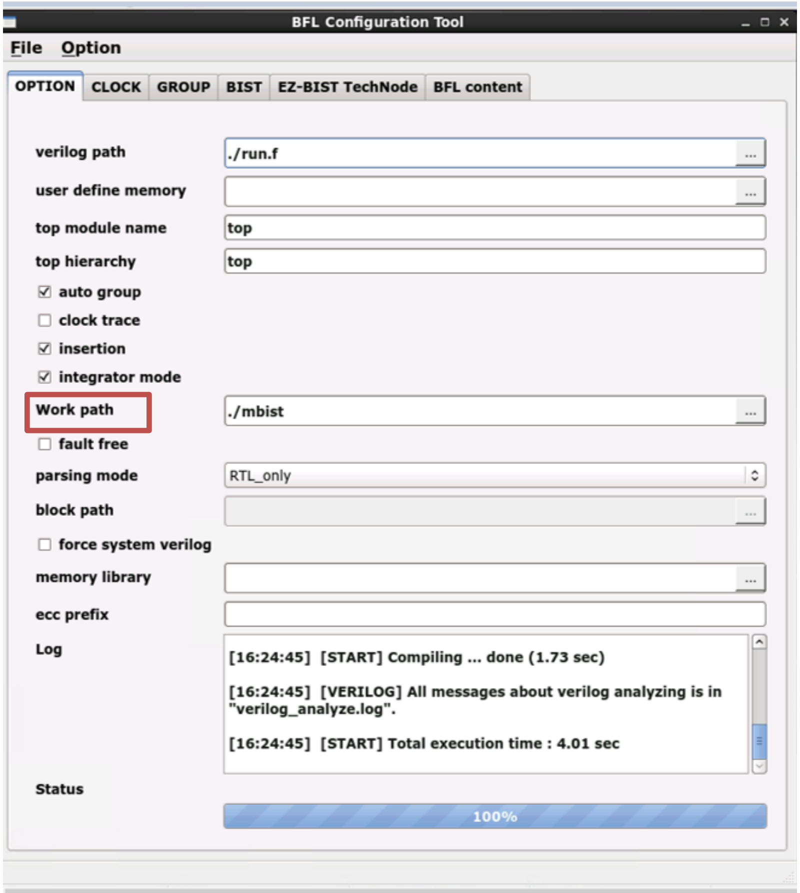
Description: This option is for setting the output directory of EZ-BIST execution results.

Example 1: *$ ezBist -v [VLOG_PATH]/[file_1].v -W [WORK_PATH]*

EZ-BIST will read the `file_1.v` design file and save output results into WORK_PATH.

Example 2: *$ ezBist -v [VLOG_PATH]/[file_1].v*

Without the *-W* option, EZ-BIST will save all generated results into the current working directory.



verilog path: ./run.f
user define memory:
top module name: top
top hierarchy: top
☑ auto group
☐ clock trace
☑ insertion
☑ integrator mode
Work path: ./mbist
☐ fault free
parsing mode: RTL_only
block path:
☐ force system verilog
memory library:
ecc prefix:
Log:

```
[16:24:45]  [START] Compiling ... done (1.73 sec)

[16:24:45]  [VERILOG] All messages about verilog analyzing is in
"verilog_analyze.log".

[16:24:45]  [START] Total execution time : 4.01 sec
```

Status: 100%

Figure 2-5 Work Path

2.4. Auto-Identify the Memory Model

Usage: *--memchecker*

Description: This option is used to execute EZ-BIST memory checker to identify memory models defined by users with the *-v* option.

Example: *$ ezBist --memchecker -f filelist.f*

Users can check if there is a memory model that cannot be identified by reviewing the output messages as Figure 2-6.

```
Input file(s):
   [1] /home /workspace/project/memchecker/memory/rom_6144_64.v
   [2] /home//workspace/project/memchecker/memory/rf_2p_24x56.v
   [3] /home//workspace/project/memchecker/memory/sram_sp_4096x64.v
   [4] /home//workspace/project/memchecker/memory/sram_sp_640x32.v
   [5] /home//workspace/project/memchecker/memory/sram_sp_2048x64.v
   [6] /home//workspace/project/memchecker/memory/rf_2p_72x14.v
   [7] /home//workspace/project/memchecker/memory/RA1RW_D2048_W140…
   [8] /home//workspace/project/memchecker/memory/RA1RW_D2048_W128…
   [9] /home//workspace/project/memchecker/memory/sram_sp_1024x32.v
   [10] /home//workspace/project/memchecker/memory/rf_sp_128x22.v
   [11] /home//workspace/project/memchecker/top.v
   [12] /home//workspace/project/memchecker/memory/sram_dp_1024x64.v
   [13] /home//workspace/project/memchecker/memory/RA1RW_D1024_W128…
   [14] /home//workspace/project/memchecker/memory/rf_2p_24x28.v
   [15] /home//workspace/project/memchecker/memory/rf_2p_64x64.v
Valid file(s):
   [1] /home//workspace/project/memchecker/memory/rom_6144_64.v
   [2] /home//workspace/project/memchecker/memory/rf_2p_24x56.v
   [3] /home//workspace/project/memchecker/memory/sram_sp_4096x64.v
   [4] /home//workspace/project/memchecker/memory/sram_sp_640x32.v
   [5] /home//workspace/project/memchecker/memory/sram_sp_2048x64.v
   [6] /home//workspace/project/memchecker/memory/rf_2p_72x14.v
   [7] /home//workspace/project/memchecker/memory/RA1RW_D2048_W140_BE_RE.v
   [8] /home//workspace/project/memchecker/memory/RA1RW_D2048_W128_BE_RE.v
   [9] /home//workspace/project/memchecker/memory/sram_sp_1024x32.v
   [10] /home//workspace/project/memchecker/memory/rf_sp_128x22.v
   [11] /home//workspace/project/memchecker/memory/sram_dp_1024x64.v
   [12] /home//workspace/project/memchecker/memory/rf_2p_24x28.v
   [13] /home//workspace/project/memchecker/memory/rf_2p_64x64.v
Unrecognized file(s):
   [1] /home//workspace/project/memchecker/top.v
```

Figure 2-6 Memchecker Information

2.5. The Generate the ROM Signature

Usage: *--memchecker*

Description: This option is used to execute the EZ-BIST memory checker to generate a golden ROM signature with the *-v [ROM memory RTL code file]* option.

Example: *$ ezBist --memchecker -v [ROM memory RTL code file]*

Users can verify the signature created by the MBIST and compare with the golden one.

$ ezBist --memchecker -v rom_6144_64.v

Note: The value of a signature will be saved in the `*_gold_signature.txt` file (see Figure 2-7) and in the meantime, a `top.v` file will be generated and replaced the previous one in the memory folder.

```
rom_6144_64_verilog gold_signature = 7be4eb
```

Figure 2-7 The Example of *_gold_signature.txt

2.6. Template File Generator

Usage: *--tempgen*

Description: This option is used to generate a template file of EZ-BIST. These template files include BII (MBIST Integration Information) files, BFL (MBIST Feature List) files, UDM files, and PGF files as Figure 2-8.

Example: *$ ezBist --tempgen*

```
[ezBist][TEMPLATE] ezBist template generator:
                1. BIST Feature List (BFL)
                2. BIST Integration Information (BII)
                3. User defined memory
                4. Pattern Gen File (PGF)
                5. Quit
[ezBist][TEMPLATE] Select an option (Enter ':q' to quit):
```

Figure 2-8 EZ-BIST Template Generator

2.7. Input BFL File

Usage: *-bfl BFL_FILE*

Description: This option is used to define a BFL file for EZ-BIST.

Example:

```
$ ezBist -bfl [filename].bfl -W [WORK_PATH]
```

After executing this command, EZ-BIST will base on the parameter setting in the `[filename].bfl` file to generate MBIST related files into WORK_PATH.

2.8. Insert MBIST to Design

Usage: *-I, --insert*

Description: This option is used to integrate the generated MBIST circuits into the original system designs. Users need to define a top module name with the *-top* option when using this option.

Example:

```
$ ezBist -I -top [TOP_MODULE] -v [VLOG_PATH]/[file_1].v
```

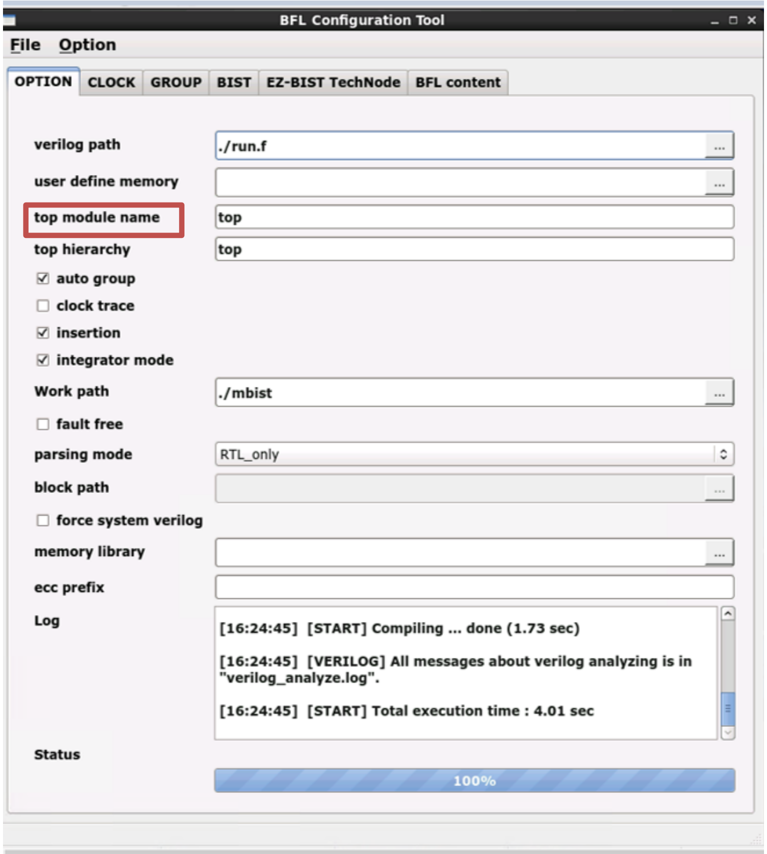
2.9. Specify Top Module

Usage: *-top [TOP_MODULE]*

Description: This option is used to integrate the generated MBIST circuits into the original system designs. Users need to define a top module name with the *-top* option when using this option.

Example:

```
$ ezBist -I -top [TOP_MODULE] -v [VLOG_PATH]/[file_1].v
```



verilog path: ./run.f

user define memory:

top module name: top

top hierarchy: top

☑ auto group

☐ clock trace

☑ insertion

☑ integrator mode

Work path: ./mbist

☐ fault free

parsing mode: RTL_only

block path:

☐ force system verilog

memory library:

ecc prefix:

Log

```
[16:24:45]  [START] Compiling ... done (1.73 sec)

[16:24:45]  [VERILOG] All messages about verilog analyzing is in
"verilog_analyze.log".

[16:24:45]  [START] Total execution time : 4.01 sec
```

Status: 100%

Figure 2-9 Top Module Name

2.10. Disable Clock Tracing

Usage: *-N, --disabletracedclk*

Description: This option is used to disable the clock tracing function of EZ-BIST. The default setting is "enabled".

Example: *$ ezBist -N -I --top [TOP_MODULE] -f file_list.f*

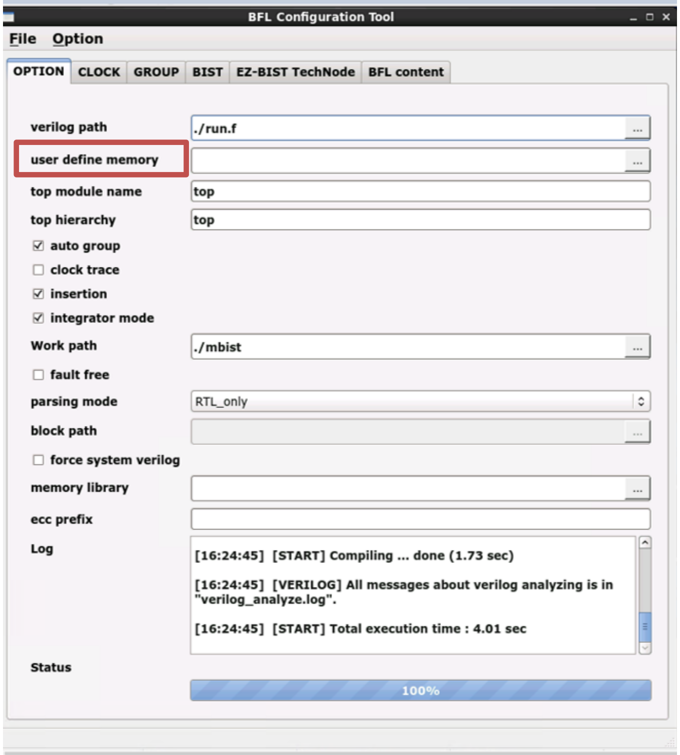
2.11. Input UDM File

Usage: *-u UDM_FILE*

Description: This option is used to read the UDM files generated by users. Users can generate UDM files when EZ-BIST cannot identify memory models automatically. To edit a UDM file, please refer to Application Notes for details.

Example: *$ ezBist -bfl [filename].bfl -u *.udm -W [WORK_PATH]*

EZ-BIST will read BFL files and UDM files in the working directory. The output results will be saved into WORK_PATH.



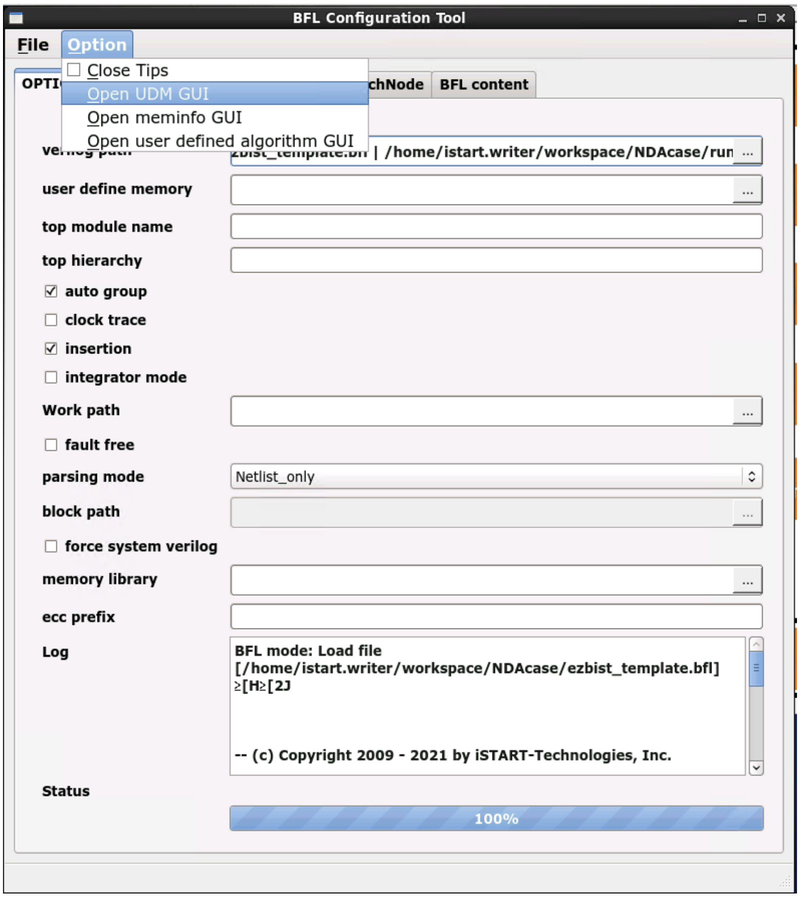
verilog path: ./run.f
user define memory:
top module name: top
top hierarchy: top
☑ auto group
☐ clock trace
☑ insertion
☑ integrator mode
Work path: ./mbist
☐ fault free
parsing mode: RTL_only
block path:
☐ force system verilog
memory library:
ecc prefix:
Log:

```
[16:24:45]  [START] Compiling ... done (1.73 sec)

[16:24:45]  [VERILOG] All messages about verilog analyzing is in
"verilog_analyze.log".

[16:24:45]  [START] Total execution time : 4.01 sec
```

Figure 2-10 User Defined Memory

2.12. Generate UDM File in GUI Mode

User can choose **Open UDM GUI** directly from BFL GUI.



verilog path: zbist_template.bfl | /home/istart.writer/workspace/NDAcase/run

user define memory

top module name

top hierarchy

☑ auto group

☐ clock trace

☑ insertion

☐ integrator mode

Work path

☐ fault free

parsing mode: Netlist_only

block path

☐ force system verilog

memory library

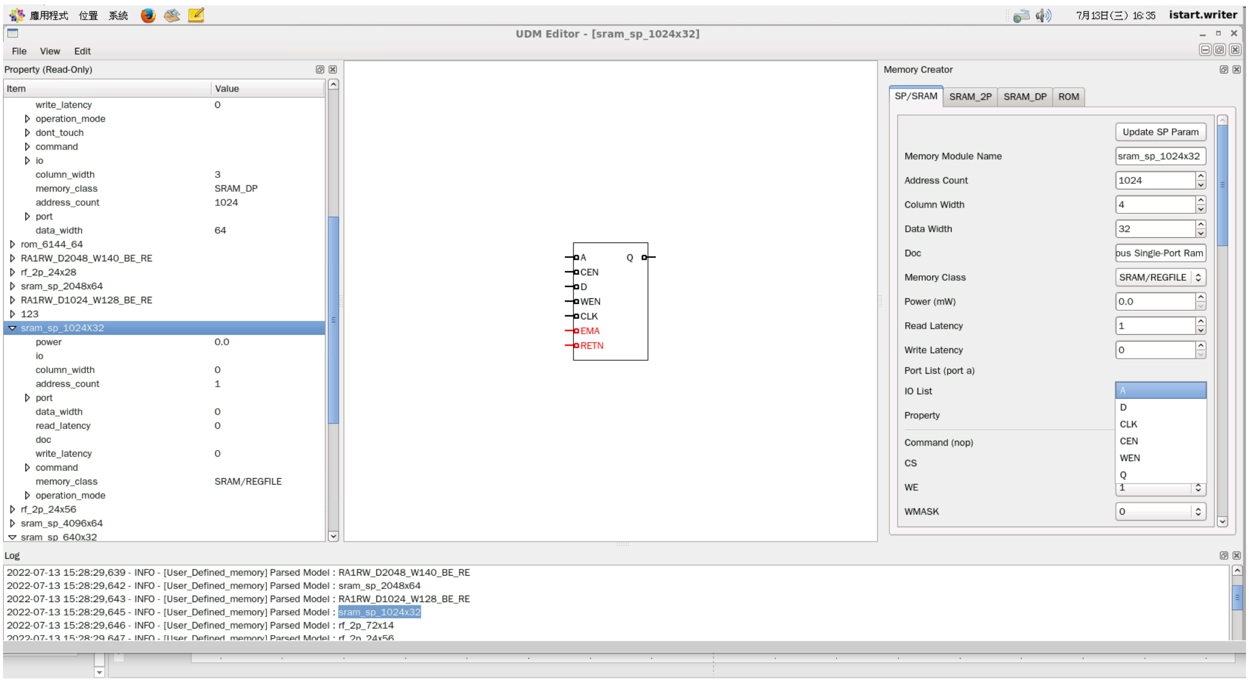
ecc prefix

Log

```
BFL mode: Load file
[/home/istart.writer/workspace/NDAcase/ezbist_template.bfl]
≥[H≥[2J


-- (c) Copyright 2009 - 2021 by iSTART-Technologies, Inc.
```

Status: 100%

Figure 2-11 Open UDM GUI



Item	Value
write_latency	0
operation_mode	
dont_touch	
command	
io	
column_width	3
memory_class	SRAM_DP
address_count	1024
port	
data_width	64
rom_6144_64	
RA1RW_D2048_W140_BE_RE	
rf_2p_24x28	
sram_sp_2048x64	
RA1RW_D1024_W128_BE_RE	
123	
sram_sp_1024X32	
power	0.0
io	
column_width	0
address_count	1
port	
data_width	0
read_latency	0
doc	
write_latency	0
command	
memory_class	SRAM/REGFILE
operation_mode	
rf_2p_24x56	
sram_sp_4096x64	
sram_sp_640x32	

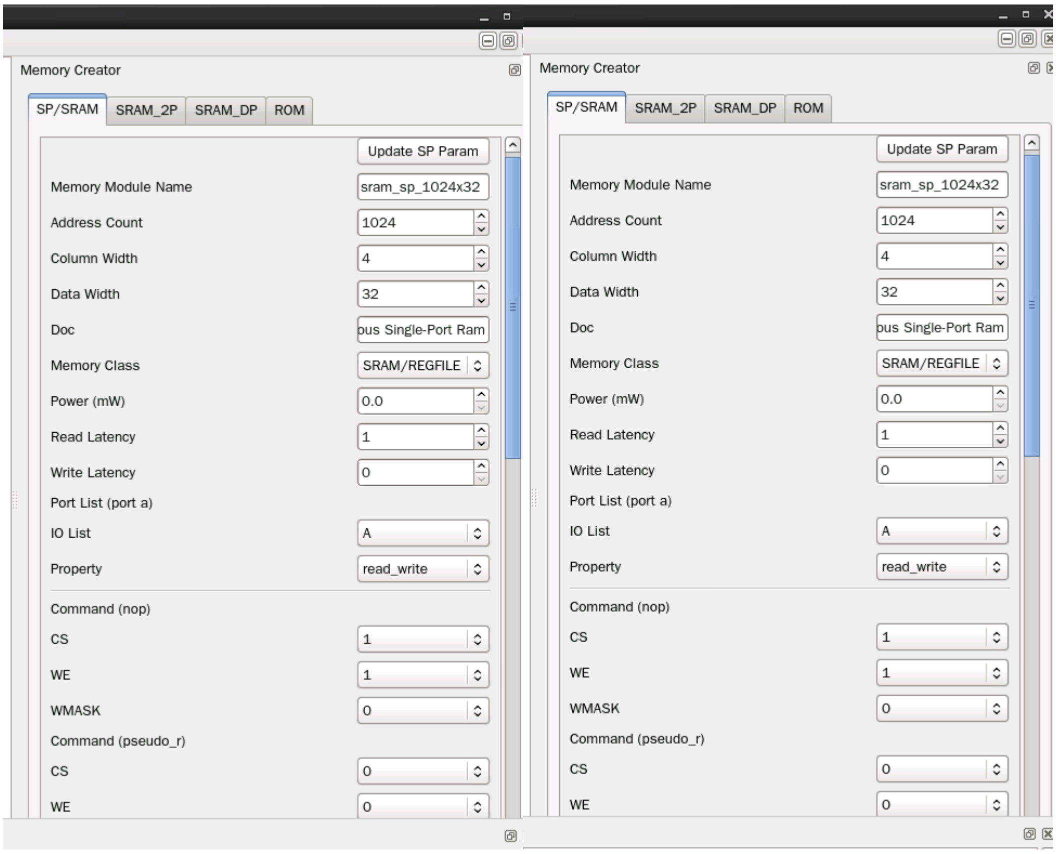
Memory Module Name: sram_sp_1024x32
Address Count: 1024
Column Width: 4
Data Width: 32
Doc: pus Single-Port Ram
Memory Class: SRAM/REGFILE
Power (mW): 0.0
Read Latency: 1
Write Latency: 0
Port List (port a)
IO List: A, D, CLK, CEN, WEN, Q
Property
Command (nop)
CS
WE: 1
WMASK: 0

```
2022-07-13 15:28:29,639 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D2048_W140_BE_RE
2022-07-13 15:28:29,642 - INFO - [User_Defined_memory] Parsed Model : sram_sp_2048x64
2022-07-13 15:28:29,643 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D1024_W128_BE_RE
2022-07-13 15:28:29,645 - INFO - [User_Defined_memory] Parsed Model : sram_sp_1024x32
2022-07-13 15:28:29,646 - INFO - [User_Defined_memory] Parsed Model : rf_2p_72x14
2022-07-13 15:28:29,647 - INFO - [User_Defined_memory] Parsed Model : rf_2p_24x56
```

Figure 2-12 Support Batches Adding and Multiple Formats

Set the parameters below through GUI:

- Memory basic parameter
- Port read/write behavior
- Test Port
- IO Port, Don’t Touch Port, Repair Port

