START User Manual

v5.1

Contents

1. START Architecture 2
2. START Command Options and Parameters 4
2.1. Input Verilog Files 5
2.2. Specify the Working Path 6
2.3. Auto-Identify the Memory Model 7
2.4. Generate the ROM Signature 8
2.5. Template File Generator 8
2.6. Set BFL File in the GUI Mode 9
2.7. Input BFL File 10
2.8. Insert MBIST/MBISR to Design 10
2.9. Specify Top Module 10
2.10. Input Integration SPEC File 11
2.11. Input UDM File 11
2.12. Integrate Multiple MBIST/MBISR Circuits 11
2.13. Generate UDM File with Library File 12
2.14. Generate UDM File with Configuration File 12
2.15. Parsing Type Definition 13
2.16. Fault Free 13
2.17. RCF Generator 14
2.18. STIL Format 14
3. START BFL Options 15
3.1. Clock Sub Function Block 16
3.2. Option Function Block 16
3.2.1. Group Sub Function Block 21
3.2.2. PHYSICAL Sub Function Block 25
3.3. MBIST Function Block 26
3.3.1. Default Algorithm Sub Function Block 49
3.3.2. Programmable Algorithm Sub Function Block 49
4. START Output Files 52
4.1. Self-MBIST Related Files 52
4.2. Self-MBISR Related Files 53
4.3. Insert MBIST Related Files 54
4.4. Insert MBISR Related Files 55
4.5. “Generate for Loop” Related Files 56
4.6. Generate Folders 57
4.7. Makefile 58
4.8. Macro File 60

5. START BII Files 62
5.1. Integrator Function Block 62
5.1.1. Hookup Sub Function Block 65
5.1.2. Hookup Port Type 68
5.1.3. Group Sub Function Block 70
5.2. Testbench Function Block 71
5.2.1. Initial_sequence Sub Function Block 72
6. Appendixes 73
6.1. “Include” Case 73
6.2. Parsing Mode 73
6.3. *.rcf File 73
6.4. Supported Testing Algorithm 74
6.5. Statistics in TSMC SP Memory 78
6.6. RTL Syntax Restrictions 83

List of Figures

Figure 1-1 START Operation Flow Diagram 2
Figure 2-1 START Command Options 4
Figure 2-2 File-List File Example 5
Figure 2-3 Memchecker Information 7
Figure 2-4 Example of *_gold_signature.txt 8
Figure 2-5 START Template Generator 8
Figure 2-6 Perspective of BFL Configuration Tool 9
Figure 2-7 UDM Configuration File Example 12
Figure 3-1 Block Diagram of System Design with MBISR Inserted 18
Figure 3-2 Block Diagram of System Design with MBIST Inserted 18
Figure 3-3 Memory Info Setting Information 24
Figure 3-4 Scheme of the async_reg circuit 28
Figure 3-5 Scheme of the GCK_MUX circuit 28
Figure 3-6 Scheme of the glitch_free circuit 29
Figure 3-7 The Waveform of the hard_repair_with_buffer Option 31
Figure 3-7 Example of Synchronous/Asynchronous Circuit 32
Figure 3-8 Example of ATPG Circuit 32
Figure 3-9 Commands for Programmable Algorithm Function 33
Figure 3-10 Example of Retention Time Option in testbech.v 39
Figure 3-11 Implementation of Bypass Circuit through Wire 40
Figure 3-12 Implementation of Bypass Circuit through Register 40
Figure 3-13 Implementation of CS & Clock Circuits (Option = No) 41
Figure 3-14 Implementation of CS & Clock Circuits (Option = Yes) 41
Figure 3-15 Example of Register Sharing 42
Figure 3-16 Clock Architecture of clock_function_hookup Option 43
Figure 3-17 Clock Architecture of clock_switch_of_memory Option 44
Figure 3-18 Diagnosis Failed Memory Information 44
Figure 3-20 The RPCT Option in the BFL Setting 48
Figure 3-21 NVM Sharing 48
Figure 3-22 Default Algorithm Function Block 49
Figure 3-23 ACTION_LOOP & ALG_CMD in testbench 50
Figure 4-1 Clock Gating Logic for Simulation and Synthesis 61
Figure 4-2 Clock Gating Cell with Waveform 61
Figure 5-1 The NRLT Waveform 65
Figure 5-2 The Example of Port Connection 66
Figure 5-3 The Example of Wire Connection 67

List of Tables

Table 1-1 START Input Files 2
Table 1-2 START Output Files 2
Table 3-1 Clock Information 16
Table 3-2 Commands for Programmable Algorithm 33
Table 3-3 BG Field Definition 35
Table 3-4 Example of Bit Inverse 35
Table 3-5 Example of Column Inverse 36
Table 3-6 Example of User-defined Background and Test Pattern 37
Table 3-7 Supported Units of Retention Time 38
Table 3-8 Fixed Four Memory Address 45
Table 3-9 Fixed Two Memory Address 45
Table 3-10 Exampled Variables of Memory Data Width 46
Table 3-11 Testing Status 47
Table 3-13 Format of March CW Element 50
Table 4-1 Self-MBIST Related Files 52
Table 4-2 Self-MBISR Related Files 53
Table 4-3 Insert MBIST Related Files 54
Table 4-4 Insert MBISR Related Files 55
Table 4-5 “Generate for Loop” Related Files 56
Table 4-6 Generated Folder 57
Table 4-7 Commands of Makefile 58
Table 5-1 Descriptions of Hookup Port Type 68
Table 6-1 Testing Algorithms for SRAM in START 74
Table 6-2 Testing Algorithms for ROM (ROM Test) in START 77
Table 6-3 Testing Algorithms for ROM (ROM Test 3n) in START 77
Table 6-4 The Default Settings of the BFL File 78
Table 6-5 Synthetic Area of default.bfl 79
Table 6-6 Area Comparison Table 81

Type conversion in this document

Conversion	Meaning for use
Bold	Items in the user interface that being selected or clicked. Text that is being typed 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 the START tool presents 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.