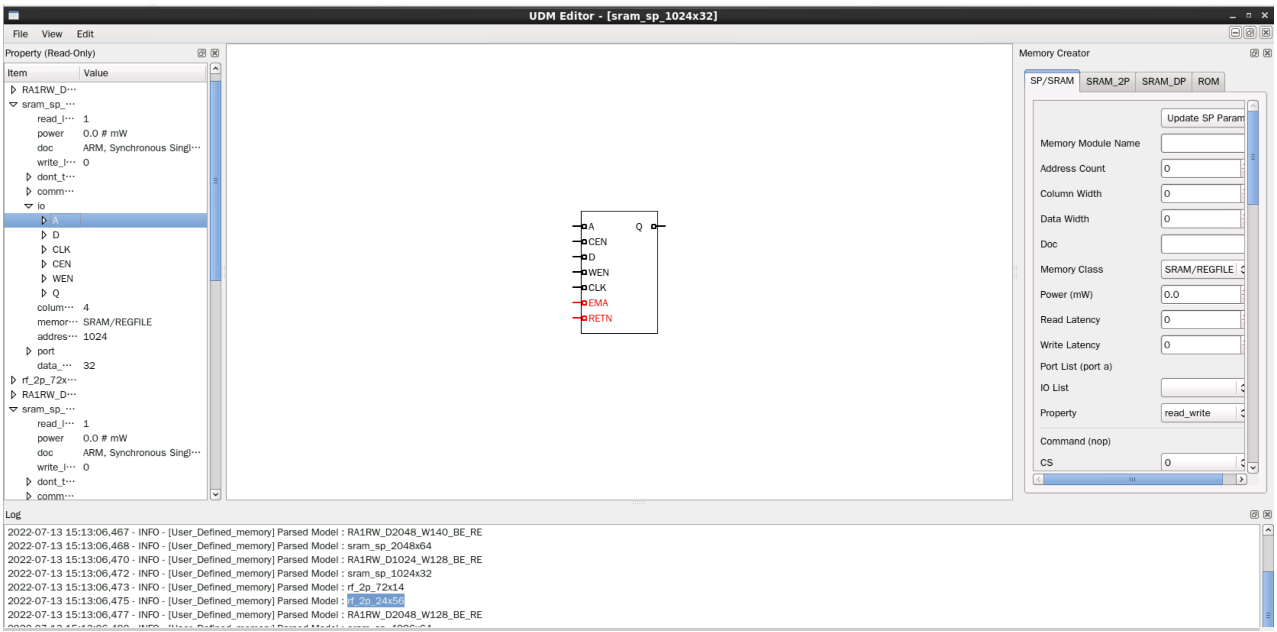


Memory Module Name: sram_sp_1024x32
Address Count: 1024
Column Width: 4
Data Width: 32
Doc: ous Single-Port Ram
Memory Class: SRAM/REGFILE
Power (mW): 0.0
Read Latency: 1
Write Latency: 0
Port List (port a)
IO List: A
Property: read_write
Command (nop)
CS: 1
WE: 1
WMASK: 0
Command (pseudo_r)
CS: 0
WE: 0

Memory Module Name: sram_sp_1024x32
Address Count: 1024
Column Width: 4
Data Width: 32
Doc: ous Single-Port Ram
Memory Class: SRAM/REGFILE
Power (mW): 0.0
Read Latency: 1
Write Latency: 0
Port List (port a)
IO List: A
Property: read_write
Command (nop)
CS: 1
WE: 1
WMASK: 0
Command (pseudo_r)
CS: 0
WE: 0

Figure 2-13 Memory Parameter Settings

(For the detailed information, please refer to Chapter 10 in Application Notes)

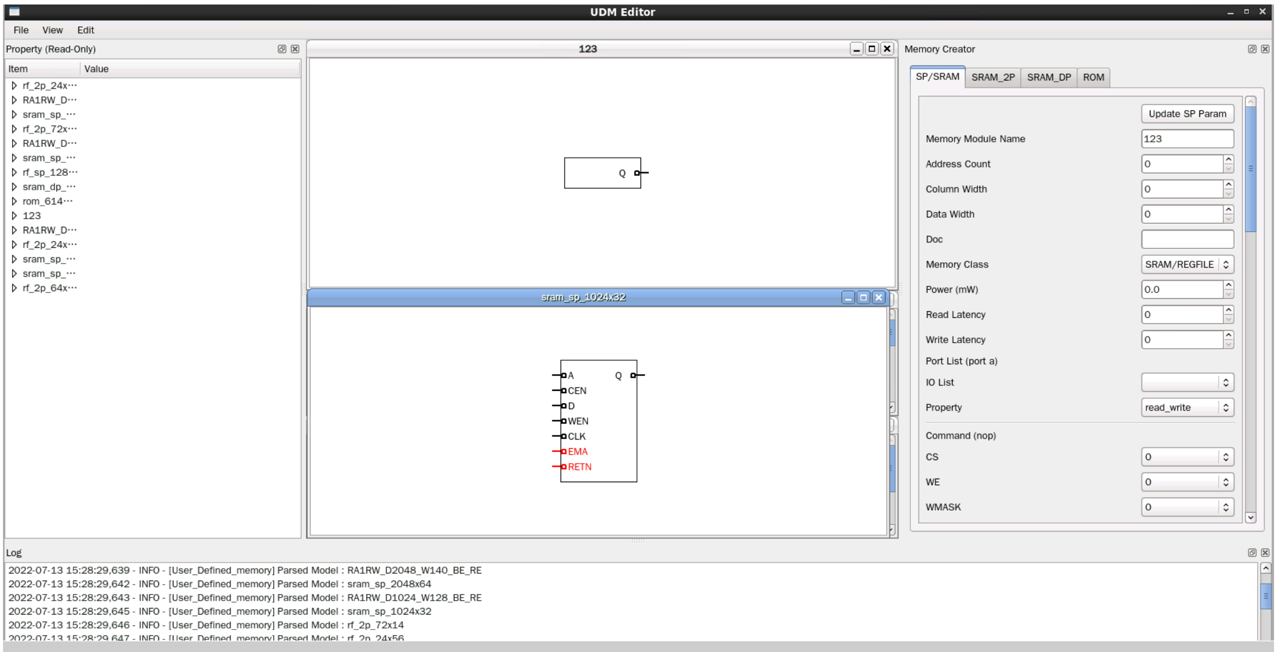


```
Item          Value
RA1RW_D···
sram_sp_···
  read_l···   1
  power       0.0 # mW
  doc         ARM, Synchronous Singl···
  write_l···  0
  dont_t···
  comm···
  io
    A
    D
    CLK
    CEN
    WEN
    Q
  colum···    4
  memor···    SRAM/REGFILE
  addres···   1024
  port
  data_···    32
rf_2p_72x···
RA1RW_D···
sram_sp_···
  read_l···   1
  power       0.0 # mW
  doc         ARM, Synchronous Singl···
  write_l···  0
  dont_t···
  comm···
```

```
Memory Module Name
Address Count        0
Column Width         0
Data Width           0
Doc
Memory Class         SRAM/REGFILE
Power (mW)           0.0
Read Latency         0
Write Latency        0
Port List (port a)
IO List
Property             read_write
Command (nop)
CS                   0
```

```
2022-07-13 15:13:06,467 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D2048_W140_BE_RE
2022-07-13 15:13:06,468 - INFO - [User_Defined_memory] Parsed Model : sram_sp_2048x64
2022-07-13 15:13:06,470 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D1024_W128_BE_RE
2022-07-13 15:13:06,472 - INFO - [User_Defined_memory] Parsed Model : sram_sp_1024x32
2022-07-13 15:13:06,473 - INFO - [User_Defined_memory] Parsed Model : rf_2p_72x14
2022-07-13 15:13:06,475 - INFO - [User_Defined_memory] Parsed Model : rf_2p_24x56
2022-07-13 15:13:06,477 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D2048_W128_BE_RE
```

Figure 2-14 IO Editing through EZ-BIST

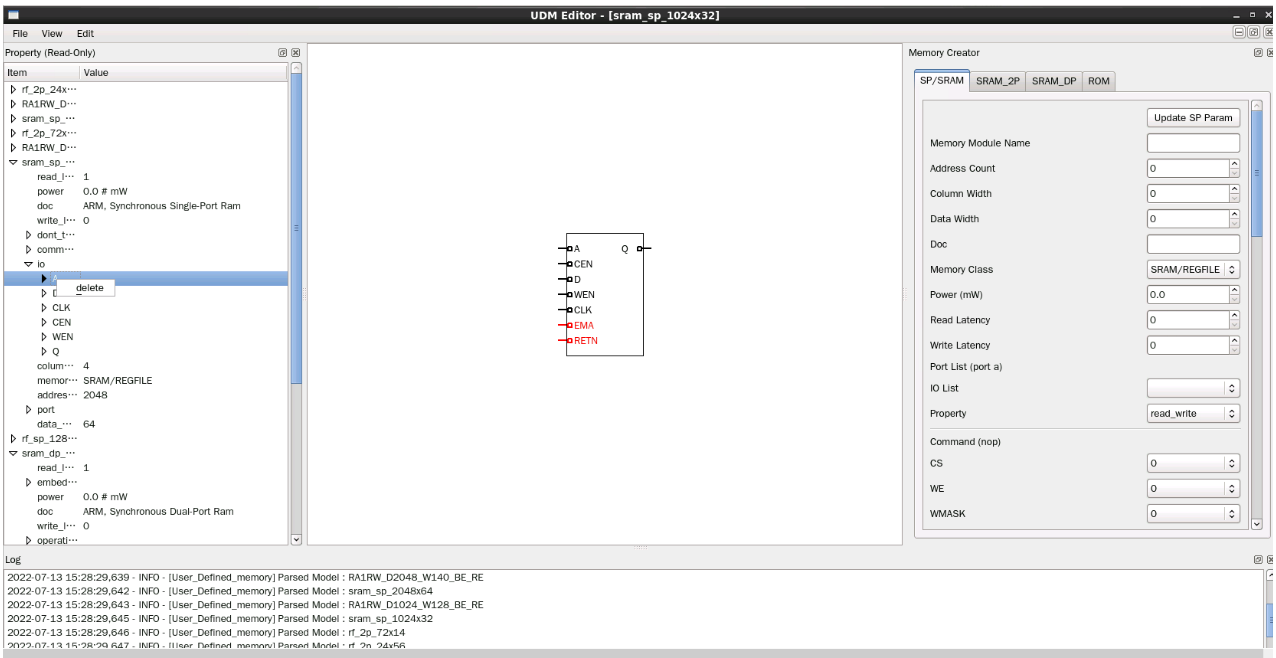


```
rf_2p_24x···
RA1RW_D···
sram_sp_···
rf_2p_72x···
RA1RW_D···
sram_sp_···
rf_sp_128···
sram_dp_···
rom_614···
123
RA1RW_D···
rf_2p_24x···
sram_sp_···
sram_sp_···
rf_2p_64x···
```

```
Memory Module Name   123
Address Count        0
Column Width         0
Data Width           0
Doc
Memory Class         SRAM/REGFILE
Power (mW)           0.0
Read Latency         0
Write Latency        0
Port List (port a)
IO List
Property             read_write
Command (nop)
CS                   0
WE                   0
WMASK                0
```

```
2022-07-13 15:28:29,639 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D2048_W140_BE_RE
2022-07-13 15:28:29,642 - INFO - [User_Defined_memory] Parsed Model : sram_sp_2048x64
2022-07-13 15:28:29,643 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D1024_W128_BE_RE
2022-07-13 15:28:29,645 - INFO - [User_Defined_memory] Parsed Model : sram_sp_1024x32
2022-07-13 15:28:29,646 - INFO - [User_Defined_memory] Parsed Model : rf_2p_72x14
2022-07-13 15:28:29,647 - INFO - [User_Defined_memory] Parsed Model : rf_2p_24x56
```

Figure 2-15 IO Adding Rapidly Using Drag & Drop

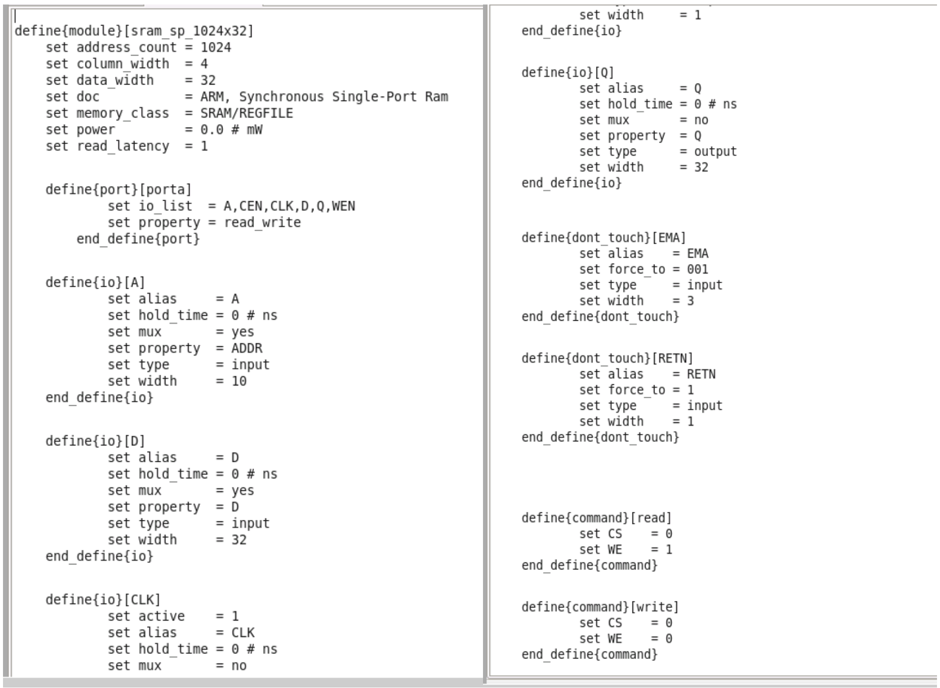


```
▷ rf_2p_24x···
▷ RA1RW_D···
▷ sram_sp_···
▷ rf_2p_72x···
▷ RA1RW_D···
▽ sram_sp_···
    read_l···  1
    power      0.0 # mW
    doc        ARM, Synchronous Single-Port Ram
    write_l··· 0
  ▷ dont_t···
  ▷ comm···
  ▽ io
      ▶ A
      ▷ D
      ▷ CLK
      ▷ CEN
      ▷ WEN
      ▷ Q
    colum···   4
    memor···   SRAM/REGFILE
    addres···  2048
  ▷ port
    data_···   64
▷ rf_sp_128···
▽ sram_dp_···
    read_l···  1
  ▷ embed···
    power      0.0 # mW
    doc        ARM, Synchronous Dual-Port Ram
    write_l··· 0
  ▷ operati···
```

Memory Module Name
Address Count: 0
Column Width: 0
Data Width: 0
Doc
Memory Class: SRAM/REGFILE
Power (mW): 0.0
Read Latency: 0
Write Latency: 0
Port List (port a)
IO List
Property: read_write
Command (nop)
CS: 0
WE: 0
WMASK: 0

```
2022-07-13 15:28:29,639 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D2048_W140_BE_RE
2022-07-13 15:28:29,642 - INFO - [User_Defined_memory] Parsed Model : sram_sp_2048x64
2022-07-13 15:28:29,643 - INFO - [User_Defined_memory] Parsed Model : RA1RW_D1024_W128_BE_RE
2022-07-13 15:28:29,645 - INFO - [User_Defined_memory] Parsed Model : sram_sp_1024x32
2022-07-13 15:28:29,646 - INFO - [User_Defined_memory] Parsed Model : rf_2p_72x14
2022-07-13 15:28:29,647 - INFO - [User_Defined_memory] Parsed Model : rf_2p_24x56
```

Figure 2-16 Delete IO with Right Click



```
define{module}[sram_sp_1024x32]
    set address_count = 1024
    set column_width  = 4
    set data_width    = 32
    set doc           = ARM, Synchronous Single-Port Ram
    set memory_class  = SRAM/REGFILE
    set power         = 0.0 # mW
    set read_latency  = 1


    define{port}[porta]
            set io_list  = A,CEN,CLK,D,Q,WEN
            set property = read_write
        end_define{port}


    define{io}[A]
            set alias     = A
            set hold_time = 0 # ns
            set mux       = yes
            set property  = ADDR
            set type      = input
            set width     = 10
    end_define{io}


    define{io}[D]
            set alias     = D
            set hold_time = 0 # ns
            set mux       = yes
            set property  = D
            set type      = input
            set width     = 32
    end_define{io}


    define{io}[CLK]
            set active    = 1
            set alias     = CLK
            set hold_time = 0 # ns
            set mux       = no
```

```
            set width     = 1
    end_define{io}

    define{io}[Q]
            set alias     = Q
            set hold_time = 0 # ns
            set mux       = no
            set property  = Q
            set type      = output
            set width     = 32
    end_define{io}


    define{dont_touch}[EMA]
            set alias    = EMA
            set force_to = 001
            set type     = input
            set width    = 3
    end_define{dont_touch}

    define{dont_touch}[RETN]
            set alias    = RETN
            set force_to = 1
            set type     = input
            set width    = 1
    end_define{dont_touch}



    define{command}[read]
            set CS    = 0
            set WE    = 1
    end_define{command}

    define{command}[write]
            set CS    = 0
            set WE    = 0
    end_define{command}
```

Figure 2-17 User Define Memory Generation

2.13. Integrate Multiple MBIST Circuits

Usage:	*--integrator*
Description:	This option is used to integrate multiple MBIST circuits.
Example:	*$ ezBist --integrator -bii [filename].bii -W [WORK_PATH]* EZ-BIST will refer to BII files to integrate multiple MBIST circuits and save output results into WORK_PATH.

2.14. Generate UDM File with Library File

Usage:	*--memlib2udm -lv [filename].memlib* or *–memlib2udm -f [filename].memlib*
Description:	This option is used to generate UDM files from memory library files. If there is only one file, use *--memlib2udm -lv [filename].memlib*. If there is a file list that contains multiple files, use *--memlib2udm -f [filename].memlib.*
Example:	*$ ezBist --memlib2udm -lv sram_512x8.memlib* EZ-BIST will generate UDM files for memory sram_512x8.

2.15. Generate UDM File with Configuration File

Usage: *--ug UDM_File config_file*

Description: This option is used to generate UDM files based on the settings in the configuration file. The configuration file is used to set different widths for address port and data port. Figure 2-18 shows an example of the configuration file. The first column defines the memory model name, the second column defines the address count, the third column defines data width, and the fourth column defines mux.

Example: *$ ezBist --ug sram_512x8.udm config.file*

EZ-BIST will generate UDM files with the same type as the sram_512x8 memory model but with different data width or address width.

```
#module_name                   address_count     data_width      mux
U40LP_VHD_SRF_16X8M4B1               16               8           4
U40LP_VHD_SRF_116X38M4B1            116              38           4
U40LP_VHD_SRF_216X28M4B1            216              28           4
U40LP_VHD_SRF_316X18M4B1            316              18           4
```

Figure 2-18 UDM Configuration File Example

2.16. Parsing Type Definition

Usage: *-pm, --parsingmode*

Description: This option is used to specify the input design type. The supported types are **RTL_only** and **Netlist_only**.

Example:

```
$   ezBist -pm Netlist_only -v example.v
```

EZ-BIST will import `example.v` with the nestlist format.

2.17. Fault Free

Usage: *--faultfree*

Description: This option is used to decide whether the generated system designs include fault memory modes or not. When this option is set, the system designs with and without fault memories will be generated. When this option is not set, only the system designs with fault memories will be generated. The file name will be `[design]_INS.v`.

Example 1:

```
$   ezBist -bfl ezBist _template.bfl -I -W ./work
```

EZ-BIST will generate an integrated system design with fault memory models.

Example 2:

```
$   ezBist -bfl ezBist _template.bfl -I --faultfree -W ./work
```

EZ-BIST will generate integrated system designs with and without fault memory models, respectively.

2.18. RCF Generator

Usage: *--rcfg address_length data width output_file*

Description: This option is used to generate an example RCF file for ROM memory model. The content of output RCF file is random.

Example:

```
$   ezBist --rcfg 32 8 example.rcf
```

EZ-BIST will generate an example RCF file with 32x8 matrix format.

2.19. STIL Format

Usage: *--STILloopformat*

Description: Change STIL file into the loop format.

Example:

```
$   ezBist --STILloopformat
```

EZ-BIST will generate STIL file into loop format.

If there are many repetitive testing commands, using the option will simplify the testing commands as loop instructions.

3. EZ-BIST BFL Options

Users can execute EZ-BIST to generate the MBIST circuits with the BFL flow. This chapter will introduce the setting options in the BFL file.

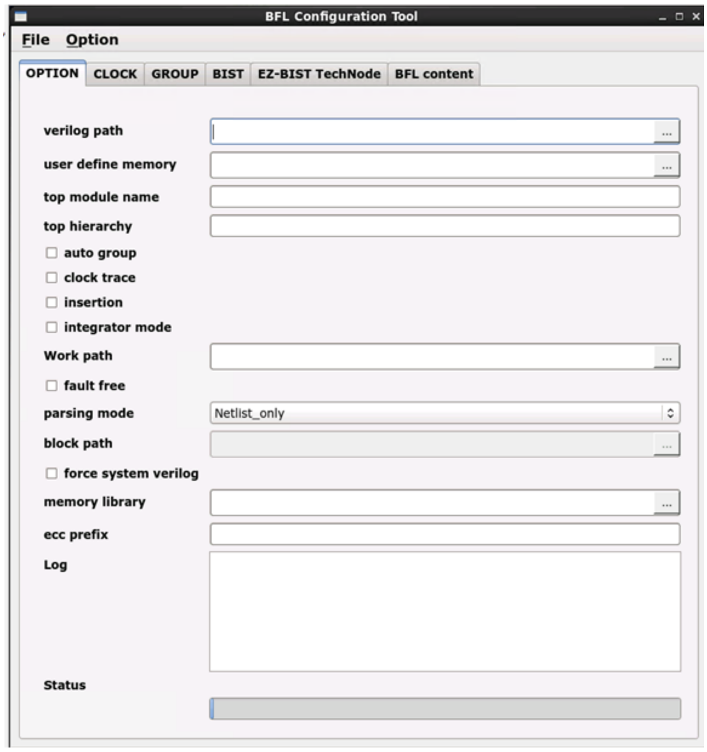
The definitions of function blocks in BFL file are defined as follows:

define{function}
…
end_define{function}

Users can find different options in each function block as below.

3.1. OPTION Function Block

Figure 3-1 shows the parameters in the OPTION function block.



parsing mode: Netlist_only

Figure 3-1 OPTION Function Block

Argument	Option
Description	
verilog_path	User defined
Set the Verilog file paths for EZ-BIST. The format can be set either by *file1.v* | *file2.v* | *fileN.v* or file-list file (`*.f`). Note: Each file is separated by a vertical bar "|". Example: *set verilog_path = ./top.f*	
user_define_memory	User defined
Set UDM file paths for EZ-BIST. The format can be *memory1.udm* | *memory2.udm* | … | *memoryN.udm.* Note: Each file is separated by a vertical bar "|". For more details, please refer to Application Notes. Example: *set user_define_memory = BRAINS.udm*	
top_module_name	User defined
Set the top module name of the system design which includes memory modules. Example: *set top_module_name = top*	
top_hierarchy	User defined
Specify the location (instance name) of the controller for MBIST circuits in the design architecture. Example: *set top_hierarchy = top*	
clock_trace	No, Yes
This option is for users to disable/enable the clock source tracing function. The default setting is "no". No: Disable the clock source tracing function Yes: Enable the clock source tracing function	

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>**auto_group**</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is for users to automatically group memory models based on the settings in the GROUP function block. The default setting is “no”.

No: Disable the clock auto-grouping function
Yes: Enable the clock auto-grouping function</td></tr>
<tr><td>**insertion**</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to integrate the generated MBIST circuits and the original system designs. Figure 3-2 shows the block diagram of the inserted system design.

No: Disable the insertion function
Yes: Enable the insertion function

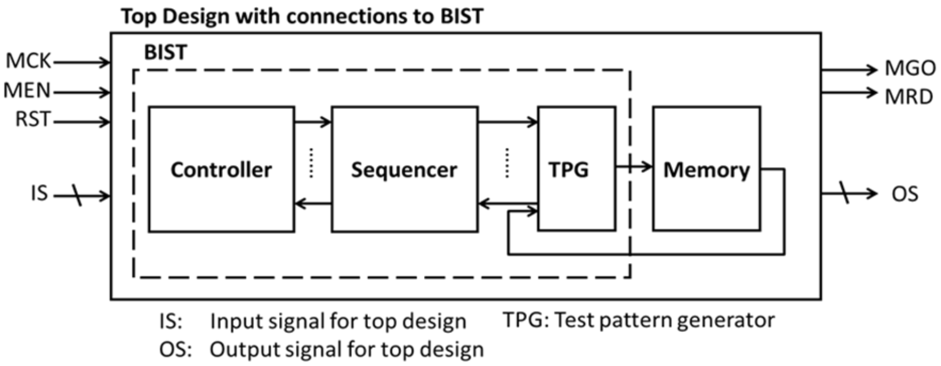


Figure 3-2 Block Diagram of System Design with MBIST Inserted</td></tr>
<tr><td>**integrator_mode**</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is for users to add the dedicated testing port in the top module of MBIST. Because these testing ports adhere to standard protocols such as IEEE 1149.1, users can use the shared pin design to reduce the pin count. The default setting is “no”.

No: EZ-BIST will generate some specific hookup pins for the BII flow. Users can use them to control MBIST or get data from MBIST.
Yes: EZ-BIST will reserve signals internally in advance for testing only in the BFL flow.

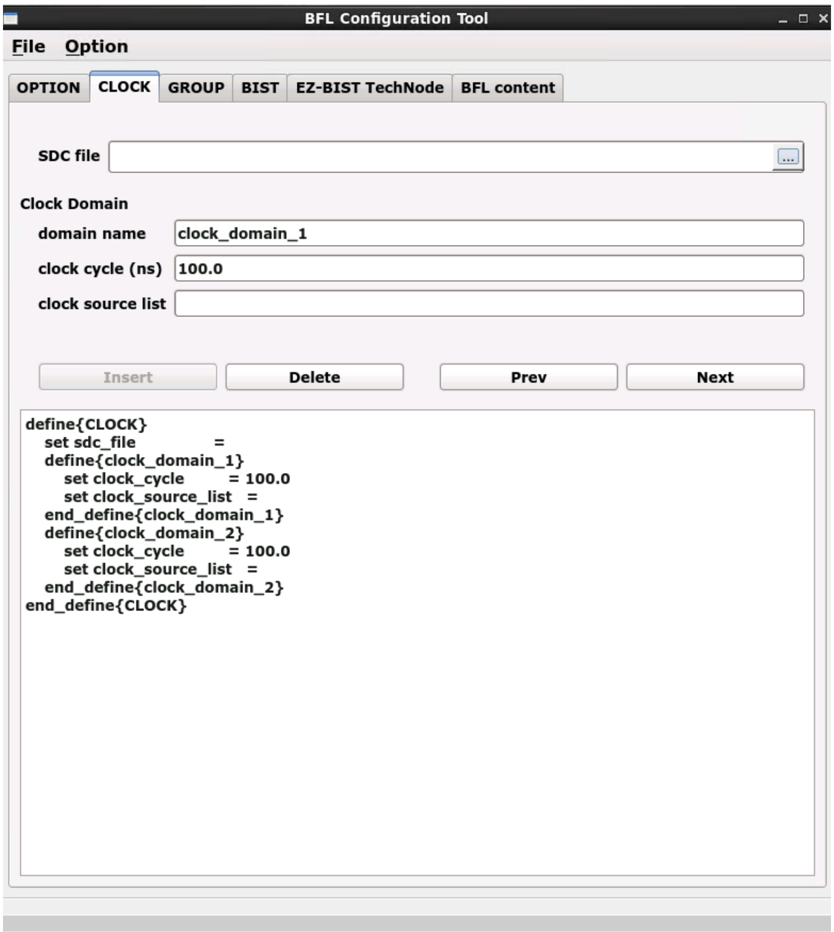
Note: The option must be set to “yes” when clock tracing turns on.</td></tr>
<tr><td>**work_path**</td><td>User defined</td></tr>
<tr><td colspan="2">Specify the path for saving the generated results in the BFL flow.</td></tr>
</table>

Argument	Option
Description	
fault_free	No, Yes
When this option is set to “no”, EZ-BIST will generate an integrated system design with fault memory models. On the contrary, when this option is set to “yes”, EZ-BIST will generate two integrated system designs with and without fault memory. However, the simulation will run on without fault memory. MBIST circuits are integrated into the original system design.	
parsing_mode	RTL_only, Netlist_only
This option defines the file format of the imported design, supporting RTL_only and Netlist_only. Note: If the Netlist file are not uniquified, the parsing mode must be set to “RLT_only.”	
ecc_prefix	User defined
Specify the prefix of ECC (Error Correction Code) related files. For example, when this option is set to “ECC”, the output ECC-related files will be named like `ECC_[design]_INS.v` and `ECC_[filename]_tb.v` etc.	
memory_library	User defined
Define memory library (shown in Example 1) to make START to load the information of memory models. If multiple files need to be set, they can be separated by a vertical bar ('|'). Alternatively, users can also fill in the memory file list as shown in Example 2. Example 1: *set memory_library = /home/workspace/ram1024x32.lvlib* Example 2: *set memory_library = ./mem_lib.f*	
block_path	User defined
While the design is implemented with the bottom-up flow to insert MBIST into the sub module, it will generate a `*.blockinfo` file in the sub module. Example: *set block_path = ./block1/START_block1.blockinfo | ./block2/START_block2.blockinfo*	
force_system_verilog	No, Yes
The default setting is “no”, in which case the initial parsing format is Verilog. The parsing format will support parsing both Verilog and System Verilog when users set the option to “yes”. No: The initial parsing format is Verilog. Yes: Parsing both Verilog and System Verilog are supported.	
disable_auto_identify	No, Yes

Argument	Option
Description	
The default setting is “no”. When the user confirms that the "set memory_library" for the memories in the design has been input in the tool, enabling "disable_auto_identify" will deactivate the tool's “auto identify memory” feature to reduce the overall runtime of the tool.	
skip_check_translate_off	No, Yes
Under the default condition (skip_check_translate_off = no), START will skip analyzing the content between "//synopsys translate_off" and "// synopsys translate_on": *//synopsys translate_off* *...(content)* *//synopsys translate_on* If users want the tool to recognize and analyze the content, please set skip_check_translate_off to yes.	

3.1.1. CLOCK Sub Function Block

Users can define the information of clock domain or provide an SDC file for EZ-BIST to do clock tracing.



SDC file

Clock Domain

domain name: clock_domain_1

clock cycle (ns): 100.0

clock source list:

```
define{CLOCK}
   set sdc_file              =
   define{clock_domain_1}
      set clock_cycle        = 100.0
      set clock_source_list   =
   end_define{clock_domain_1}
   define{clock_domain_2}
      set clock_cycle        = 100.0
      set clock_source_list   =
   end_define{clock_domain_2}
end_define{CLOCK}
```

Figure 3-3 Clock Sub Function Block

Table 3-1 Clock Information

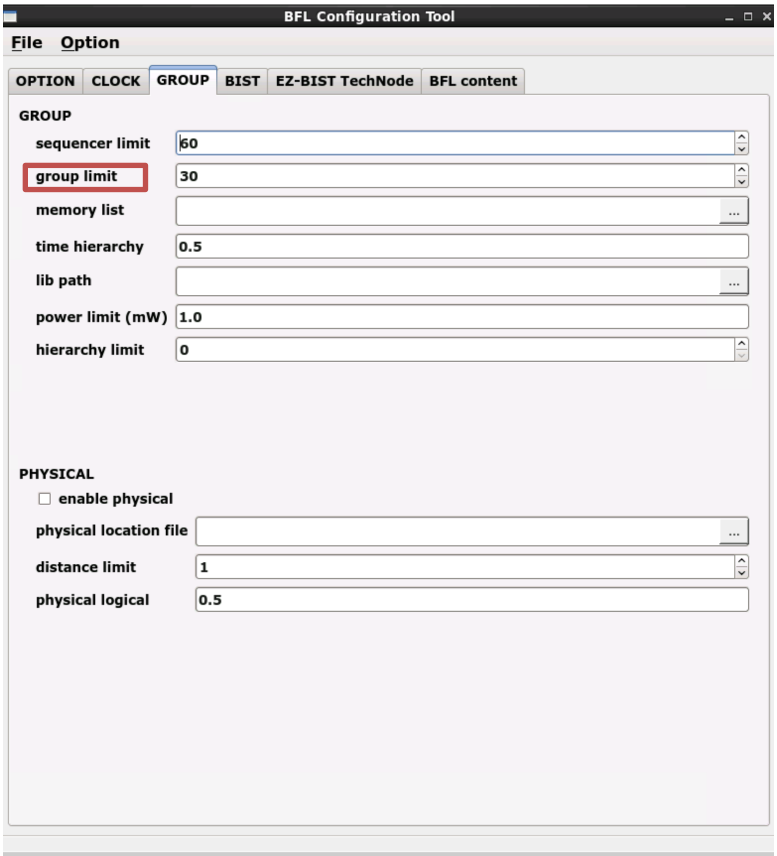
Argument	Option
Description	
sdc_file	User defined
Specify the path of an SDC file.	
define{clock_name}	User defined
Set the clock domain name.	
clock_cycle	User defined
Set the operating period of clock domain defined in “clock_name”.	
clock_source_list	User defined
Set the source pin or port of clock domain defined in “clock_name”.	

3.1.2. GROUP Sub Function Block

EZ-BIST assigns memory grouping according to the rule of clock domains, types of memory models, the criteria of grouping specifications, and power consumption. Users can also do memory grouping manually based on their own project requirements by editing the memory information file `*.meminfo`. Memory models in the same group can be tested in parallel to reduce the testing time.

Each memory will have the dedicated SEQ_ID (Sequencer ID) and GRP_ID (Group ID). Memories have the same SEQ_ID and GRP_ID are in the same group and can be tested at the same time.

The SEQ_ID is classified by types, specifications, and the clock domains of memory models. This ID means which sequencer the memory models belong to. The GRP_ID is classified by power consumption and number limitations of a single group.



sequencer limit: 60

group limit: 30

memory list:

time hierarchy: 0.5

lib path:

power limit (mW): 1.0

hierarchy limit: 0

☐ enable physical

physical location file:

distance limit: 1

physical logical: 0.5

Figure 3-4 Group Function Block

Argument	Option
Description	
sequencer_limit	User defined
This option defines the maximum amount of memory instances in a sequencer. Default Value: 60	
group_limit	User defined
This option is used to define the maximum amount of memory instances in a group. This number should be less than the value of **sequencer_limit**. Default Value: 30	
memory_list	User defined
Specify the paths of memory info file (`*.meminfo`). Figure 3-8 is an example of memory info file. For more details, please refer to Application Notes.	
lib_path	User defined
This option is for users to set the path of memory libraries. EZ-BIST will load power information of memory models from `*.lib` files and do memory grouping automatically based on the power criteria through the **power_limit** option.	
time_hierarchy	0 (time) <= value <= 1 (hierarchy)
This option is for users to adjust the weight between the testing time and design hierarchy. The default value is 0.5. For example: *set time_hierarchy = 0* EZ-BIST will assign memory grouping based on the optimized testing time. The testing time will be the highest priority. *set time_hierarchy = 1* EZ-BIST will assign memory grouping by hierarchy relationships. In this case, the logical hierarchy will be the highest priority.	
power_limit	User defined
Set the maximum limitation of power consumption in one group. For example: *set power_limit = 0.005* Note: The unit is mW and can be decimal.	

Argument	Option
Description	
hierarchy_limit	User defined
Set the maximum hierarchy number when doing auto-grouping. If the hierarchy number between memory models is larger than this number, EZ-BIST will not group these memory models into the same group. Default Value: 0 (no limitation of hierarchy number)	

As shown in Figure 3-5, users can open a memory info file by clicking the “File” menu and selecting “Open”.

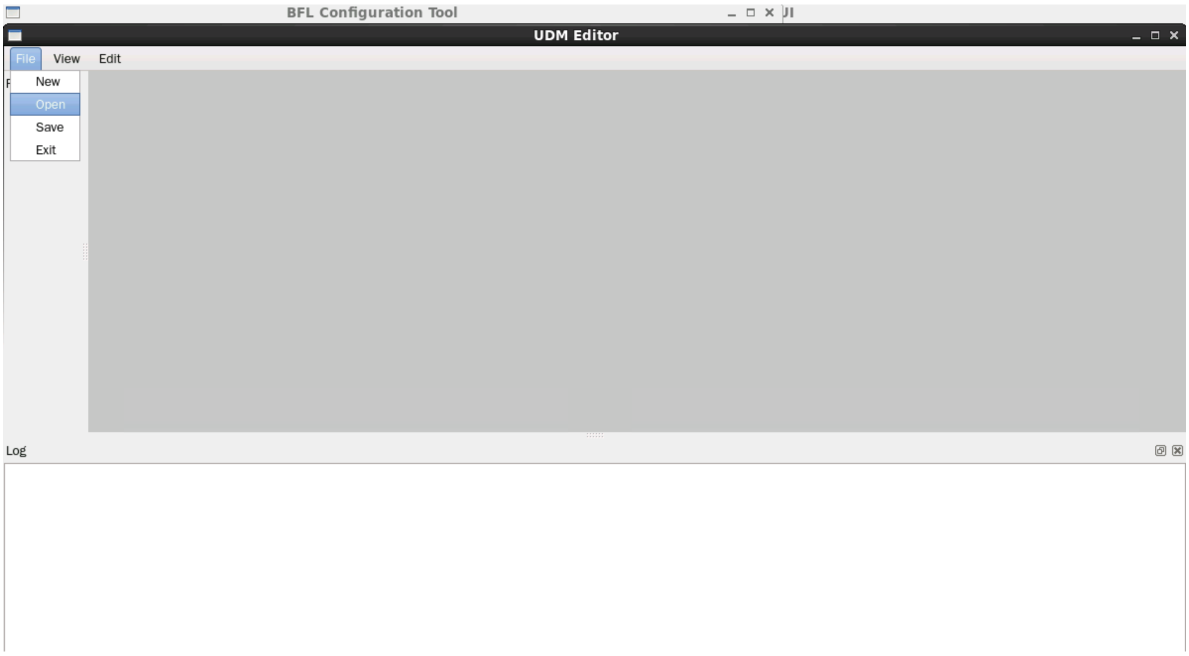
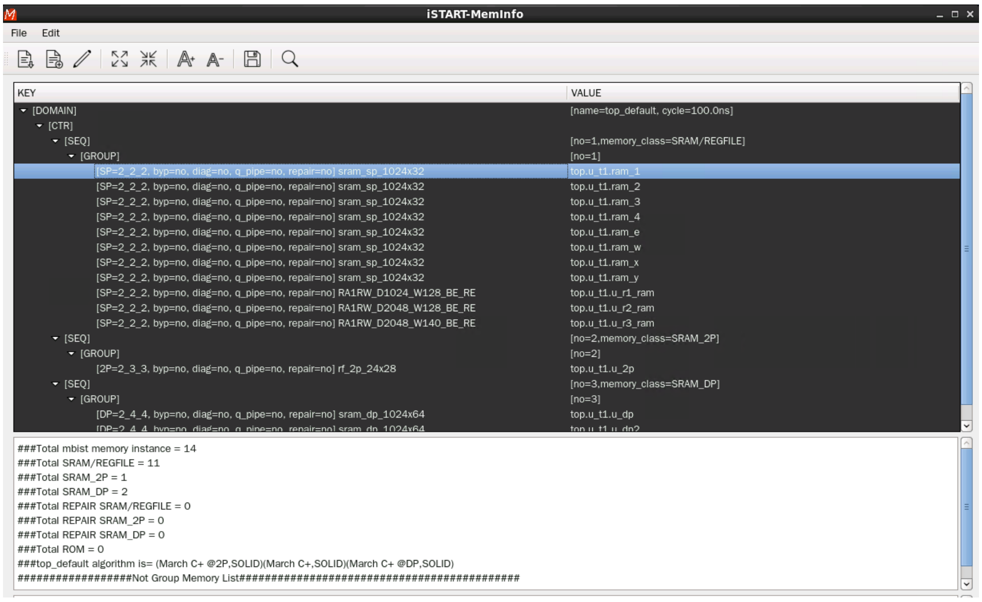


Figure 3-5 Open Memory Info File

Figure 3-6 is an example of the memory info file. For the detailed information, please refer to Chapter 7 in Application Notes.

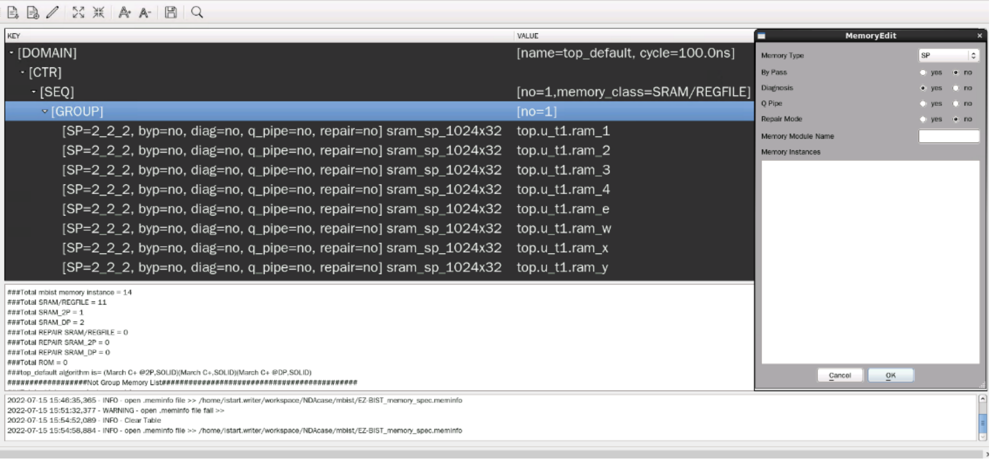


```
KEY                                                                         VALUE
▾ [DOMAIN]                                                                  [name=top_default, cycle=100.0ns]
  ▾ [CTR]
    ▾ [SEQ]                                                                 [no=1,memory_class=SRAM/REGFILE]
      ▾ [GROUP]                                                             [no=1]
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_1
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_2
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_3
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_4
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_e
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_w
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_x
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32         top.u_t1.ram_y
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] RA1RW_D1024_W128_BE_RE  top.u_t1.u_r1_ram
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] RA1RW_D2048_W128_BE_RE  top.u_t1.u_r2_ram
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] RA1RW_D2048_W140_BE_RE  top.u_t1.u_r3_ram
    ▾ [SEQ]                                                                 [no=2,memory_class=SRAM_2P]
      ▾ [GROUP]                                                             [no=2]
          [2P=2_3_3, byp=no, diag=no, q_pipe=no, repair=no] rf_2p_24x28             top.u_t1.u_2p
    ▾ [SEQ]                                                                 [no=3,memory_class=SRAM_DP]
      ▾ [GROUP]                                                             [no=3]
          [DP=2_4_4, byp=no, diag=no, q_pipe=no, repair=no] sram_dp_1024x64         top.u_t1.u_dp
          [DP=2_4_4, byp=no, diag=no, q_pipe=no, repair=no] sram_dp_1024x64         top.u_t1.u_dp2
```

```
###Total mbist memory instance = 14
###Total SRAM/REGFILE = 11
###Total SRAM_2P = 1
###Total SRAM_DP = 2
###Total REPAIR SRAM/REGFILE = 0
###Total REPAIR SRAM_2P = 0
###Total REPAIR SRAM_DP = 0
###Total ROM = 0
###top_default algorithm is= (March C+ @2P,SOLID)(March C+,SOLID)(March C+ @DP,SOLID)
##################Not Group Memory List############################################
```

Figure 3-6 Example of Memory Info File

As shown in Figure 3-7, users can right click “GROUP” and select “add mem” to add memories by batches according to the information described below.



```
KEY                                                                         VALUE
· [DOMAIN]                                                                  [name=top_default, cycle=100.0ns]
  · [CTR]
    · [SEQ]                                                                 [no=1,memory_class=SRAM/REGFILE]
      ▿ [GROUP]                                                             [no=1]
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_1
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_2
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_3
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_4
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_e
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_w
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_x
          [SP=2_2_2, byp=no, diag=no, q_pipe=no, repair=no] sram_sp_1024x32   top.u_t1.ram_y
```

```
###Total mbist memory instance = 14
###Total SRAM/REGFILE = 11
###Total SRAM_2P = 1
###Total SRAM_DP = 2
###Total REPAIR SRAM/REGFILE = 0
###Total REPAIR SRAM_2P = 0
###Total REPAIR SRAM_DP = 0
###Total ROM = 0
###top_default algorithm is= (March C+ @2P,SOLID)(March C+,SOLID)(March C+ @DP,SOLID)
##################Not Group Memory List############################################
```

```
2022-07-15 15:46:35,365 - INFO - open .meminfo file >> /home/istart.writer/workspace/NDAcase/mbist/EZ-BIST_memory_spec.meminfo
2022-07-15 15:51:32,377 - WARNING - open .meminfo file fail >>
2022-07-15 15:54:52,089 - INFO - Clear Table
2022-07-15 15:54:58,884 - INFO - open .meminfo file >> /home/istart.writer/workspace/NDAcase/mbist/EZ-BIST_memory_spec.meminfo
```

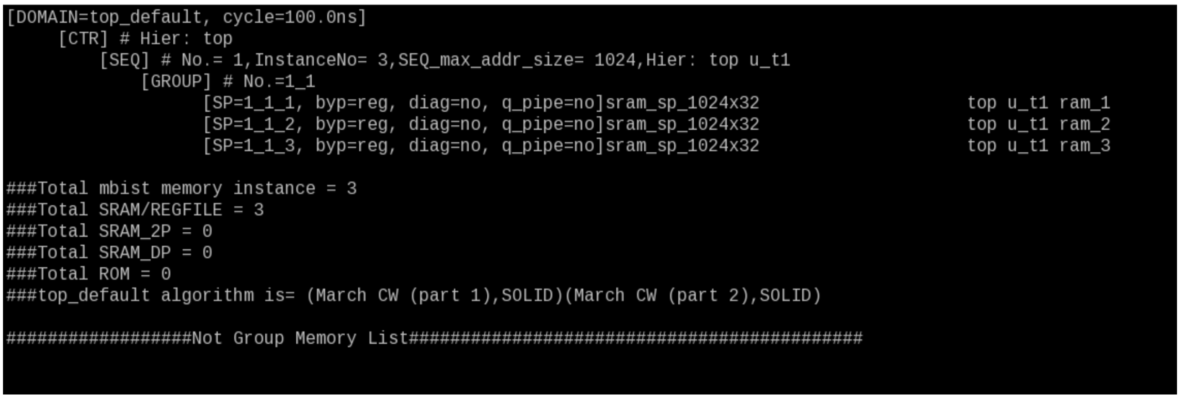
MemoryEdit

Memory Type: SP
By Pass: ○ yes ● no
Diagnosis: ● yes ○ no
Q Pipe: ○ yes ○ no
Repair Mode: ○ yes ● no
Memory Module Name:
Memory Instances:

Figure 3-7 Support Batches Adding and Multiple Formats

A memory info file includes the following items. For the detailed information, please refer to Chapter 7 in Application Notes.

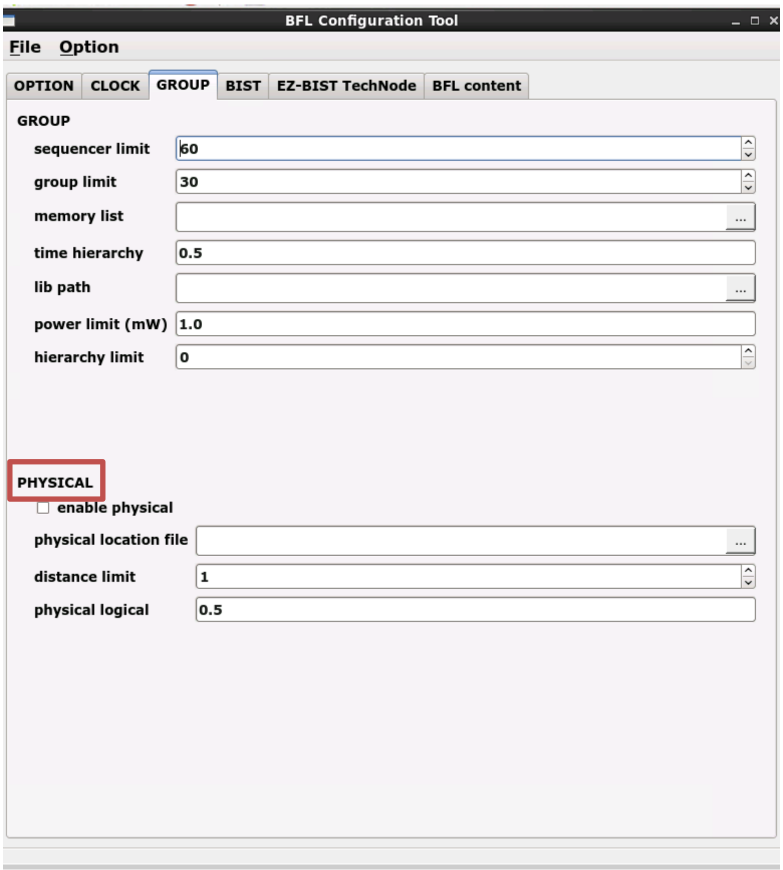
- **Clock domain**: It shows “memory clock domain name” and “testing clock cycle”.
- **Memory module**: It shows the “memory module name” and “memory hierarchy”.
- **Bypass**: Set the values of the bypass function.
- **Diagnosis**: Set the values of the diagnosis function.
- **Q_pipeline**: Set the value of the Q_pipeline option.
- **Group Architecture**: This option shows the grouping architecture information including the controller, sequencer, and group.
- **Design information**: This option shows the number of memory instances, memory types, and testing algorithms.



```
[DOMAIN=top_default, cycle=100.0ns]
     [CTR] # Hier: top
         [SEQ] # No.= 1,InstanceNo= 3,SEQ_max_addr_size= 1024,Hier: top u_t1
             [GROUP] # No.=1_1
                   [SP=1_1_1, byp=reg, diag=no, q_pipe=no]sram_sp_1024x32                    top u_t1 ram_1
                   [SP=1_1_2, byp=reg, diag=no, q_pipe=no]sram_sp_1024x32                    top u_t1 ram_2
                   [SP=1_1_3, byp=reg, diag=no, q_pipe=no]sram_sp_1024x32                    top u_t1 ram_3

###Total mbist memory instance = 3
###Total SRAM/REGFILE = 3
###Total SRAM_2P = 0
###Total SRAM_DP = 0
###Total ROM = 0
###top_default algorithm is= (March CW (part 1),SOLID)(March CW (part 2),SOLID)

##################Not Group Memory List############################################
```

Figure 3-8 Memory Info Setting Information

3.1.3. PHYSICAL Sub Function Block



GROUP

sequencer limit 60

group limit 30

memory list

time hierarchy 0.5

lib path

power limit (mW) 1.0

hierarchy limit 0

PHYSICAL

☐ enable physical

physical location file

distance limit 1

physical logical 0.5

Figure 3-9 PHYSICAL Sub Function Block

(For the detailed information, please refer to the table in the next page.)

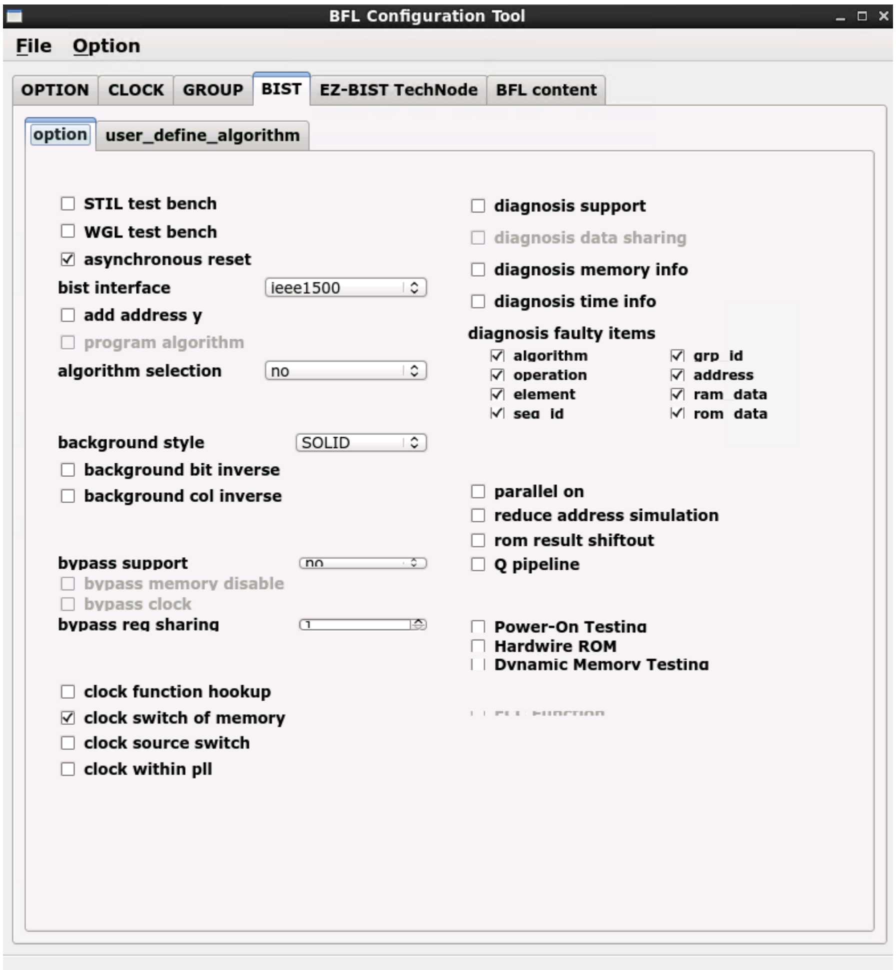
<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>enable_physical</td><td>No, Yes</td></tr>
<tr><td colspan="2">If this option is set to “yes”, EZ-BIST will auto-group based on the DEF (Design Physical Information) file.</td></tr>
<tr><td>physical_location_file</td><td>User defined</td></tr>
<tr><td colspan="2">Set the paths of the DEF file.</td></tr>
<tr><td>controller_scope</td><td>User defined</td></tr>
<tr><td colspan="2">After editing a SCOPE file, set the path of the SCOPE file. The scope information should be included with a controller name and position coordinate as follows.

Controller Name Position Coordinate (x1 y1) (x2 y2)

For example, top_default (10000 10000) (300000 400000)</td></tr>
<tr><td>physical_logical</td><td>0 <= value <= 1</td></tr>
<tr><td colspan="2">This option is to adjust the weight between physical coordinates and values defined in the time_hierarchy option.

For example:
<table>
<tr><td><i>set physical_logical = 0</i></td><td>EZ-BIST will calculate the number of intermediates based on an internal algorithm. Memory models which are located near this intermediate number will be merged into the same group.</td></tr>
<tr><td><i>set physical_logical = 1</i></td><td>EZ-BIST will execute memory grouping based on the value of the time_hierarchy option.</td></tr>
</table></td></tr>
</table>

3.2. BIST Function Block



☐ STIL test bench
☐ WGL test bench
☑ asynchronous reset
bist interface: ieee1500
☐ add address y
☐ program algorithm
algorithm selection: no
background style: SOLID
☐ background bit inverse
☐ background col inverse
bypass support: no
☐ bypass memory disable
☐ bypass clock
bypass req sharing: 1
☐ clock function hookup
☑ clock switch of memory
☐ clock source switch
☐ clock within pll

☐ diagnosis support
☐ diagnosis data sharing
☐ diagnosis memory info
☐ diagnosis time info
diagnosis faulty items
☑ algorithm
☑ operation
☑ element
☑ seq_id
☑ grp_id
☑ address
☑ ram_data
☑ rom_data
☐ parallel on
☐ reduce address simulation
☐ rom result shiftout
☐ Q pipeline
☐ Power-On Testing
☐ Hardwire ROM
☐ Dynamic Memory Testing

Figure 3-10 MBIST Function Block

Argument	Option
Description	
STIL_test_bench	No, Yes
Generate a test pattern with the STIL format (IEEE 1450-Standard Test Interface Language) for the tester machine when this option set to “yes”. Since the result in the default STIL format might be a lot of repeated codes, users can change it into the loop format by using command lines, -- *STILloopformat*. No: Not generate the test pattern with the STIL format Yes: Generate the test pattern with the STIL format	

Argument	Option
Description	
WGL_test_bench	No, Yes
Generate a test pattern with the WGL format (Waveform Generation Language) when this option is set to "yes". No: Not generate the test pattern with the WGL format Yes: Generate the test pattern with the WGL format	
bist_interface	basic, basicIO, ieee1500, ieee1149.1, ieee1687
Select the MBIST interface. Note: For more details of these interfaces, please refer to IO Pin Definition. Note: When users set **bist_interface** to "ieee1149.1", the output interface is IEEE 1149.7. Note: When users set **bist_interface** to "ieee1500", the output interface is IEEE 1149.1.	
add_address_y	No, Yes
This option defines MBIST algorithms and supports the Y direction. The generated testbench supports the X and Y addressing modes (X stands for the row of the memory, and Y stands for the column of the memory.) No: The MBIST pattern testing only supports the X direction. Yes: The MBIST pattern testing supports both X and Y directions.	

X_Y = 00	Write MBIST pattern in the X direction only.
X_Y = 01	Write MBIST pattern the X direction first, and then Y direction.
X_Y = 10	Write MBIST pattern in the Y direction first, and then X direction.
X_Y = 11	Write MBIST pattern in the Y direction only.

Note: This option does not support memories with a column width of "0". Note: To define the X or Y directions, users must modify the X_Y setting in the testbench file.	
clock_source_switch	No, Yes
This option is used to select the testing frequency while the **clock_within_pll** option and **clock_switch_of_memory** option is turned on. The MBIST circuit will have a dedicated test input signal named TRANS. Users can use this input signal to choose the testing frequency (from SCK or MCK). Note: The option must be set to "no" when clock tracing is turned on.	

Argument	Option
Description	
clock_mux_method	No, async_reg, GCK_MUX, glitch_free

This option enables users to add async_reg, GCK_MUX or glitch_free circuits at the memory clock port. Before using **clock_mux_method**, please make sure that **clock_source_switch** has been turned on.

No:	The tool operates with its initial functionality.
async_reg:	The TRANS signal will be synchronized by 2T.
GCK_MUX:	The macro folder will be generated in work_path. In the folder are two files, `ctrl_name_gck.v` and `ctrl_name_or2.v`. In the circuit, the GCK_MUX module will be generated for use.
glitch_free:	The macro folder will be generated in work_path. In the folder are two files, `ctrl_name_and2.v` and `ctrl_name_or2.v`. There will be an MBY port in CTR to support hookup of the SCAN port. In the circuit, the glitch_free module will be generated for use.

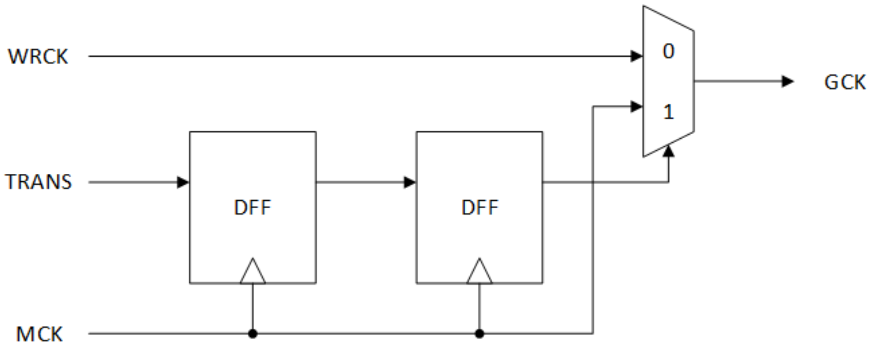


Figure 3-11 Scheme of the async_reg circuit

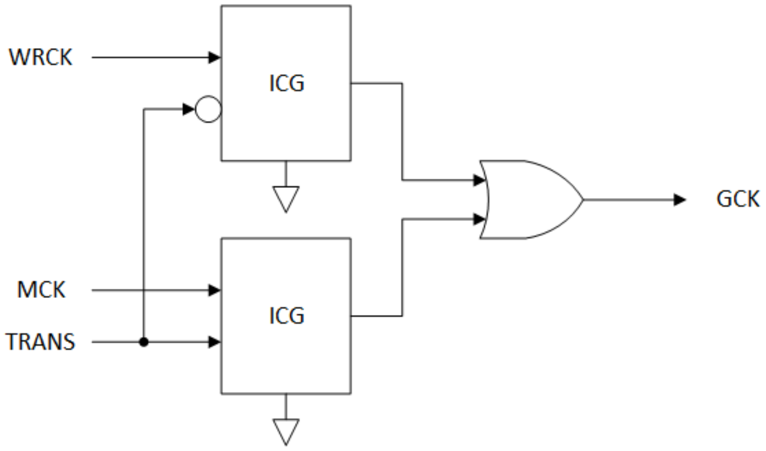
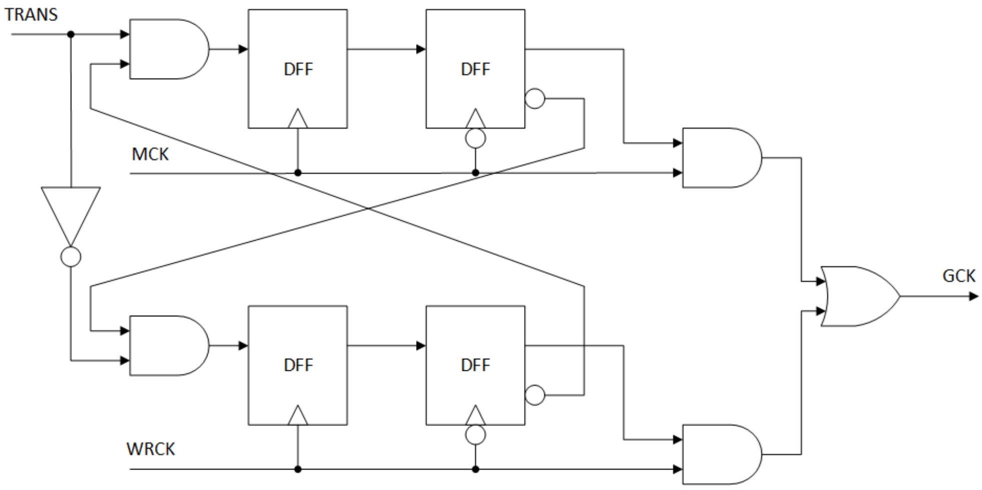


Figure 3-12 Scheme of the GCK_MUX circuit

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td colspan="2">
Figure 3-13 Scheme of the glitch_free circuit</td></tr>
<tr><td>clock_within_pll</td><td>No, Yes</td></tr>
<tr><td colspan="2">If this option is set to “yes”, the MBIST circuit will have another clock input source, SCK. This signal is used to connect with an ATE (Automatic Test Equipment) machine.
Note: The option must be set to “no” when clock tracing is turned on.</td></tr>
<tr><td>diagnosis_support</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to enable the diagnosis mode, which can provide users with the failure time and failed memory information.
No: Disable the Diagnosis mode
Yes: Enable the Diagnosis mode</td></tr>
<tr><td>diagnosis_data_sharing</td><td>No, Yes</td></tr>
<tr><td colspan="2">The tool can integrate diagnosis circuits into the sequencer to do diagnosis storage sharing to reduce the area of MBIST circuits when this option set to “yes”.</td></tr>
<tr><td>diagnosis_ctr_sharing</td><td>No, Yes</td></tr>
<tr><td colspan="2">To enable this function, diagnosis_data_sharing must be turned on first. When this option is set to “yes”, the diagnosis circuits in the sequencer will be transferred to the controller to share diagnosis storage, reducing the area of MBIST circuits.</td></tr>
<tr><td>diagnosis_faulty_items</td><td>algorithm, operation, element, seq_id, grp_id, address, ram_data, rom_data</td></tr>
<tr><td colspan="2">This option is used to select the output items of the diagnosis result based on the chip failure analysis requirement.</td></tr>
</table>

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td colspan="2">Example:
<i>set diagnosis_faulty_items = algorithm, operation, element, seq_id, grp_id, address, ram_data, rom_data</i></td></tr>
<tr><td>rom_result_shiftin</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to do ROM memory testing and import the signatures for internal verification. The scenario is used when the contents of the ROM memory is not confirmed at the initial development stage.

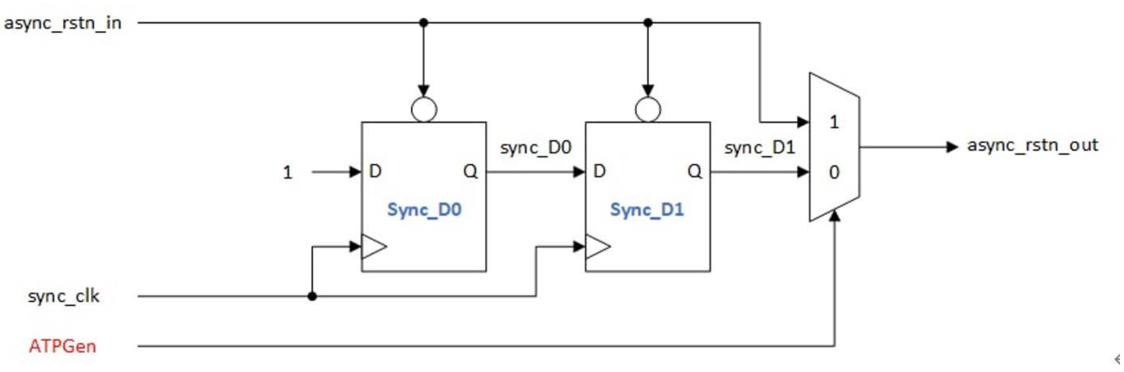
For example, when users set rom_result_shiftin to “yes” and the POT function is enabled, the testing results of ROM memory will be transferred to the internal circuit via commands.</td></tr>
<tr><td>rom_result_shiftout</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to do ROM memory testing and export the signatures for external verification. The scenario is used when the contents of ROM memory is not confirmed at the initial development stage.

For example, when user set rom_result_shiftout to “yes” and the testing results of the ROM memory will be transferred to the output interfaces via commands.</td></tr>
<tr><td>Q_pipeline</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to add an extra pipeline register to MBIST logics.

No: No extra register will be added to the data output of a memory model.
Yes: An extra register will be added to the data output of a memory model to enhanc operating timing of MBIST logics.</td></tr>
<tr><td>asynchronous_reset</td><td>No, Yes</td></tr>
<tr><td colspan="2">The option is used to specify asynchronous or synchronous reset of MBIST. The circuit can be differentiated into two types, “synchronous reset” and “asynchronous reset”. “Synchronous reset” indicates all DFFs are triggered to reset and then reset at the same time. “Asynchronous reset” indicates the reset of the circuit is based on the sequential order. In other word, this is not synchronous reset.

No: Synchronous reset will be applied with two DFFs. In addition, hookup the RSTN port (the MBIST reset signals) and the ATPGen port.
Yes: It indicates the asynchronous reset while one reset signal asserts. Additionally, hookup the RSTN port in the BII flow.

Figure 3-14 shows an example of synchronous/asynchronous circuits. When the ATPGen port is under the “scan mode”, the synchronous circuit will be bypassed and be regarded as the asynchronous circuit to select signals.</td></tr>
</table>

Argument	Option
Description	
 Figure 3-14 Example of Synchronous/Asynchronous Circuit	

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>atpg_reset</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is for users to reset the “Automatic Test Pattern Generation”. When the option is set to “yes”, EZ-BIST tool will string all the reset signals under MBIST into a series of ATPG_rstn.

Note: In the BII flow, hookup ATPG_rstn and ATPGen ports at the same time.
Note: When users set atpg_reset to “yes”, the ATPG signal will be inserted into the multiplexer (MUX) for the selection of ATPG_rstn or async_rstn_in signal as shown in Figure 3-15.

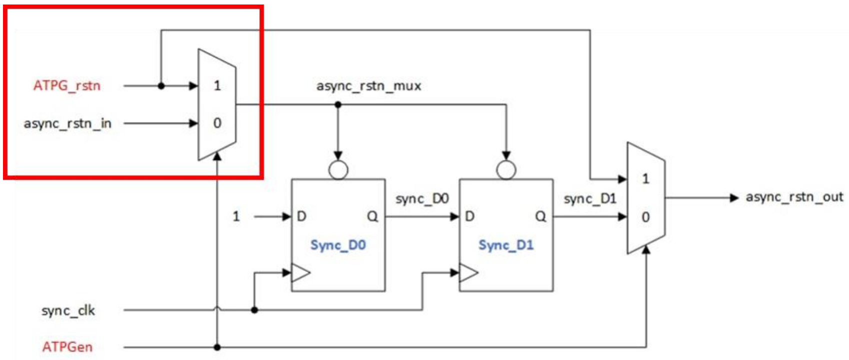


Figure 3-15 Example of ATPG Circuit</td></tr>
<tr><td>select_elem_testing</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is for users to do testing with user-defined test algorithms rather than EZ-BIST built-in algorithms by controlling input interfaces. When this function is turned on, users can select the algorithm elements in the SEQ, and the elements can be tested in the testbench.
A programmable algorithm is presented as a PROG entry. Figure 3-16 shows the testing commands while this option is turned on. Table 3-2 is the definition of these entries.

Note: User-defined testing algorithms cannot support ROM memory testing and the diagnosis function.

<table><tr><td>PROG</td><td>SEQ_ID</td><td>GRP_ID</td><td>MEB_ID</td><td>BG</td><td>ALG_CMD</td></tr></table>
← SDI_Command →

Figure 3-16 Commands for Programmable Algorithm Function</td></tr>
</table>

Table 3-2 Commands for Programmable Algorithm

Command	Description
PROG	PROG = 0, executing the EZ-BIST built-in algorithm PROG = 1, executing the user-defined algorithm
SEQ_ID	Sequencer ID of the memory
GRP_ID	Group ID of the memory
MEB_ID	Memory ID of the memory
BG	“SOLID” is the default background style. Only when “5A” is chosen, users can select four different modes to test. For more details, please refer to Table 3-3.
ALG_CMD	This ALG_CMD entry is based on March algorithm, users also can define it. While PROG = 1, MBIST circuits will execute user-defined algorithms. The width of the ALG_CMD entry is based on the March element definition.

Argument	**Option**
Description	
algorithm_selection	No, Outside, Scan

This option is for users to choose a single test algorithm or multiple test algorithms to test sequentially.

No:	Users can select algorithms which will be sequentially tested with MBIST/MBISR circuits.
Outside:	Users can select the test algorithm with the input port ALG and this input port will be added when the basicIO interface is defined.
Scan:	Users can launch the test through IEEE standard interfaces which is basic, IEEE 1149.1, IEEE 1500 or IEEE1687.

The number of bits in ALG varies depending on the algorithm configuration. In the following example, 2 algorithms are used (March 17N and Non-March BM), so 2 bits are required in ALG to select which test algorithm to execute.

ALG	Selected Algorithm
0x00	Both algorithms (March 17N → Non-March BM)
0x01	March 17N only
0x10	Non-March BM only
0x11	NA

Argument	**Option**
algorithm_loop_test	No, Yes

This option is for users to improve the loop mode testing efficiency. Some tests require a loop mode, but using multiple testing commands can cause delays between the commands.

No:	Not support continuous memory testing
Yes:	Support continuous memory testing

Users can send commands to control the BURN_IN signal to define the period of testing as Figure 3-17 when this option set to “yes”.

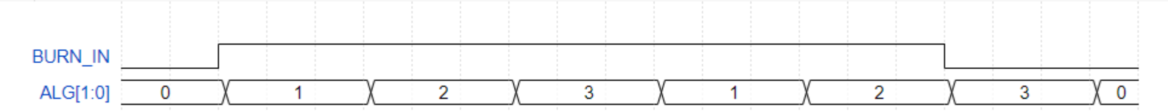


Figure 3-17 The Example Loop Test Waveform

Argument	**Option**
mem_protect_pwd	User defined

The memory output value will be safeguarded by the set password. Users must enter the correct password to retrieve the accurate data value.

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>background_style</td><td>SOLID, 5A</td></tr>
<tr><td colspan="2">
<p>The type of background_style can be set to “SOLID” and “5A” (Check Board), the contents are defined in the bg_table file.</p>
<p>There is an entry named BG (Background) in the SDI_Command. When background_style is set to 5A, the BG settings are shown as Table 3-3.</p>
<p>Note: If users adopt “March Mdsn1” as an algorithm, background_style cannot be set to “5A”.</p>
<p>Table 3-3 BG Field Definition</p>
<table>
<tr><th>BG [1:0]</th><th>Definition</th></tr>
<tr><td>00</td><td>SOLID + 5A</td></tr>
<tr><td>01</td><td>SOLID</td></tr>
<tr><td>10</td><td>5A</td></tr>
<tr><td>11</td><td>SOLID + 5A</td></tr>
</table>
</td></tr>
<tr><td>background_bit_inverse</td><td>No, Yes</td></tr>
<tr><td colspan="2">
<p>Bit inverse means that the BG testing data will be inversed by the increasing order or decreasing order of the memory address.
For example, the BG testing data of a 64x8 memory with SOLID BG is shown as Table 3-4.</p>
<p>Table 3-4 Example of Bit Inverse</p>
<table>
<tr><th>Memory Address</th><th>SOLID BG Test Data</th><th>Description</th></tr>
<tr><td>0000_0000</td><td>0000_0000</td><td>testing data non-inversed</td></tr>
<tr><td>0000_0001</td><td>1111_1111</td><td>testing data inversed</td></tr>
<tr><td>0000_0010</td><td>0000_0000</td><td>testing data non-inversed</td></tr>
<tr><td>0000_0011</td><td>1111_1111</td><td>testing data inversed</td></tr>
<tr><td>…</td><td>…</td><td>…</td></tr>
</table>
</td></tr>
</table>

Argument	Option
Description	
background_col_inverse	No, Yes

Column inverse means that the BG testing data will be inversed by the changes of the row memory address. If this changing time is larger than the CIC (Column Inverse Counts) number, the BG testing data will be inversed. The CIC number is defined by the memory Mux value.

For example, a 64x8 memory with Mux = 4 and the BG type = SOLID. The BG testing data is shown as Table 3-5.

Table 3-5 Example of Column Inverse

Memory Address	SOLID BG Test Data	Description
0000_0000	0000_0000	testing data non-inversed
0000_0001	0000_0000	
0000_0010	0000_0000	
0000_0011	0000_0000	
0000_0100	1111_1111	testing data inversed
0000_0101	1111_1111	
0000_0110	1111_1111	
0000_0111	1111_1111	
0000_1000	0000_0000	testing data non-inversed
0000_1001	0000_0000	
0000_1010	0000_0000	
0000_1011	0000_0000	
…	…	…

Argument	Option
Description	
user_define_bg	User defined

Users can specify the background test pattern via the setting of **user_define_bg**.

For example, if the width of the data is 4 bits:

Example 1:	When users assign **user_define_bg** to “3” and **background_style** to “SOLID”, then the testing pattern will be 0x3.
Example 2:	When users assign **user_define_bg** to “3” and **background_style** to “5A”, then the testing pattern will be 0x3,0xC,0x5,0xA.

Table 3-6 lists the example of user-defined background and the corresponding test patterns.

Table 3-6 Example of User-defined Background and Test Pattern

Background Style	User-defined Background	Test Pattern
SOLID	3	3
5A	3	3, C, 5, A
	3, 7	3, C, 7, 8

Argument	Option
Description	
retention	Handshake, Time

This option is for users to set the mode of retention.

Handshake: The retention time can be set in **retention_time** option in BFL file or in `testbech.v` file as shown in Figure 3-18.

Time: The retention time is fixed after being set in the **retention_time** option in the BFL file.

```
`timescale 1ns / 1ps

module stimulus;
parameter top_default_bcyc              = 100.0;
parameter RP_default_bcyc               = 100.0;
parameter tcyc                          = 100.0;
parameter rcyc                          = 100.0;
parameter cyc                           = tcyc;

parameter CORE_ID                       = {3{1'b1}};
parameter RP_default_RET_time           = 5.0;
parameter top_default_RET_time          = 5.0;
parameter TAP_IR_width                  = 1*3;

parameter test_result_width             = 10;
parameter test_command_width            = 17;
parameter max_config_width              = 17;
parameter WIR_width                     = 6;
parameter COMMAND_DR_ID                 = {2'b1, 2'b1, 2'b1};
parameter TEST_RESULT_DR_ID             = {2'd2, 2'd2, 2'd2};
parameter max_config_width              = 367;
parameter DIAG_RESULT_DR_ID             = {2'd3, 2'd3, 2'd3};
parameter top_default_ALG_width         = 2;
parameter top_default_SEQ_ID_width      = 2;
parameter top_default_GRP_ID_width      = 1;
```

Figure 3-18 Example of Retention Time Option in testbech.v

Argument	Option
Description	
retention_time	User defined

This option is used to define the retention time. The supported unit of retention time are listed in Table 3-7.

Table 3-7 Supported Units of Retention Time

Symbol	Unit
T	10^{12}
G	10^{9}
M	10^{6}
K or k	10^{3}
m	10^{-3}
u	10^{-6}
n	10^{-9}
p	10^{-12}

Some memory testing algorithms allow users to do retention testing.
For example, March-RET algorithm is <(wb) (SLP) <(rb) >(wa) (SLP) >(ra). The (SLP) element indicates the sleeping time is 1ms. If users want to extend the sleeping time more than 1ms, they can specify the retention time through **retention_time**.

```
define{BIST}
    …
    set retention_time = 1m
    …
end_define{BIST}
```

Note: The syntax of the retention time has different formats.

Take 1ms as an example,
The timing setting format in Verilog is **retention_time** = 1000000 (n).
The timing setting format in System Verilog is **retention_time** = 1000000 (n) or **retention_time** = 1m.

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>**bypass_support**</td><td>No, Wire, Reg</td></tr>
<tr><td colspan="2">Define whether the bypass circuit is implemented by wire or register.

No: Disable the bypass mode
Wire: Implement the bypass circuit by wire as Figure 3-19
Reg: Implement the bypass circuit by register as Figure 3-20

When entering the bypass mode, all input signals of the memory will be combined with normal access output. This option can increase logical testability and fault coverage.

Figure 3-19 Implementation of Bypass Circuit by Wire

Figure 3-20 Implementation of Bypass Circuit by Register

Note: If the **bypass_support** option is enabled, the ATPG clock (Scan) will switch to MBIST clock (MCK, memory clock) in the multi-source scenario.</td></tr>
</table>

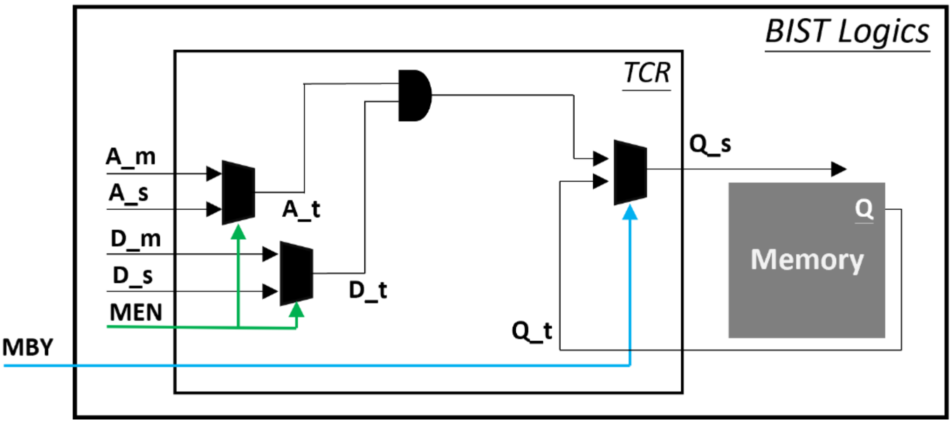


Figure 3-19 Implementation of Bypass Circuit by Wire

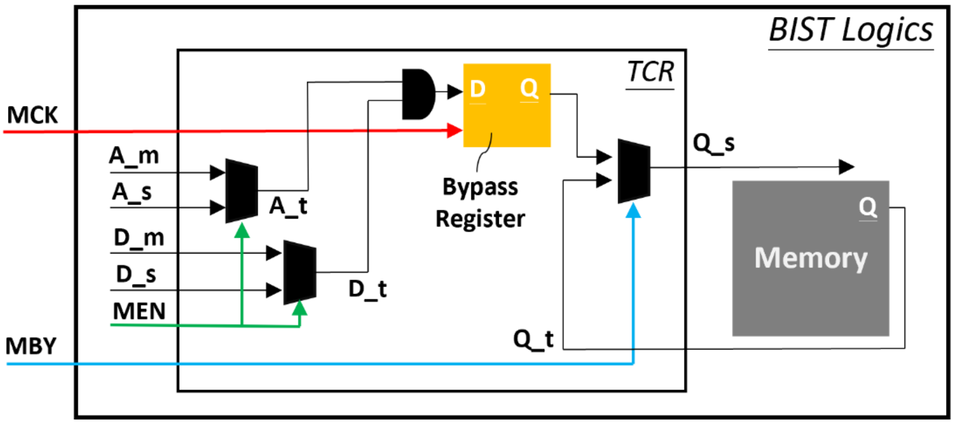


Figure 3-20 Implementation of Bypass Circuit by Register

Argument	Option
Description	
bypass_memory_disable	No, Yes

This option is available only when **bypass_support** is enabled. The memory CS (chip select) will be disabled. For example, when CS is active high, the parameter of CS will be "0". When CS is active low, the parameter of CS will be "1". The memory clock will be tied to "0".

No:	The memory CS will be enabled. The clock remains fixed on the speedy clock (i.e., the MBIST clock, MCK), shown as Figure 3-21.
Yes:	The memory chip select (CS) will be disabled (e.g., set to 0 if active high). The memory clock will be tied to 0, shown as Figure 3-22.

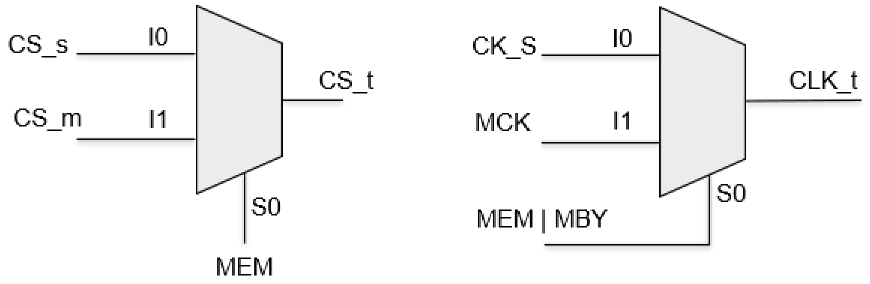


Figure 3-21 Implementation of CS and Clock Circuits (Option = No)

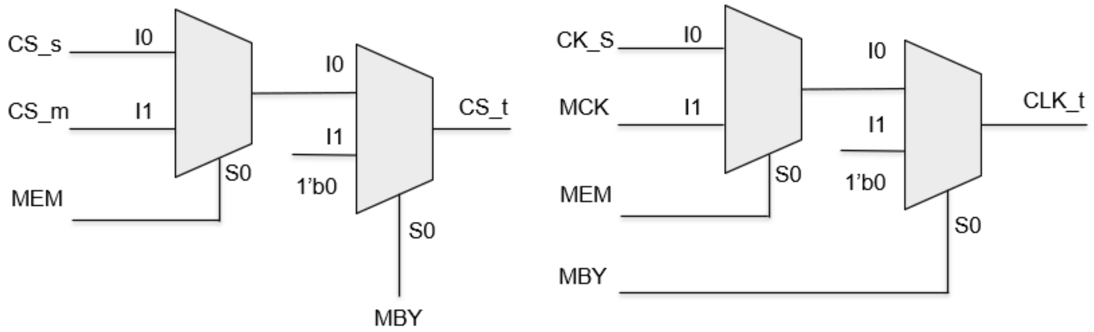


Figure 3-22 Implementation of CS and Clock Circuits (Option = Yes)

bypass_reg_sharing	1 <= value <= 1024

Users can set this option to define the register sharing number of bypass registers when **bypass_suppor** is set to “reg”. The range is between “1 ~1024”. EZ-BIST will base on this option to implement register sharing to reduce the area of bypass registers.

For example, when users assign **bypass_reg_sharing** to “4” and data output Q to “32” bits, the number of bypass registers will be “8” as Figure 3-23.

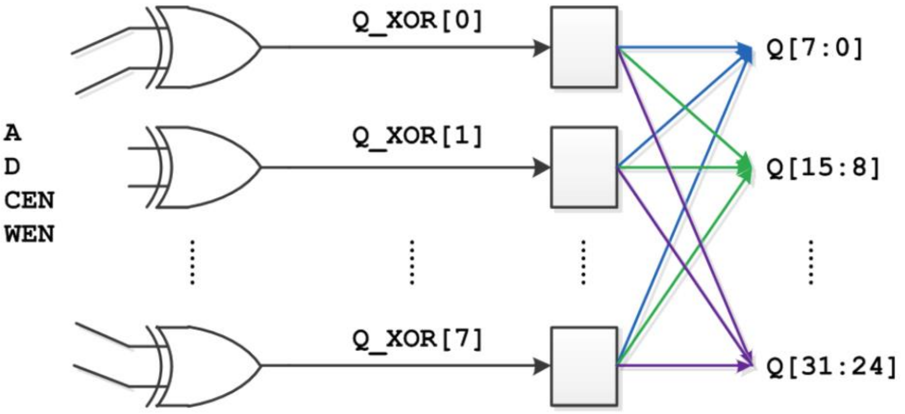


Figure 3-23 Example of Register Sharing

bypass_clock	No, Yes

If users decide to implement the bypass circuit by “reg” method, they can turn on this option to add a dedicated input port BCK for the bypass register and users can define the frequency of BCK based on their project requirements.

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>clock_function_hookup</td><td>No, Yes</td></tr>
<tr><td colspan="2">
This option is for users to hookup MCK with a memory functional clock. When this option is set to “yes”, MCK will be driven by the memory functional clock directly.

Note: The option is available only when clock tracing is turned on. Figure 3-24 shows the clock architecture of this option.

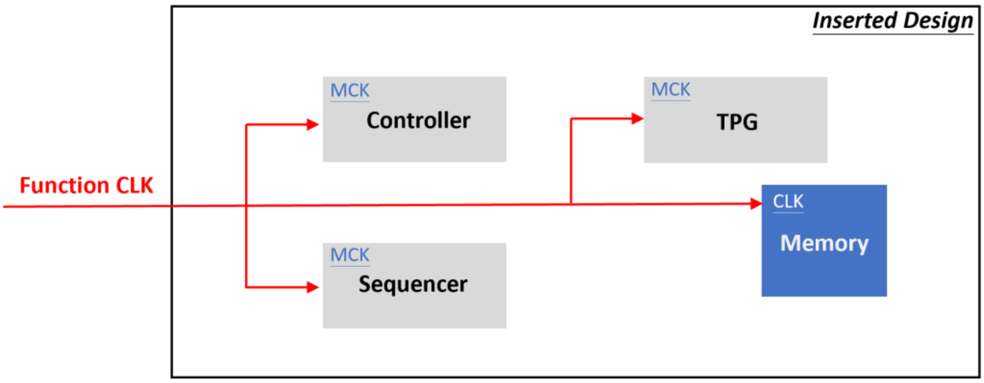


Figure 3-24 Clock Architecture of clock_function_hookup Option
</td></tr>
<tr><td>clock_switch_of_memory</td><td>No, Yes</td></tr>
<tr><td colspan="2">
When this option is set to “yes”, the clock signal of the memory model will be changed to MCK by clock multiplexer in the test mode. The clock signal of the memory model is running at the same frequency according to users’ requirements. Figure 3-25 shows the clock architecture of this option. The MCK also can be driven by the internal testing clock. Users can hookup it with internal clock signal in BII mode.

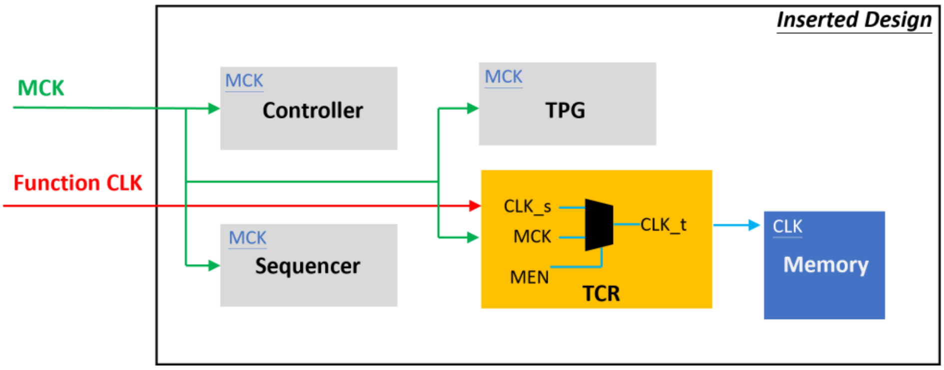


Figure 3-25 Clock Architecture of clock_switch_of_memory Option
</td></tr>
</table>

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>diagnosis_memory_info</td><td>No, Yes</td></tr>
<tr><td colspan="2">EZ-BIST will generate MBIST circuits with N-bits width LATCH_GO output signals when this option is turned on. N means the number of memory models and each bit of a LATCH_GO signal indicates one memory model. Figure 3-26 shows the waveforms of LATCH_GO signals. When the signal turns from high to low, it indicates that memory has failed.

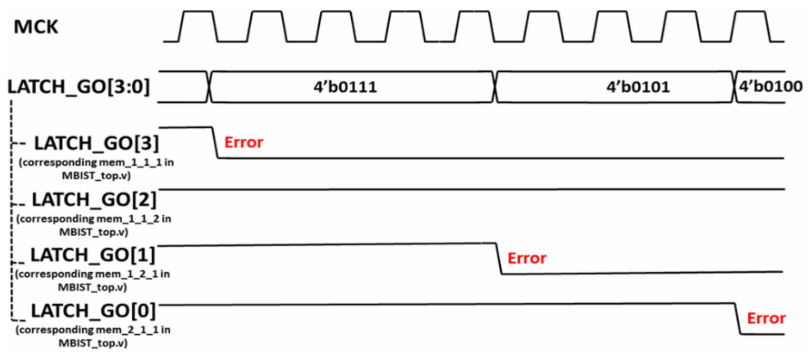


Figure 3-26 Diagnosis Fail Memory Information</td></tr>
<tr><td>diagnosis_time_info</td><td>No, Yes</td></tr>
<tr><td colspan="2">EZ-BIST will generate MBIST circuits with the MBIST_GO output signal when this option is turned on. If the memory fails, this signal will change from high to low and return to high in the next clock cycle as shown in Figure 3-27.

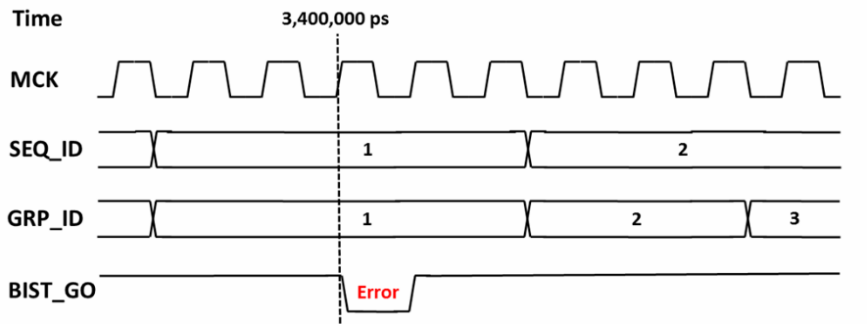


Figure 3-27 Diagnosis Fail Time Information</td></tr>
</table>

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><th colspan="2">Description</th></tr>
<tr><td>parallel_on</td><td>No, Yes</td></tr>
<tr><td colspan="2">Specify the memory to support parallel testing. When this option is set to “yes” and assign testbench parameter PRL_ON to “1”, all memories under a controller will launch the testing simultaneously.</td></tr>
<tr><td>reduce_address_simulation</td><td>No, Yes</td></tr>
<tr><td colspan="2">EZ-BIST executes testing with fixed four memory addresses as Table 3-8. This option speeds up simulation by reducing memory testing addresses. If the column width is zero, the testing address will be fixed to two memory addresses as Table 3-9.

Table 3-8 Fixed Four Memory Addresses
<table>
<tr><th>Memory Address Row</th><th>Memory Address Column</th></tr>
<tr><td>…000000</td><td>000000…</td></tr>
<tr><td>…000000</td><td>111111…</td></tr>
<tr><td>…111111</td><td>000000…</td></tr>
<tr><td>…111111</td><td>111111…</td></tr>
</table>
Table 3-9 Fixed Two Memory Addresses
<table>
<tr><th>Memory Address Row</th></tr>
<tr><td>…000000</td></tr>
<tr><td>···111111</td></tr>
</table></td></tr>
<tr><td>pot</td><td>No, basic, hw_rom, rom, ram</td></tr>
<tr><td colspan="2">When the system requires the Power_On testing, the following options are available. For more details, please refer to Chapter 9 in Application Notes.
<table>
<tr><td>No:</td><td>Disable the POT function.</td></tr>
<tr><td>Basic:</td><td>It indicates supporting some generic signals to enable or disable MBIST and the test results. This function only supports the RAM test.</td></tr>
<tr><td>hw_rom:</td><td>It indicates that the POT testing commands will be designed to hardwired circuits. This function supports the ROM test.</td></tr>
<tr><td>Rom:</td><td>It indicates that the POT testing commands will be stored in the ROM. This function supports the ROM test.</td></tr>
</table></td></tr>
</table>

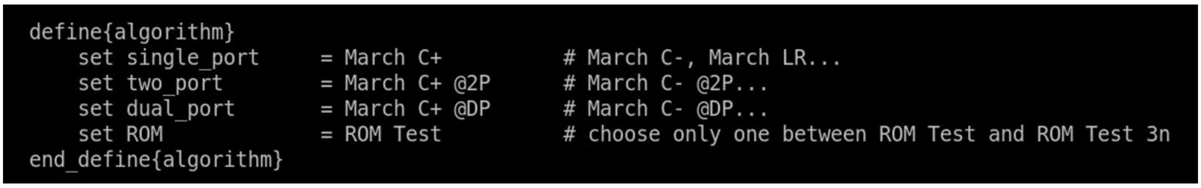
Ram:	An RAM interface is supported for POT. It enables users to dynamically adjust MBIST test instructions.

3.2.1. Default Algorithm Sub Function Block

EZ-BIST provides various testing algorithms for users to choose according to different testing requirements. Figure 3-28 shows the default setting of single-port memories is the March C+ algorithm. If users want to add more testing algorithms into MBIST circuits, they just need to add algorithms into this function block.

The ROM setting is used to set the algorithm for ROM, and there are two options: ROM test and ROM Test 3n.

Section 6.4 shows the testing algorithms provided by EZ-BIST.

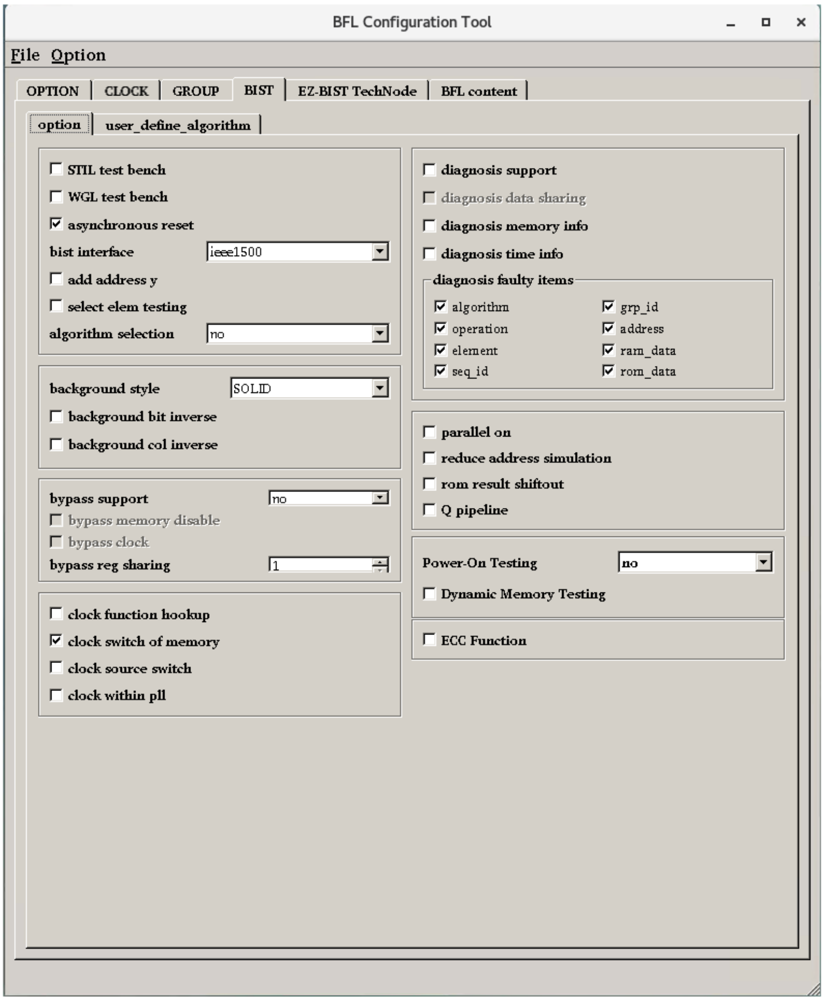


```
define{algorithm}
    set single_port     = March C+         # March C-, March LR...
    set two_port        = March C+ @2P     # March C- @2P...
    set dual_port       = March C+ @DP     # March C- @DP...
    set ROM             = ROM Test         # choose only one between ROM Test and ROM Test 3n
end_define{algorithm}
```

Figure 3-28 Default Algorithm Function Block

3.2.2. Programmable Algorithm Sub Function Block

As shown in Figure 3-29, users can set the programmable algorithm in the GUI mode.



☐ STIL test bench
☐ WGL test bench
☑ asynchronous reset
bist interface: ieee1500
☐ add address y
☐ select elem testing
algorithm selection: no

background style: SOLID
☐ background bit inverse
☐ background col inverse

bypass support: no
☐ bypass memory disable
☐ bypass clock
bypass reg sharing: 1

☐ clock function hookup
☑ clock switch of memory
☐ clock source switch
☐ clock within pll

☐ diagnosis support
☐ diagnosis data sharing
☐ diagnosis memory info
☐ diagnosis time info

☑ algorithm
☑ operation
☑ element
☑ seq_id
☑ grp_id
☑ address
☑ ram_data
☑ rom_data

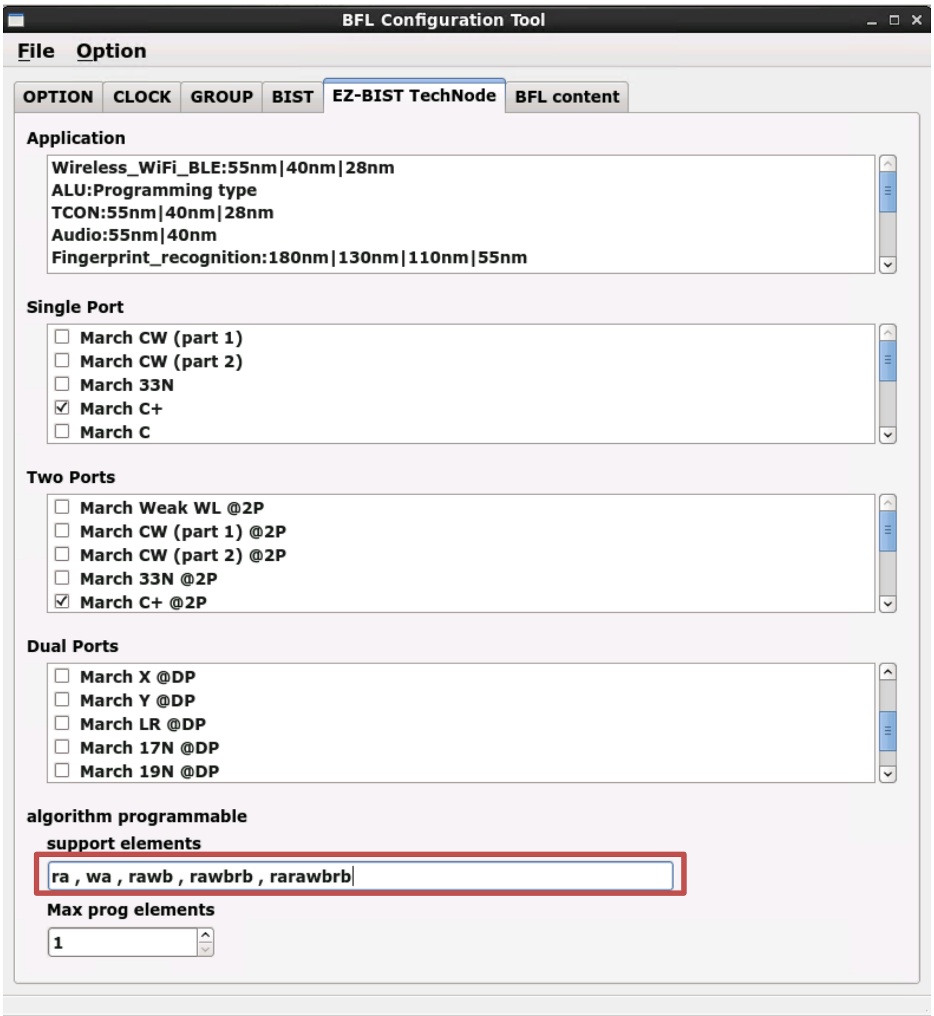
☐ parallel on
☐ reduce address simulation
☐ rom result shiftout
☐ Q pipeline

Power-On Testing: no
☐ Dynamic Memory Testing

☐ ECC Function

Figure 3-29 select_elem_testing

Figure 3-30 shows the select testing elements sub function block, describing the testing elements created by users.



Application

Wireless_WiFi_BLE:55nm|40nm|28nm
ALU:Programming type
TCON:55nm|40nm|28nm
Audio:55nm|40nm
Fingerprint_recognition:180nm|130nm|110nm|55nm

Single Port

- ☐ March CW (part 1)
- ☐ March CW (part 2)
- ☐ March 33N
- ☑ March C+
- ☐ March C

Two Ports

- ☐ March Weak WL @2P
- ☐ March CW (part 1) @2P
- ☐ March CW (part 2) @2P
- ☐ March 33N @2P
- ☑ March C+ @2P

Dual Ports

- ☐ March X @DP
- ☐ March Y @DP
- ☐ March LR @DP
- ☐ March 17N @DP
- ☐ March 19N @DP

algorithm programmable

support elements: ra , wa , rawb , rawbrb , rarawbrb

Max prog elements: 1

Figure 3-30 Select Testing Elements Sub Function Block

While users chose the programmable algorithm function, the ALG_CMD entry will be added for programming usage. Users can define elements of their own testing algorithms. This function is Test Element Change (TEC) (Figure 3-31), with which users can adjust the element sequence of testing algorithms according to their needs. Through just 4 elements, w, r, rWR and Rwr (shown as

Table 3-10), a standard March C+ algorithm can be combined and created.

The March C+ algorithm is an example of a memory testing algorithm provided by START. The content of this algorithm is:

>(wa)

>(ra, wb, rb)

>(rb, wa, ra)

<(ra, wb, rb)

<(rb, wa, ra)

<(ra),

and the number of March elements is 6 and supported elements = r, w, rWR and Rwr. In this case, the width of the ALG_CMD entry is 5 × 6 = 30 (6 indicates the element width/EOT, and the end of test should be 0) and the format definition of March element can be Direction, Parity and Operation as

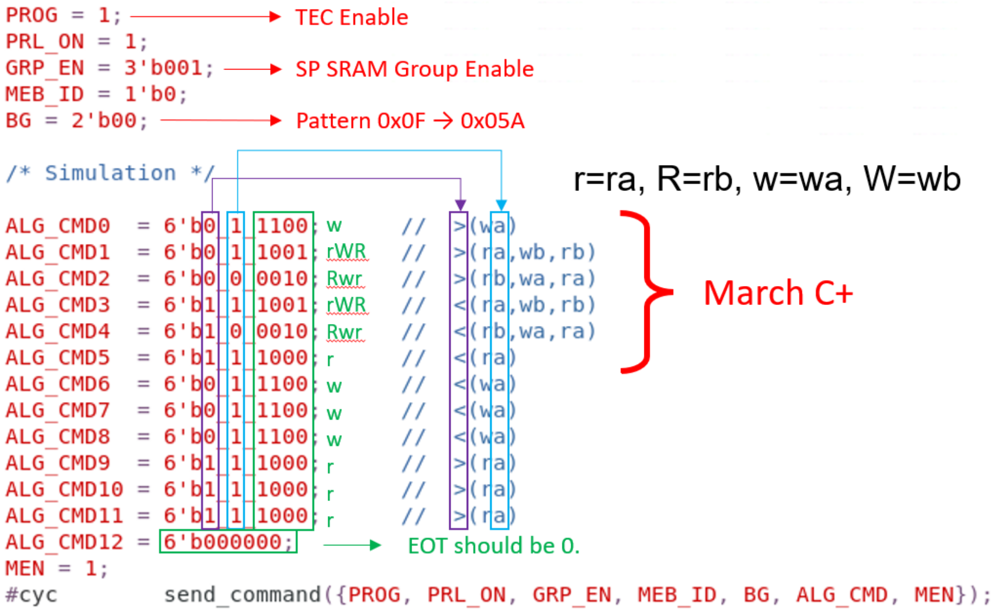
Table 3-10. Users can also find the definition in the `march_command.alias` file.

ALG_CMD = {ALG_CMD5, ALG_CMD4, … , ALG_CMD1, ALG_CMD0}

Table 3-10 Format of March CW Element

Type	Field	Width	Value	Description
Direction	>	1	0	Address increases
	<		1	Address decreases
Data Background	a	1	0	Data background
	b		1	Inverse data background
Operation	r	3	010	Read
	Rwr		001	Read, Write

	rWR		011	Read, Write, Read
	w		100	Write

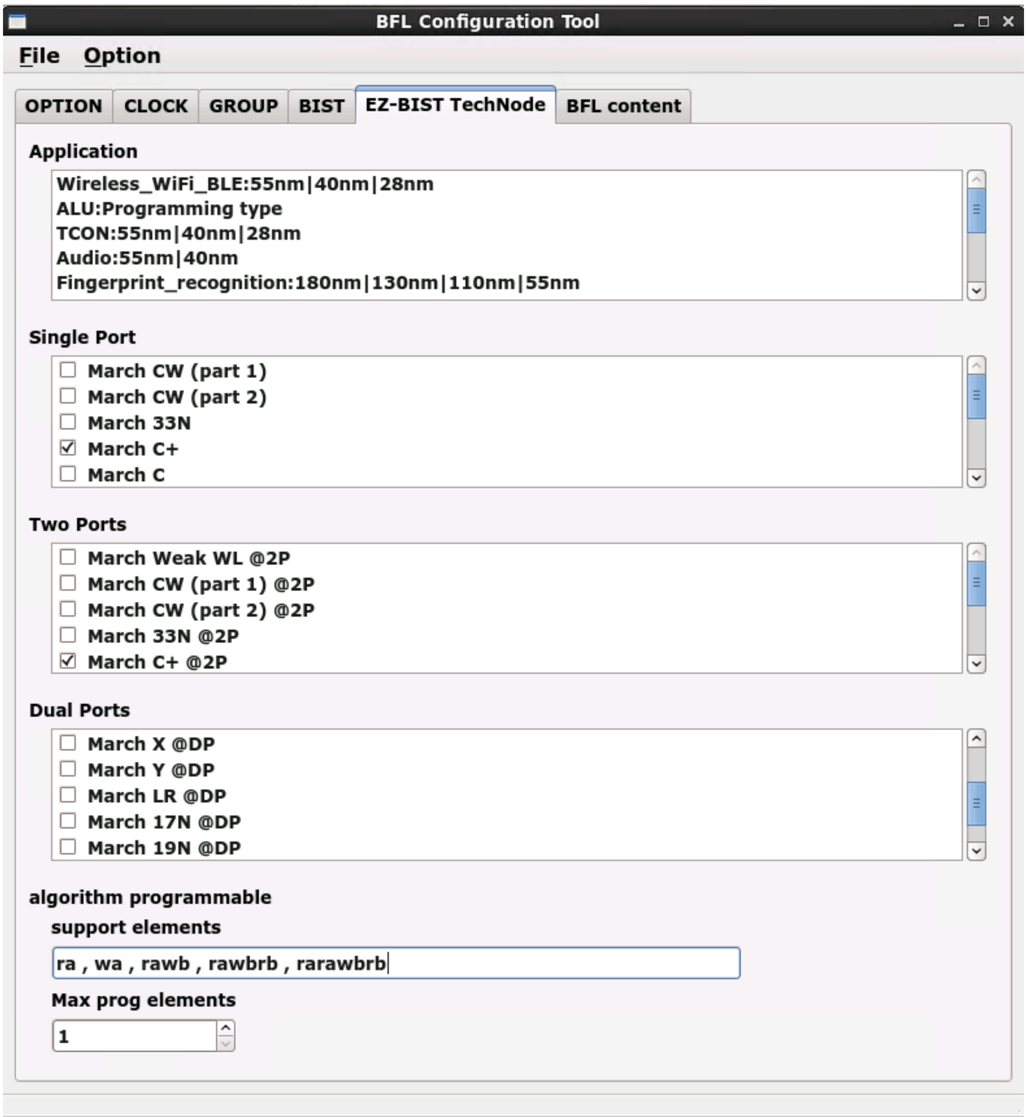


```
PROG = 1;
PRL_ON = 1;
GRP_EN = 3'b001;
MEB_ID = 1'b0;
BG = 2'b00;

/* Simulation */

ALG_CMD0  = 6'b011100; w    // >(wa)
ALG_CMD1  = 6'b011001; rWR  // >(ra,wb,rb)
ALG_CMD2  = 6'b000010; Rwr  // >(rb,wa,ra)
ALG_CMD3  = 6'b111001; rWR  // <(ra,wb,rb)
ALG_CMD4  = 6'b100010; Rwr  // <(rb,wa,ra)
ALG_CMD5  = 6'b111000; r    // <(ra)
ALG_CMD6  = 6'b011100; w    // <(wa)
ALG_CMD7  = 6'b011100; w    // <(wa)
ALG_CMD8  = 6'b011100; w    // <(wa)
ALG_CMD9  = 6'b111000; r    // >(ra)
ALG_CMD10 = 6'b111000; r    // >(ra)
ALG_CMD11 = 6'b111000; r    // >(ra)
ALG_CMD12 = 6'b000000;
MEN = 1;
#cyc        send_command({PROG, PRL_ON, GRP_EN, MEB_ID, BG, ALG_CMD, MEN});
```

Figure 3-31 ALG_CMD in Testbench

3.2.3. BFL TechNode

To avoid the possibility of dynamic defects in electronic devices which are manufactured from the advanced processes below 50nm, more accurate algorithms are needed for memory testing. EZ-BIST provides another way to select the algorithms. According to the needs of different processes and applications, EZ-BIST TechNode will check the recommended algorithms for users as Figure 3-32.



Application

Wireless_WiFi_BLE:55nm|40nm|28nm
ALU:Programming type
TCON:55nm|40nm|28nm
Audio:55nm|40nm
Fingerprint_recognition:180nm|130nm|110nm|55nm

Single Port

- ☐ March CW (part 1)
- ☐ March CW (part 2)
- ☐ March 33N
- ☑ March C+
- ☐ March C

Two Ports

- ☐ March Weak WL @2P
- ☐ March CW (part 1) @2P
- ☐ March CW (part 2) @2P
- ☐ March 33N @2P
- ☑ March C+ @2P

Dual Ports

- ☐ March X @DP
- ☐ March Y @DP
- ☐ March LR @DP
- ☐ March 17N @DP
- ☐ March 19N @DP

algorithm programmable

support elements

ra , wa , rawb , rawbrb , rarawbrb

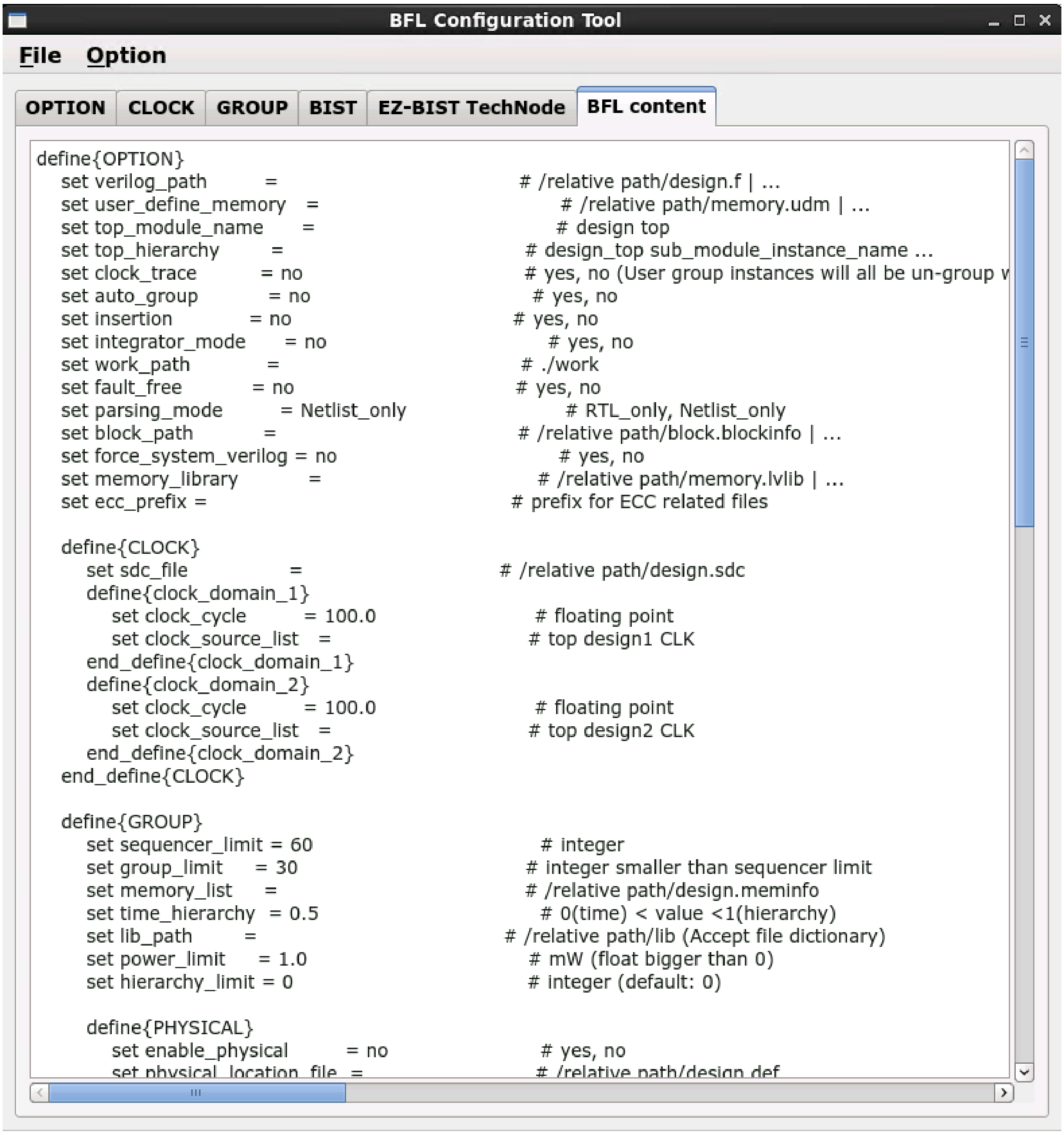
Max prog elements

1

Figure 3-32 BFL TechNode

3.2.4. BFL Setting File

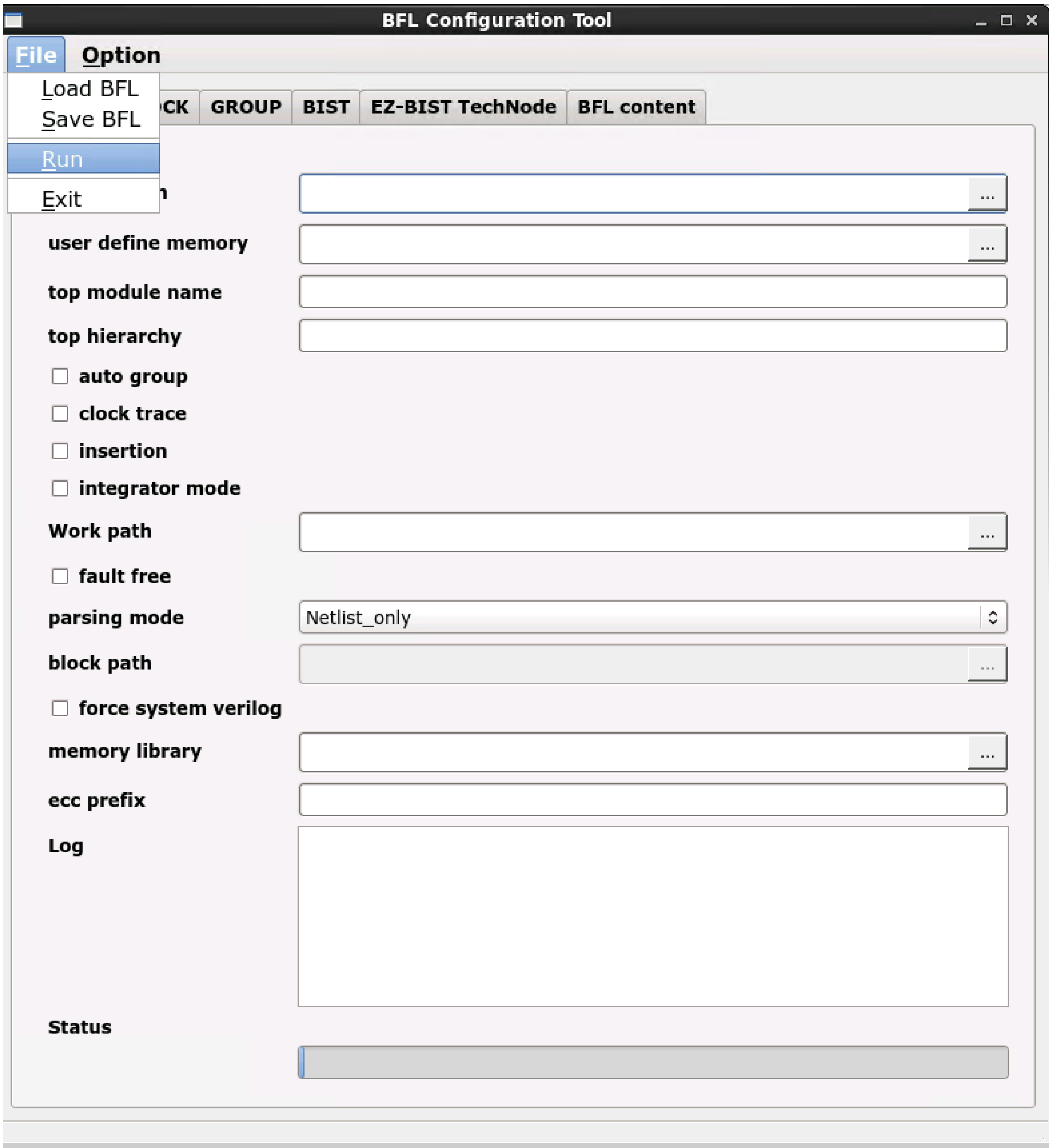
Users can check the settings of the BFL file in the BFL content page.



```
define{OPTION}
   set verilog_path          =                              # /relative path/design.f | ...
   set user_define_memory    =                                   # /relative path/memory.udm | ...
   set top_module_name       =                                   # design top
   set top_hierarchy         =                              # design_top sub_module_instance_name ...
   set clock_trace          = no                            # yes, no (User group instances will all be un-group w
   set auto_group           = no                             # yes, no
   set insertion           = no                            # yes, no
   set integrator_mode       = no                                # yes, no
   set work_path             =                              # ./work
   set fault_free          = no                            # yes, no
   set parsing_mode          = Netlist_only                        # RTL_only, Netlist_only
   set block_path            =                              # /relative path/block.blockinfo | ...
   set force_system_verilog = no                                   # yes, no
   set memory_library           =                             # /relative path/memory.lvlib | ...
   set ecc_prefix =                                        # prefix for ECC related files

   define{CLOCK}
      set sdc_file              =                        # /relative path/design.sdc
      define{clock_domain_1}
         set clock_cycle        = 100.0                      # floating point
         set clock_source_list   =                          # top design1 CLK
      end_define{clock_domain_1}
      define{clock_domain_2}
         set clock_cycle        = 100.0                      # floating point
         set clock_source_list   =                          # top design2 CLK
      end_define{clock_domain_2}
   end_define{CLOCK}

   define{GROUP}
      set sequencer_limit = 60                               # integer
      set group_limit      = 30                           # integer smaller than sequencer limit
      set memory_list      =                              # /relative path/design.meminfo
      set time_hierarchy  = 0.5                              # 0(time) < value <1(hierarchy)
      set lib_path          =                         # /relative path/lib (Accept file dictionary)
      set power_limit      = 1.0                          # mW (float bigger than 0)
      set hierarchy_limit = 0                             # integer (default: 0)

      define{PHYSICAL}
         set enable_physical          = no                   # yes, no
         set physical_location_file  =                      # /relative path/design.def
```

Figure 3-33 BFL Setting File

As shown in Figure 3-34, users can click “Run” from the “File” drop-down menu to complete the MBIST execution.



parsing mode: Netlist_only

Figure 3-34 Run the BFL Setting File

4. EZ-BIST Output Files

This chapter introduce EZ-BIST's output files and their usages. These output files are divided into Self-MBIST and Inserted-MBIST. Users can use these generated files to verify the MBIST circuit, and also verify the MBIST circuit integrated with customers' own logic design.

4.1. Self-MBIST Related Files

The generated self-MBIST related files include the self MBIST circuits (`.v`), test bench (`.v`), file-list file (`.f`), synthesis script (`.tcl`) and brief introduction file (`.html`). When users run simulations with these output files, it only simulates between MBIST circuits and memories.

Table 4-1 Self-MBIST Related Files

	EZ-BIST output	Description
Project file (.bid)	[filename]_spec.bid	This is an EZ-BIST project file and includes all settings of EZ-BIST.
Self MBIST circuits (.v)	[filename]_top.v [filename].v	`[filename]_top.v` includes memory models, fault memory models and MBIST circuits. It is integrated with MBIST circuits and memory models of original system design. `[filename].v` is HDL file of MBIST circuits.
Test bench (.v)	[filename]_tb.v	This is a test bench for testing `[filename]_top.v`.
File list (.f)	[filename].f	File-list file records `[filename]_top.v`, `[filename].v` and memory models. This file is used for simulation.
Synthesis script (.tcl)	[filename].tcl	This is a script file for synthesis of MBIST circuits.

4.2. Insert MBIST Related Files

EZ-BIST can insert MBIST circuits into customers' design. Users can verify the inserted-MBIST with their own system circuit. The following table shows the related files of the insert MBIST circuits.

Table 4-2 Insert MBIST Related Files

	EZ-BIST output	Description
Inserted MBIST circuits (.v)	[design]_INS.v [design]_INS_f.v	The file `[design]_INS.v` integrate MBIST circuits with user's system designs. The [design] is the name of user's system designs. This file does not include fault memory models. Different from `[design]_INS.v`, `[design]_INS_f.v` is integrated with fault memory models.
Test bench (.v)	[filename]_tb_INS.v	This is a test bench for testing `[design]_INS.v`.
File list (.f)	[filename]_INS.f [filename]_INS_FAULT.f	File-list `[filename]_INS.f` records `[design]_INS.v`, memory models and MIBST circuits. Different from file-list `[filename]_INS.f`, `[filename]_INS_FAULT.f` also includes fault memory models.

4.3. "Generate for Loop" Related Files

When the original design includes the use of a "Generate for Loop" syntax, START will additionally output files containing the "EXP" keyword in their filenames.

Table 4-3 "Generate for Loop" Related Files

EZ-BIST output	Description
[filename]_EXP.v:	This file is an intermediate file generated by expanding the "Generate for Loop" syntax in the original design.
[filename]_EXP.f	The file list `[filename]_EXP.f` records `[filename]_EXP.v` and the file list of memory modules.
[filename]_EXP_INS.v	This file is the result of inserting MBIST/MBISR-related logic based on `[filename]_EXP.v`.
iSTART_EXP_MAP.log	A filename mapping table between the EXP files and the original design files.

4.4. Generate Folders

The following table shows the generated folders when executing EZ-BIST.

Table 4-4 Generated Folders

	EZ-BIST Output	Description
REPORT		This folder is used to save the results of synthesis.
FAULT_MEMORY	[mem_name]_f.v fault_memory.f	`[mem_name]_f.v` is fault memory models. Some values inside of memory are tied to 0 or 1. This is used to verify functional correctness of MBIST circuits. The file-list `fault_memory.f` records all generated fault memory models.
fusion	[ctr_name]_ICL.icl [ctr_name]_PDL.pdl [ctr_name]_INS_fusion.f	When the interface is configured as IEEE 1687, a folder named fusion will be automatically generated. In BFL, the fusion folder will contain one set of files for each controller, including `[ctr_name]_ICL.icl`, `[ctr_name]_PDL.pdl`, and `[ctr_name]_INS_fusion.f`. The number of file sets corresponds to the number of controllers. In BII, the fusion folder will contain a single set of integration-level files: `[ctr_name]_ICL.icl`, `[ctr_name]_PDL.pdl`, and `[ctr_name]_INS_fusion.f`.

4.5. Makefile

EZ-BIST also generates Makefile which includes related commands of simulation and synthesis for users to verify their designs. Using Makefile, it can easily run various simulations along with MBIST circuits. Table 4-5 shows the commands of Makefile.

Table 4-5 Commands of Makefile

	Command	Description
Self-MBIST simulation	make [bistname] FUNC=tb	It is used to run self MBIST simulation with `[bistname]_tb.v` and `[bistname].f`. The simulation results will be printed out in the command line window.
Self-MBIST simulation with fault memories	make [bistname] FUNC=tb_f	It is used to run self MBIST simulation with `[bistname]_tb.v`, `[filename].f` and fault memory models. This simulation will show “Failed” because MBIST has detected faults in the memory models.
MBIST circuits synthesis	make [bistname] FUNC=dc	It is used to run synthesis with `[bistname].tcl` scripts using Design Compiler. The output will be saved into the REPORT folder.
Check syntax of self MBIST circuits with nLint	make [bistname] FUNC=lint	It is used to run syntax check with `[bistname].f` by using nLint. The checking result will be saved to file `[bistname]_lint.log`.
Remove generated files	make clean	It is used to remove generated files including `*.log`, `*.fsdb`,`*.db`, `*.sdf` and `*.rpt` files in the REPORT folder.
Inserted-MBIST simulation	make [bistname] FUNC=tb_INS	It is used to run the Inserted MBIST simulation with `[bistname]_tb_INS.v` and `[bistname]_INS_FAULT.f`, the simulation results will be printed out in the command line window. This command is available while the BFL option **insertion** is "yes".
Inserted-MBIST simulation with fault memories	make [bistname] FUNC=tb_INS_f	It is used to run the Inserted MBIST simulation with `[bistname]_tb_INS.v`, `[bistname]_INS_FAULT.f` and fault memory models. The simulation results will show “Failed” because MBIST has detected faults in the memory

		models.
Check syntax of inserted-MBIST circuits with nLint	make [bistname] FUNC=lint_INS	It is used to run syntax check with `[bistname]_INS_FAULT.f` by using nLint. This checking result will be saved to file `[bistname]_lint_INS.log`.
Formal checking	make [bistname] FUNC=fm	It is used to run formal checking with `[bistname]_fm.tcl`. The output message will be saved into `[bistname]_fm.log`.

4.6. Macro File

iSTART's latch-based clock gating cell model is `*_GCK.v` (* will be generated according to the module name in customers' designs). It can be synthesized in RTL modeling. However, to control clock skews, it is preferrable to integrate clock cells from the standard library.

Note: Please change each module in the macro file into the corresponding standard cell. Figure 4-1 is the example of a clock gating module. Here "ctr_name" means the prefix name coming from the controller name in the customer's design.

```
module ctr_name_gck (clk_out, clk_en, clk_in, test_en);

input clk_in;
input clk_en;
input test_en;

output clk_out;

`ifdef SYNTHESIS
    GCK_VENDOR_CELL gck(
            Q(clk_out)
            E(clk_en)
            TE(test_en)
            CK(clk_in)
            );
`else
        reg latch_out;

        assign clk_out = clk_in & latch_out;

        always @(clk_in or clk_en or test_en) begin
          if (~clk_in) begin
            latch_out = clk_en | test_en;
```

```
            end
          end
`endif

endmodule
```

Figure 4-1 Clock Gating Logic for Simulation and Synthesis

Figure 4-2 shows the schematic diagram of a clock gating cell with the waveform.

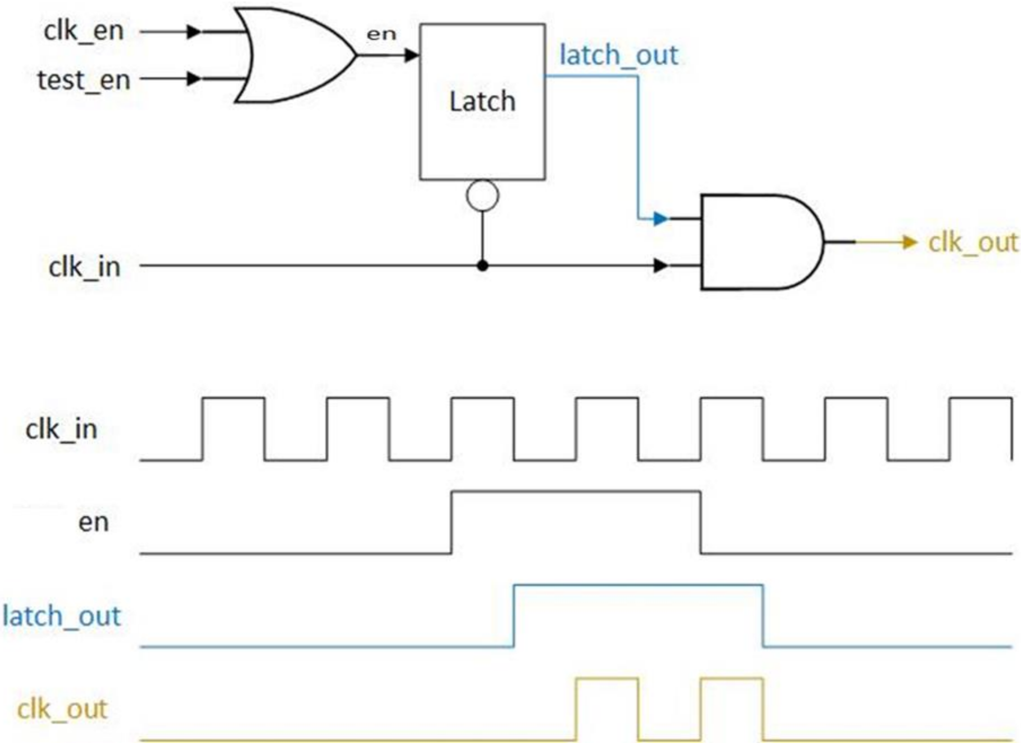


Figure 4-2 Clock Gating Cell with Waveform

5. BII File

EZ-BIST provides a BII (Integration Information) File for the integration task, which is in charge of integrating different MBIST controllers with an integrator module and then use IEEE1149.1 interface to communicate with ATE. This is used to save the pin count of the chip level. We will introduce the options of a BII file in this chapter.

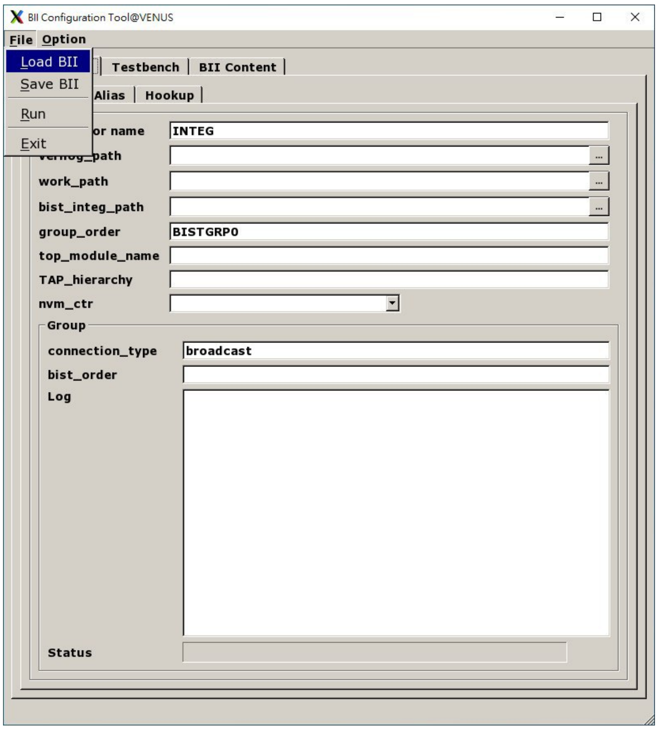
5.1. Integrator Function Block

Users can define the hookup pin mapping settings and order of different MBIST controller in the following function block.

define{Integrator}[Name]
…
end_define{Integrator}

The parameter, [Name] can be modified by users, and this will be the module name of the generated integrator module. This integrator module will integrate the WSI signal and WSO signal of each MBIST controller.

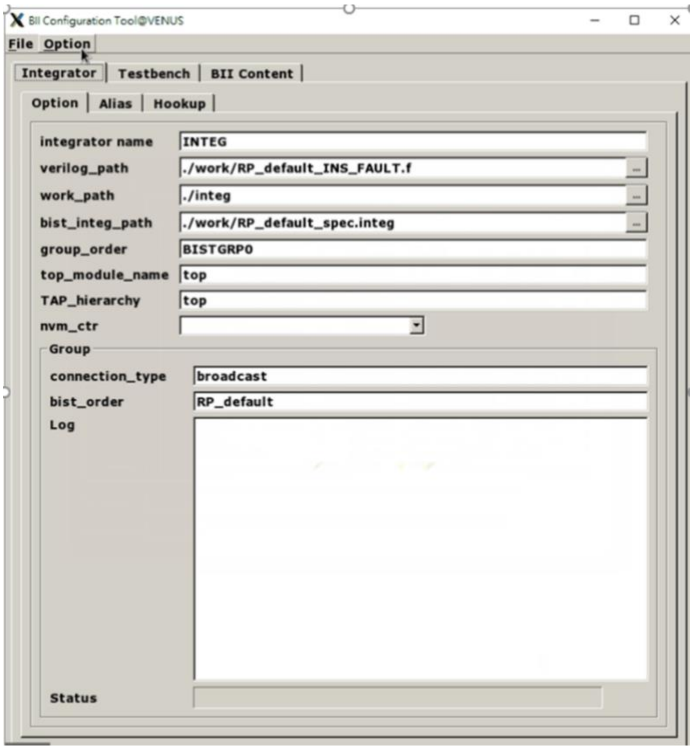
Figure 5-1 shows an example to load the existing BII file as the default setting.



or name: INTEG
group_order: BISTGRP0
connection_type: broadcast

Figure 5-1 Load BII

The options of the integrator function block are shown in Figure 5-2.



integrator name	INTEG
verilog_path	./work/RP_default_INS_FAULT.f
work_path	./integ
bist_integ_path	./work/RP_default_spec.integ
group_order	BISTGRP0
top_module_name	top
TAP_hierarchy	top
nvm_ctr	
connection_type	broadcast
bist_order	RP_default

Figure 5-2 Options of Integrator Function Block

The following is the list of BII parameters and their functionalities:

Argument	Option
Description	
group_order	User defined
This option is for users to define the ordering of an MBIST controller by setting the group sub function block. The testing sequence will follow the setting of **group_order**.	
top_module_name	User defined
This option is for users to define the top level module of their design.	
TAP_hierarchy	User defined
This option is for users to define the hierarchy of integrator module.	
verilog_path	User defined
Specify the file list which is generated by the BFL flow. For example, If the BFL option, **fault_free** is set to “yes”, the generated filelist file is `*_INS.f`. If the BFL option, **fault_free** is set to “no”, the generated filelist file is `*_INS_FAULT.f`. Users can assign “`*_INS.f`” or “`*_INS_FAULT.f`” to the **verilog_path** option.	
work_path	User defined
Specify the path of the working directory of the BII flow. All generated files in the BII flow will be saved to **work_path**.	
bist_integ_path	User defined
Set the path of the integration specification file `*_spec.integ`. Users can assign more than one integration specification files and separate them by the vertical bar “|”. For example, *bist_1_spec.integ* | *bist_2_spec.integ* | *bist_3_spec.integ.*	
skip_include_check	No, Yes
No: Transform all included paths in the output files into absolute paths Yes: Only transform the included paths in the modified files (which are named with keyword “_INS”) into the absolute path	
mem_prot_dec	User defined
This function is used for decryption and should be set together with BFL's mem_protect_pwd. Users need to enter the correct password to perform decryption.	

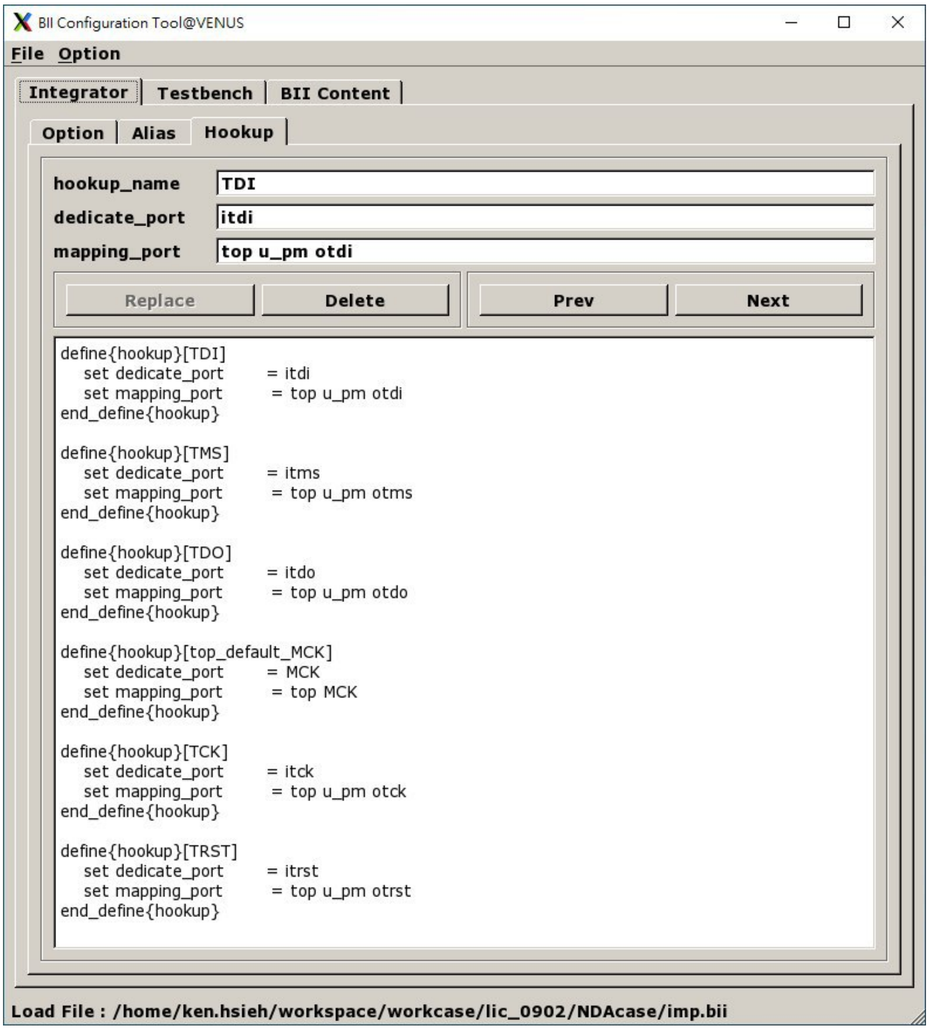
Argument	Option
Description	
postfixinfo	User defined
This option is for the bottom-up flow. If users plan to instantiate a harden module multiple times, a `*.postfixinfo` file need to be set in the BII file. The `*.postfixinfo` file will be generated by the tool after BIST logics insertion for the top module. Example: *set postfixinfo = ./mbist_top/EZ-BIST_top.postfixinfo*	
serial _order	User defined
The option is used to specify the memory testing order under the individual controller group. If the option **parallel_on** in the BFL file is “yes”, the memory will be tested by one controller sequentially one after another. For some particular cases, users want to test memories under more than one controller at the same time. By using the **serial_order** option, users can assign the controller group priority testing order, and the controller group contains one or more controllers. For example, when users assign **serial_order** to “top_default0, top_default1 | RP_default0 | RP_default1” and set “**parallel_on**” to “yes”. In this case: The priority testing order is [top_default0 & top_default1] => [RP_default0] => [RP_default1] Note: Each memory controller under a group separated by comma “,” is tested at the same priority order. An individual testing controller group is separated by a vertical bar “|”.	

5.1.1. Hookup Sub Function Block

EZ-BIST can support to implement the hookup function automatically. When the MBIST has been completed, users can get the `*.integ` file in the MBIST folder. The `*.integ` file provides the hookup pins shown in Figure 5-4. Furthermore, users can define the hookup pin information and pin the remapping information in hookup sub function block.

The definitions of hookup sub function blocks in the BII file are defined as follows:

define{hookup}[signal]
…
end_define{hookup}



hookup_name TDI
dedicate_port itdi
mapping_port top u_pm otdi

```
define{hookup}[TDI]
    set dedicate_port       = itdi
    set mapping_port        = top u_pm otdi
end_define{hookup}

define{hookup}[TMS]
    set dedicate_port       = itms
    set mapping_port        = top u_pm otms
end_define{hookup}

define{hookup}[TDO]
    set dedicate_port       = itdo
    set mapping_port        = top u_pm otdo
end_define{hookup}

define{hookup}[top_default_MCK]
    set dedicate_port       = MCK
    set mapping_port        = top MCK
end_define{hookup}

define{hookup}[TCK]
    set dedicate_port       = itck
    set mapping_port        = top u_pm otck
end_define{hookup}

define{hookup}[TRST]
    set dedicate_port       = itrst
    set mapping_port        = top u_pm otrst
end_define{hookup}
```

Load File : /home/ken.hsieh/workspace/workcase/lic_0902/NDAcase/imp.bii

Figure 5-3 Hookup Sub Function Block

Consequently, the BII hookup information table in `*.integ` file might differ depending on the user’s interface.

In Figure 5-4, it shows the IEEE 1149.1 JTAG interface. EZ-BIST supports several interfaces, such as basic, basicIO, IEEE1149.7, and IEEE1149.1.

```
 # BII flie hookup information table
    #       interface TCK      => define{hookup}[TCK]
    #       interface TRST     => define{hookup}[TRST]
    #       interface TMS      => define{hookup}[TMS]
    #       interface TDI      => define{hookup}[TDI]
    #       interface TDO      => define{hookup}[TDO]
    #       controller clock   => define{hookup}[top_default_MCK]
    #       BIST reset in => define{hookup}[RSTN]
end_define{BIST}
```

Figure 5-4 BII File Hookup Information Table in *.integ File

Argument	**Option**
Description	
hookup	User defined
It indicates a hookup sub function block.	
signal	User defined
It indicates the signals on the integrator module and will be connected with the mapping port. This signal could be IEEE 1149.1, IEEE 1149.7 signals, IEEE 1687 signals, MCK, or TCK. For example, the signal name in IEEE 1149.1 could be TCK, TDI, TMS, TRST, and TDO.	
dedicate_port	User defined
Set the pin name on the boundary port of a chip. This port could be IEEE 1149.1, IEEE 1149.7 signals, IEEE 1687 signals, MCK, or TCK.	
mapping_port	User defined
mapping_port is users' reserved port for MBIST and it can be connected to MBIST by the mode of replacing the port. The hierarchy must be specified and can be separated by a space bar. Figure 5-5 is the example of port connection. The following is the example of command setting:	

```
define{hookup}[TCK]
        set dedicate_port = itck
        set mapping_port = top u_pm otck
end_define{hookup}
```

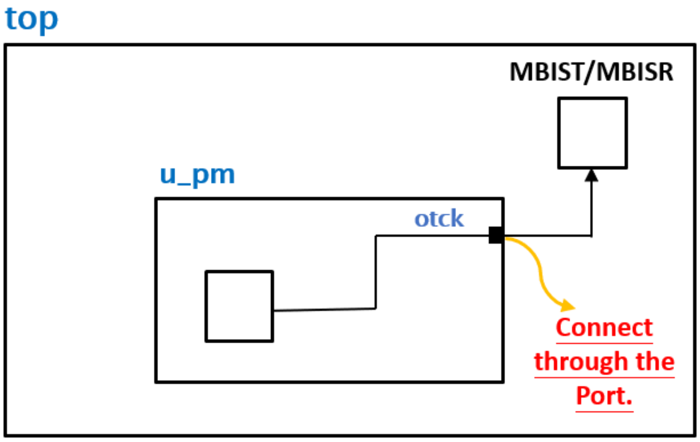


Figure 5-5 The Example of Port Connection

Argument	Option
Description	
mapping_wire	User defined

It connects to MBIST through the wire assignment. The hierarchy must be specified and can be separated by a space bar. Figure 5-6 is the example of wire connection.

The following is the example of command setting:

```
define{hookup}[TCK]
    set dedicate_port = itck
    set mapping_wire = top u_pm otck
end_define{hookup}
```

Note: Either **mapping_port** or **mapping_wire** can be chosen.

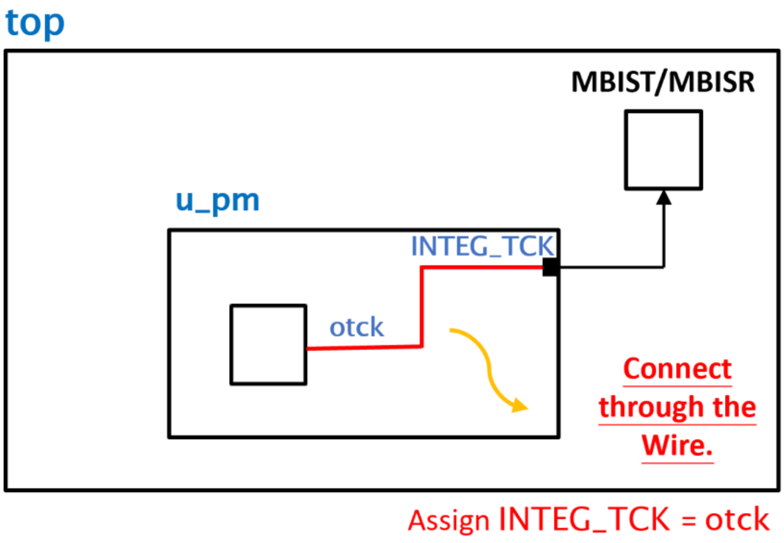


Figure 5-6 The Example of Wire Connection

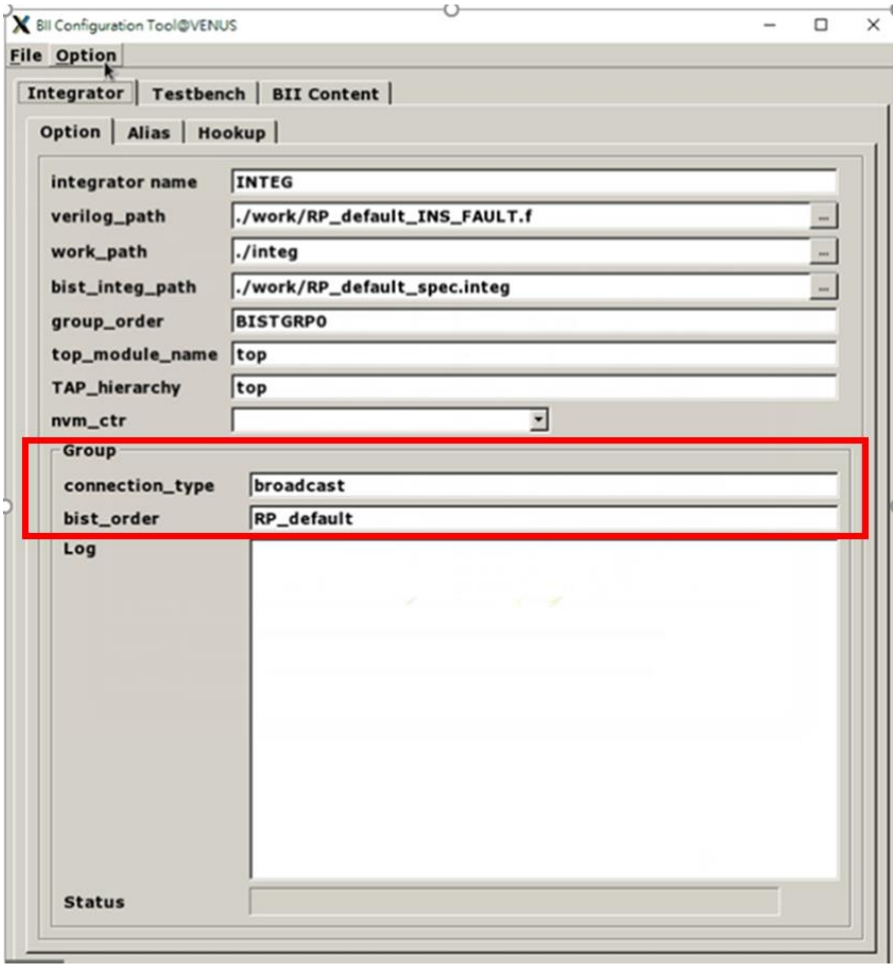
5.1.2. Group Sub Function Block

The Group sub function block defines the grouping mechanism of all MBIST controllers.

The following syntax defines the Group sub function block.

```
define{group}[group_name]
    set connection_type = …
    set bist_order = …
end_define{group}
```

Note: *[group_name]* should be the name which is listed in the column of **group_order**.



integrator name: INTEG
verilog_path: ./work/RP_default_INS_FAULT.f
work_path: ./integ
bist_integ_path: ./work/RP_default_spec.integ
group_order: BISTGRP0
top_module_name: top
TAP_hierarchy: top
nvm_ctr:
connection_type: broadcast
bist_order: RP_default

Figure 5-7 Group Sub Function Block

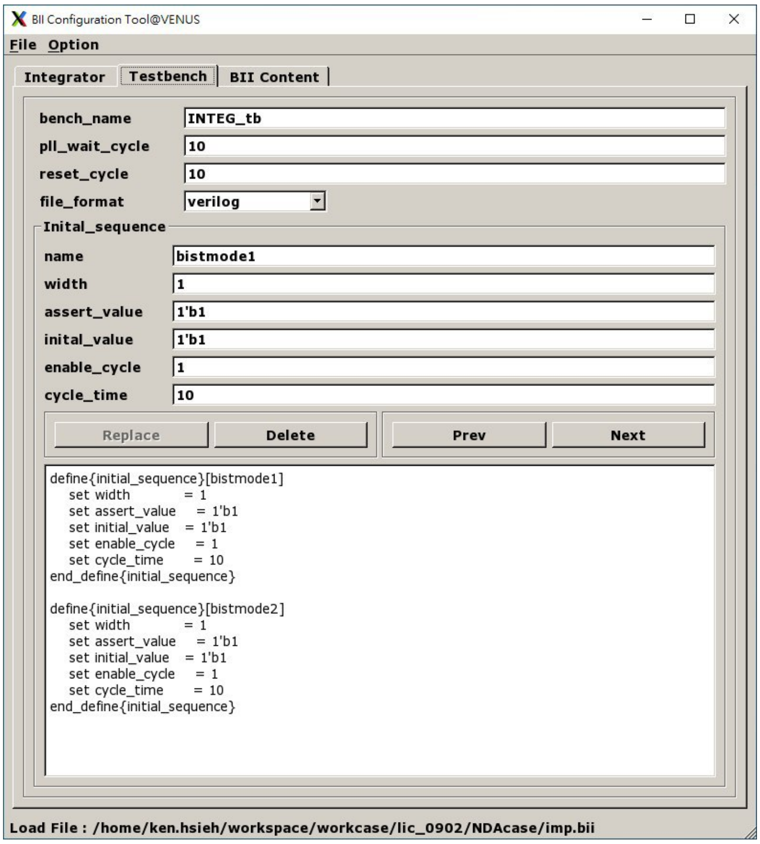
Argument	Option
Description	
bist_order	User defined
This option is for users to establish the connection order of controllers in a chain. For example, set **bist_order** to “bist1_controller, bist2_controller, bist3_controller”, and separate each MBIST controller with a comma “,”. In this case, the integrating order is bist1_controller → bist2_controller → bist3_controller.	

5.2. Testbench Function Block

The testbench block defines testbench conditions like testbench file format, pll stable cycles and reset cycles.

The following syntax defines the testbench sub function block:

```
define{Testbench}[integration_filename]
    set pll_wait_cycle = …
    set reset_cycle = …
    set file_format = …
    …sub function block…
end_define{Testbench}
```



bench_name: INTEG_tb
pll_wait_cycle: 10
reset_cycle: 10
file_format: verilog

Inital_sequence

name: bistmode1
width: 1
assert_value: 1'b1
inital_value: 1'b1
enable_cycle: 1
cycle_time: 10

```
define{initial_sequence}[bistmode1]
    set width          = 1
    set assert_value    = 1'b1
    set initial_value   = 1'b1
    set enable_cycle    = 1
    set cycle_time      = 10
end_define{initial_sequence}

define{initial_sequence}[bistmode2]
    set width          = 1
    set assert_value    = 1'b1
    set initial_value   = 1'b1
    set enable_cycle    = 1
    set cycle_time      = 10
end_define{initial_sequence}
```

Load File : /home/ken.hsieh/workspace/workcase/lic_0902/NDAcase/imp.bii

Figure 5-8 Testbench Function Block

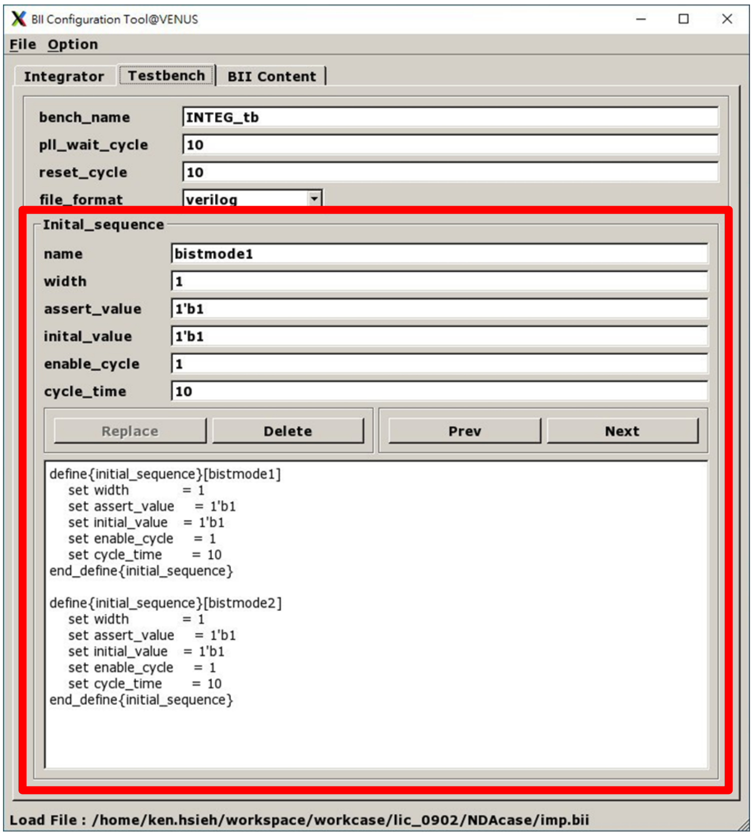
Argument	Option
Description	
bench_name	User defined
Set the test bench file name, and the default name is “INTEG_tb”.	
pll_wait_cycle	User defined
Specify the stable cycle time of PLL. The MBIST circuit will be reset after these stable cycles. Default Value: 100000	
reset_cycle	User defined
This option defines the waiting cycles to reset the MBIST circuit. While PLL is stable, the MBIST circuit will be reset after the period of **reset_cycle**.	
file_format	STIL format, WGL format, Verilog
Define the output format of testbench. The default Setting is “Verilog”.	

5.2.1. Initial_sequence Sub Function Block

The Initial_sequence sub function defines the signals on the top level which can force the system to enter testing mode. In a real chip, users may use some signals to switch function or testing mode. To run MBIST mode simulation, EZ-BIST will switch these signals to testing mode. The following syntax defines the testbench sub function block:

```
define{initial_sequence}[signal]
    set width = …
    set assert_value = …
    set initial_value = …
    set enable_cycle = …
    set cycle_time = …
end_define{initial_sequence}
```

Figure 5-9 is the example of Initial sequence from the GUI view.



bench_name	INTEG_tb
pll_wait_cycle	10
reset_cycle	10
file_format	verilog

Inital_sequence

name	bistmode1
width	1
assert_value	1'b1
inital_value	1'b1
enable_cycle	1
cycle_time	10

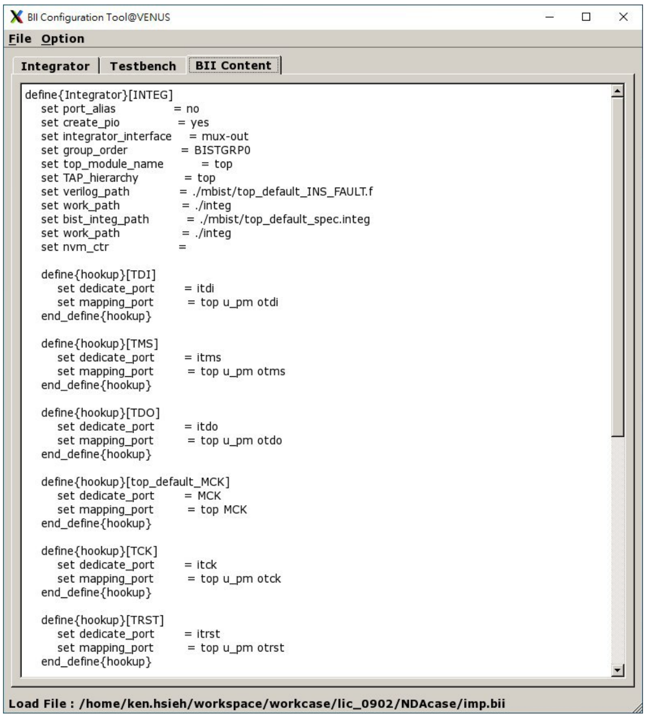
```
define{initial_sequence}[bistmode1]
    set width          = 1
    set assert_value    = 1'b1
    set initial_value   = 1'b1
    set enable_cycle    = 1
    set cycle_time      = 10
end_define{initial_sequence}

define{initial_sequence}[bistmode2]
    set width          = 1
    set assert_value    = 1'b1
    set initial_value   = 1'b1
    set enable_cycle    = 1
    set cycle_time      = 10
end_define{initial_sequence}
```

Load File : /home/ken.hsieh/workspace/workcase/lic_0902/NDAcase/imp.bii

Figure 5-9 Initial_sequence Sub Function Block

Argument	Option
Description	
width	User defined
Define the width of a signal. On the top level, users will use pins to switch function mode and testing mode.	
assert_value	User defined
Define the assert_value while entering the testing operation.	
initial_value	User defined
Define the initial value of the switch signal.	
enable_cycle	User defined
The defined signal will be changed from the initial value to the asserted value after cycle values are defined with this option.	
cycle_time	User defined
The defined signal will keep the asserted value with the cycle number which is defined in this option.	

Figure 5-10 is the example of the BII setting content from the GUI view.

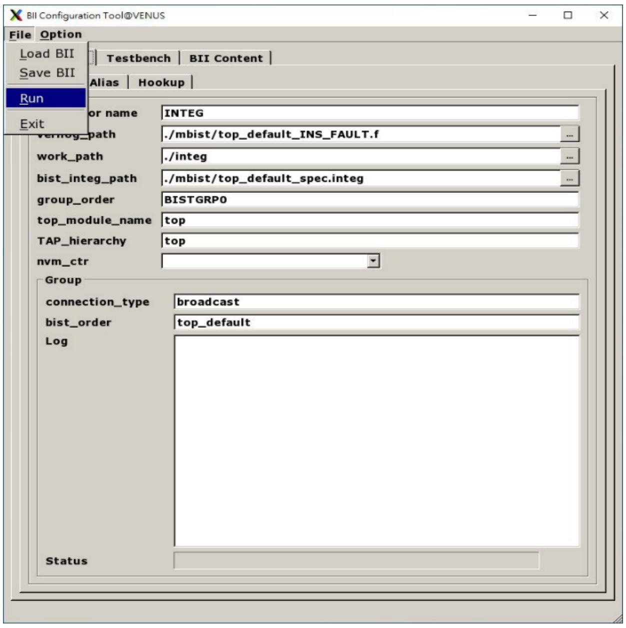


```
define{Integrator}[INTEG]
   set port_alias              = no
   set create_pio              = yes
   set integrator_interface    = mux-out
   set group_order             = BISTGRP0
   set top_module_name         = top
   set TAP_hierarchy           = top
   set verilog_path            = ./mbist/top_default_INS_FAULT.f
   set work_path               = ./integ
   set bist_integ_path         = ./mbist/top_default_spec.integ
   set work_path               = ./integ
   set nvm_ctr                 =

   define{hookup}[TDI]
      set dedicate_port      = itdi
      set mapping_port       = top u_pm otdi
   end_define{hookup}

   define{hookup}[TMS]
      set dedicate_port      = itms
      set mapping_port       = top u_pm otms
   end_define{hookup}

   define{hookup}[TDO]
      set dedicate_port      = itdo
      set mapping_port       = top u_pm otdo
   end_define{hookup}

   define{hookup}[top_default_MCK]
      set dedicate_port      = MCK
      set mapping_port       = top MCK
   end_define{hookup}

   define{hookup}[TCK]
      set dedicate_port      = itck
      set mapping_port       = top u_pm otck
   end_define{hookup}

   define{hookup}[TRST]
      set dedicate_port      = itrst
      set mapping_port       = top u_pm otrst
   end_define{hookup}
```

Load File : /home/ken.hsieh/workspace/workcase/lic_0902/NDAcase/imp.bii

Figure 5-10 Example of BII Setting Content

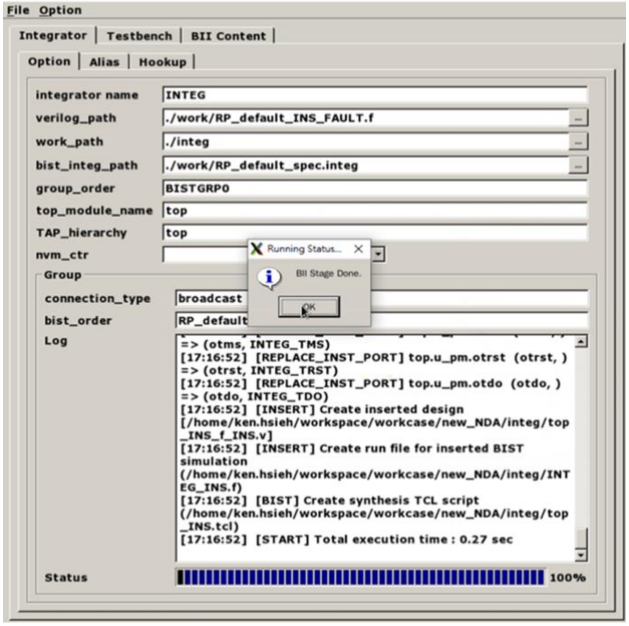
Select and click “Run” from the “File” drop-down list to execute the BII flow as Figure 5-11 shows.



or name: INTEG
path: ./mbist/top_default_INS_FAULT.f
work_path: ./integ
bist_integ_path: ./mbist/top_default_spec.integ
group_order: BISTGRP0
top_module_name: top
TAP_hierarchy: top
nvm_ctr:
Group
connection_type: broadcast
bist_order: top_default

Figure 5-11 Run BII Setting File

When the BII flow is completed, the status window will pop up to inform you the result after BFL executed as Figure 5-12 shows.



integrator name: INTEG
verilog_path: ./work/RP_default_INS_FAULT.f
work_path: ./integ
bist_integ_path: ./work/RP_default_spec.integ
group_order: BISTGRP0
top_module_name: top
TAP_hierarchy: top
nvm_ctr:
Group
connection_type: broadcast
bist_order: RP_default

BII Stage Done.

```
=> (otms, INTEG_TMS)
[17:16:52] [REPLACE_INST_PORT] top.u_pm.otrst (otrst, )
=> (otrst, INTEG_TRST)
[17:16:52] [REPLACE_INST_PORT] top.u_pm.otdo (otdo, )
=> (otdo, INTEG_TDO)
[17:16:52] [INSERT] Create inserted design
[/home/ken.hsieh/workspace/workcase/new_NDA/integ/top
_INS_f_INS.v]
[17:16:52] [INSERT] Create run file for inserted BIST
simulation
(/home/ken.hsieh/workspace/workcase/new_NDA/integ/INT
EG_INS.f)
[17:16:52] [BIST] Create synthesis TCL script
(/home/ken.hsieh/workspace/workcase/new_NDA/integ/top
_INS.tcl)
[17:16:52] [START] Total execution time : 0.27 sec
```

Figure 5-12 The Status Window When BII Flow is Completed

6. Appendixes

6.1. “Include” Case

For those designs, which contain a relative path with “include” and will be modified, EZ-BIST will rewrite the relative path to absolute path. Therefore, if user plan to copy the design to another path, please manually edit the absolute path based on new path or re-execute EZ-BIST to generate the correct path.

6.2. Parsing Mode

If the design is RTL, please make sure it could be synthesized. Otherwise, EZ-BIST cannot parse the design for inserting MBIST circuit to the design.

Due to the diverse syntax of RTL, we suggest users using netlist as an input if RTL keeps having parsing issue.

6.3. *.rcf File

To avoid simulation failure, please use the absolute path in `rom.v` if you try to open a `*.rcf` file.

6.4. Supported Testing Algorithm

Table 6-1 Testing Algorithms for SRAM in EZ-BIST

Memory Type	Name	Fault Detection	Algorithm
SRAM	March CW (part 1)	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF	>(wa) >(ra,wb) >(rb,wa,ra) <(ra,wb,rb) <(rb,wa) <(ra)
	March CW (part 2)	Word-oriented CF	>(wa) >(wb) >(rb,wa,ra)
	March Y	SAF, TF, CFin, SOF, RDF	>(wa) >(ra,wb,rb) <(rb,wa,ra) <(ra)
	March X	SAF, TF, AF, CFin	>(wa) >(ra,wb) <(rb,wa) <(ra)
	MATS++	SAF, TF, AF, SOF	>(wa) >(ra,wb) <(rb,wa,ra)
	MOVI	SAF, TF, AF, CFin, CFst, SOF, RDF	<(wa) >(ra,wb,rb) >(rb,wa,ra) <(ra,wb,rb) <(rb,wa,ra)
	Ext March C-	SAF, TF, AF, CFin, CFid, CFst, SOF	>(wa) >(ra,wb) >(rb,wa,ra) <(ra,wb) <(rb,wa) <(ra)
	*March C+	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF	>(wa) >(ra,wb,rb) >(rb,wa,ra) <(ra,wb,rb) <(rb,wa,ra) <(ra)
	March C-	SAF, TF, AF, CFin, CFid, CFst	>(wa) >(ra,wb) >(rb,wa) <(ra,wb) <(rb,wa) <(ra)
	March C Gray	ADOF	>(wa) >(ra,wb) >(rb,wa) <(ra,wb) <(rb,wa) <(ra) Address only one bit change
	March LR	SAF, TF, AF, CFin, CFid, CFst, SOF	>(wa) >(ra,wb) >(rb,wa,ra,wb) >(rb,wa) >(ra,wb,rb,wa) >(ra)
	March C	SAF, TF, AF, CFin, CFid, CFst	>(wa) >(ra,wb) >(rb,wa) >(ra) <(ra,wb) <(rb,wa) <(ra)
	March B	SAF, TF, AF, CFin, CFid, SOF	>(wa) >(ra,wb,rb,wa,ra,wb) >(rb,wa,wb) <(rb,wa,wb,wa) <(ra,wb,wa)
	March A	SAF, TF, AF, CFin, CFid	>(wa) >(ra,wb,wa,wb) >(rb,wa,wb) <(rb,wa,wb,wa) <(ra,wb,wa)
	March 17N	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF	>(wb) >(rb,wa,ra) >(ra,wb,rb) >(rb,wa) <(ra,wb,rb) >(rb) <(rb,wa,ra) >(ra)

	March 19N	'SAF', 'TF', 'AF', 'CFin', 'CFid', 'CFst', 'SOF', 'RDF'	>(wa,ra) >(wa) >(ra,wb,rb) >(rb) >(rb,wa,ra) >(ra) <(ra,wb,rb) >(rb) <(rb,wa,ra) >(ra)
	March 33N	dRDF, dIRF, dDRDF, dTF, dWDF	>(wa) >(wa,wb,wa,wb) >(rb,wa,wa) >(wa,wa) >(ra,wb,rb,wb,rb,rb) <(rb) <(wb, wa,wb,wa) <(ra,wb,wb) <(wb,wb) <(rb,wa,ra,wa,ra,ra) <(ra)
	March 33N-	'dRDF', 'dIRF','dDRDF', 'dTF', 'dWDF'	'>(wa) >(wa,wb,wa,wb) >(r-1b,wa,wa) >(wa,wa) >(r-1a,wb,r-1b,wb,r-1b,r-1b) <(r-1b) <(wb,wa,wb,wa) <(r-1a,wb,wb) <(wb,wb) <(r-1b,wa,r-1a,wa,r-1a,r-1a) <(r-1a)'
	March M	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF	>(wa) >(ra,wb,rb,wa) >(ra) >(ra,wb) >(rb) >(rb,wa,ra,wb) >(rb) <(rb,wa)
	March Mdsn1	SAF, TF, AF, CFin, CFid, CFst RET	Part1~Part4
	March Mdsn1 (part1)	SAF, TF, AF, CFin, CFid, CFst	>(wa) >(wb,wa) (SLP) >(ra,wb,wb)
	March Mdsn1 (part2)	SAF, TF, AF, CFin, CFid, CFst	>(rb,wa,ra,wa,ra,wb) >(rb,rb)
	March Mdsn1 (part3)	SAF, TF, AF, CFin, CFid, CFst	<(wa,wb) (SLP) <(rb,wa,wa)
	March Mdsn1 (part4)	SAF, TF, AF, CFin, CFid, CFst	<(ra,wb,rb,wb,rb,wa) <(ra,ra)
	March SSSc	SAF, TF, AF, CFin, CFid, CFst	>(wa) >(wb,wb,rb,rb,wa) >(wb) >(wb,wb,rb,rb,wa)
	Non-March BM	detect bit/group write enable faults and datapath shorts.	>(wa) >(wB5b,rB5b) <(wBAb,rBFb) >(wBAa,rBAa) <(wB5a,rBFa)
	MARCH_RET	RET	<(wb) (SLP) <(rb) >(wa) (SLP) >(ra)
	CB	BF	>(wa) >(ra) >(wb) >(rb)
	March 8R	dRDF	>(wa,ra,ra,ra,ra,ra,ra,ra,ra) >(wb,rb,rb,rb,rb,rb,rb,rb,rb)
	March 5W	SAF, TF, CFst, dWDF, WDF	>(wa) >(ra,wb,rb,wb,wb,wb,wb,wb)

			>(rb,wa,wa,wa,wa,wa) <(ra,wb,wb,wb,wb,wb) <(rb,wa,wa,wa,wa,wa)
	March RP	WDF	>(wa) >(ra,wb) >(rb,wa,r-1a) <(ra,wb,r-1b) <(rb,wa) >(ra)
	**March d2PF	'SAF', 'TF', 'AF', 'CFin', 'CFid', 'CFst', 'SOF', 'RDF', 'Weak WL', '2PFavS'	'>(n|wa) >(r+1a|n,n|wb) >(r+1b|n,n|wb) >(r+1b|n,n|wa) >(r+1a|n,n|wa) >(r+1a|n,n|wb) >(r+1b|n,n|wb) >(r+1b|n,n|wa) >(r+1a|n,n|wa)'
	**March s2PF	'SAF', 'TF', 'AF', 'CFin', 'CFid', 'CFst', 'SOF', 'RDF', 'Weak WL', '2PF1s', '2PF1as'	'>(n|wa) >(ra|n,ra|n, n|wb) >(rb|n, rb|n, n|wa) <(ra|n, ra|n, n|wb) <(rb|n, rb|n, n|wa) <(ra|n)'
	***March A2PF-M	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF, Weak WL, A2PF	>(wa|n) >(ra|ra,wb|r+1a,wb|r-1b,rb|rb) >(rb|rb,wa|r+1b,wa|r-1a,ra|ra) <(ra|ra,wb|r-1a,wb|r+1b,rb|rb) <(rb|rb,wa|r-1b,wa|r+1a,ra|ra) <(ra|n)
	March CROWN	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF, dDRDF, dWDF	>(wb) >(rb) >(wa,ra) >(ra) <(wb,wa,ra) <(wb,rb,rb)
	March 17N POLARIS	SAF, TF, AF, CFin, CFid, CFst, SOF, RDF, dDRDF, dWDF	>(wb) >(rb) >(wa,ra) >(ra) <(wb,wb,rb) <(wa,wb,rb) <(wb,rb,rb) <(wa,ra,ra)

***: Default testing algorithm of EZ-BIST**

**** : Support two port memory only**

***** : Support dual port memory only**

±1: used to increase/ decrease memory address

|: used to separate operation of different port

>: indicates address count from o to the highest address in a memory.

<: indicates address count from the highest address to 0 in a memory.

a: indicates test pattern.

b: indicates inverse “a” test pattern.

MISR (Multiple-Input Signature Register)

LFSR (linear feedback shift register)

Table 6-2 Testing Algorithms for ROM in EZ-BIST

Memory Type	Name	Address sequence	Operation	Description
ROM	*ROM Test	LFSR	(rc)	Reads and compresses ROM's content
		N/A	Compare MISR	Compares the final signature

Table 6-3 Testing Algorithms for ROM (ROM Test 3n) in START

Memory Type	Name	Algorithm	Description
ROM	ROM Test 3n	>(rc) <(rc) >(rc)	Reads, compresses the ROM's content, and compares the final signature.

6.5. Statistics in TSMC SP Memory

Design Architecture:

- ✓ **Memory**: Single-port SRAM *20 and ROM *1
- ✓ **Process**: TSMC 55nm
- ✓ **Library**: sc9_cln55lp_base_rvt_ss_typical_max_1p08v_125c
- ✓ **NAND Gate area**: 1.44 um^2

I. The default setting of BFL file: `default.bfl`

Table 6-4 The Default Setting of BFL file

BFL File Column	Default Value
clock_trace	no
STIL_test_bench	no
asynchronous_reset	yes
bist_interface	basic
address_fast_y	no
algorithm_selection	no
background_style	SOLID
background_bit_inverse	no
background_col_inverse	no
bypass_support	no
bypass_clock	no

BFL File Column	Default Value
bypass_reg_sharing	1
clock_function_hookup	no
clock_switch_of_memory	yes
clock_source_switch	no
clock_within_pll	no
diagnosis_support	no
diagnosis_data_sharing	no
diagnosis_memory_info	no
diagnosis_time_info	no
diagnosis_faulty_items	all
parallel_on	no
reduce_address_simulation	no
rom_result_shiftout	no
Q_pipeline	no
algorithm	March C+
meminfo	1 Ctr 2 Seq 2 Group

Table 6-5 Synthetic Area of default.bfl

Referenced Library	Total Area
top_default_controller	798.120025
top_default_sequencer1	528.84001
top_default_sequencer2	258.480007
top_default_ter_1_1_1	402.840007
top_default_ter_1_1_2	402.840007
top_default_ter_1_1_3	402.840007
top_default_ter_1_1_4	402.840007
top_default_ter_1_1_5	402.840007
top_default_ter_1_1_6	402.840007
top_default_ter_1_1_7	402.840007
top_default_ter_1_1_8	402.840007
top_default_ter_1_1_9	402.840007
top_default_ter_1_1_10	402.840007
top_default_ter_1_1_11	402.840007
top_default_ter_1_1_12	402.840007
top_default_ter_1_1_13	402.840007
top_default_ter_1_1_14	402.840007

Referenced Library	Total Area
top_default_ter_1_1_15	402.840007
top_default_ter_1_1_16	402.840007
top_default_ter_1_1_17	402.840007
top_default_ter_1_1_18	402.840007
top_default_ter_1_1_19	402.840007
top_default_ter_1_1_20	402.840007
top_default_ter_2_1_1	164.160006
top_default_tpg_1_1_1	334.8
top_default_tpg_1_1_2	336.24
top_default_tpg_1_1_3	336.24
top_default_tpg_1_1_4	334.8
top_default_tpg_1_1_5	333.36
top_default_tpg_1_1_6	334.8
top_default_tpg_1_1_7	333.36
top_default_tpg_1_1_8	334.8
top_default_tpg_1_1_9	331.92
top_default_tpg_1_1_10	333.36
top_default_tpg_1_1_11	334.8
top_default_tpg_1_1_12	334.8
top_default_tpg_1_1_13	334.8
top_default_tpg_1_1_14	334.8
top_default_tpg_1_1_15	333.36
top_default_tpg_1_1_16	333.36
top_default_tpg_1_1_17	333.36
top_default_tpg_1_1_18	331.92
top_default_tpg_1_1_19	331.92
top_default_tpg_1_1_20	331.92
top_default_tpg_2_1_1	606.960003
Total 145 references	**17092.08019**

(Unit:um2)

II. Refer to Circuit Area Comparison table to change each option in default.bfl file.

For example, set the option **asynchronous_reset** to “no”, the circuit area will become 99.085% of the original circuit area, which means the circuit area will decrease by about 0.91%.

Table 6-6 Area Comparison Table

Process/Lib.: TSMC 55nm/ sc9_cln55lp_base_rvt_ss_typical_max_1p08v_125c +: Increase, -: Decrease	**Default .bfl**
asynchronous_reset = no	-0.91%
address_fast_y = yes	3.08%
clock_within_pll = yes	0.11%
parallel_on =yes	0.96%
reduce_address_simulation = yes	3.10%
rom_result_shiftout = yes	7.05%
Q_pipeline = yes	67.60%

bist_interface = ieee1500	1.03%
bist_interface = ieee1149.1	2.25%

algorithm add March C-	0.44%
algorithm add March C- algorithm_selection = outside	0.61%
algorithm add March C- algorithm_selection = scan	0.61%

Note: If the option **algorithm_selection** set to “outside” or “scan”, the circuit area will increase by 0.17%.

Process/Lib.: TSMC 55nm/ sc9_cln55lp_base_rvt_ss_typical_max_1p08v_125c +: Increase, -: Decrease	**Default .bfl**
background_style = 5A	0.93%
background_bit_inverse = yes	2.07%
background_col_inverse = yes	0.67%

bypass_support = wire	14.42%
bypass_support = reg	82.81%
bypass_support = reg bypass_clock = yes	82.81%
bypass_support = reg bypass_clock = yes bypass_reg_sharing = 2	60.38%
bypass_support = reg bypass_clock = yes bypass_reg_sharing = 4	45.74%

Note: If the option **bypass_support** is set to “reg”, the circuit area will increase by 82.81%. If the option **bypass_clock** is set to “yes”, the circuit area will increase by 82.81%. However, if **bypass_reg_sharing** is set to “2”, the circuit area will only increase by 60.38%. The option **bypass_reg_sharing** can effectively reduce the circuit area.

6.6. RTL Syntax Restrictions

I. For a module instance, empty port information is not allowed.
Example:

```
module UART (D, Q, CK);
        input D, CK;
        ...
endmodule
```

The following syntax is not supported:

```
UART u_uart();
```

Instead, the following syntax is supported:

```
UART u_uart(.CK());
```

II. A module with no content inside is not supported. A module must have at least one line of RTL code inside.
Example:

```
module wrapper (input ck);
endmodule
```