November 2025

1. START Architecture

START is a tool that can generate the test circuit for MBIST (Memory Built-In Self-Test) and MBISR (Memory Built-In Self Repair), providing total solutions including comprehensive test algorithms, auto-grouping mechanism, and auto-integration mechanism for MBIST/MBISR circuits and the original circuit. It is easy for users to generate optimized MBIST/MBISR circuits. Figure 1-1 shows the operation flow of START.

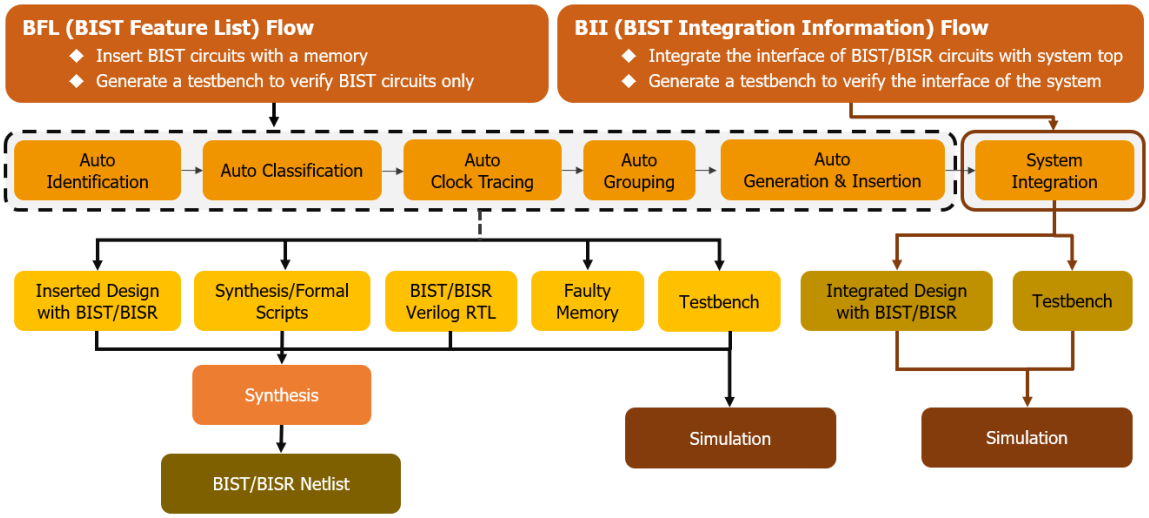


Figure 1-1 START Operation Flow Diagram

START input files include the files listed below:

Table 1-1 START Input Files

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

START output files include the files listed below:

Table 1-2 START Output Files

Inserted Design	Integrated MBIST circuits into 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 pre-defined error bit memory.
Testbench	Testbench of MBIST circuits simulation

2. START Command Options and Parameters

Users can execute the START commands with the options, *--help* or *-h*, to know all options supported by START. Figure 2-1 shows an example when executing START 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: start [-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 verilog file(s)
  -W DIR                              specify working path
  -top MODULE, -T MODULE              specify top module
  -I, --insert                        insert MBIST to design
(……)
```

Figure 2-1 START Command Options

2.1. Input Verilog Files

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”. START provides an auto-insertion function to integrate MBIST/MBISR circuits into the original system design. For this reason, users need to provide the whole design files rather than the memory files only.

It 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 START commands. START will read design files automatically. The file-list file also supports *+define+, +incdir+* and *-y* options.

Example 1: *$ start -v vlog_1*

START will read Verilog files in vlog_1 directory.

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

START will read `file1.v` in vlog_1 directory and `file4.v` in vlog_2 directory.

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

START will read designs in `flielist.f`. Figure 2-2 is an example of a 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 2-2 File-List File Example

2.2. Specify the Working Path

Usage: *-W [WORK_PATH]*

Description: This option is used to set the output directory of the START execution results.

Example 1: *$ start -v [VLOG_PATH]/[file_1].v -W [WORK_PATH]*
START will read the `file_1.v` design file and save the output results into the WORK_PATH directory.

Example 2: *$ start -v [VLOG_PATH]/[file_1].v*
Without the *-W* option, START will save all generated results into the current working directory.

2.3. Auto-Identify the Memory Model

Usage: *--memchecker*

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

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

Users can check if there is a memory model which cannot be identified by reviewing the output messages as Figure 2-3.

```
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 (…)
  [8] /home//workspace/project/memchecker/memory/RA1RW_D2048 (…)
  [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_1024 (…)
  [13] /home//workspace/project/memchecker/memory/RA1RW_D1024 (…)
  [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 (…)
  [8] /home /workspace/project/memchecker/memory/RA1RW_D2048 (…)
  [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_1024 (…)
  [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 2-3 Memchecker Information

2.4. Generate the ROM Signature

Usage: *--memchecker*

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

Example: *$ start --memchecker -v [ROM memory RTL code file]*
$ start --memchecker -v rom_6144_64.v

Note: The value of signature will be saved in the `*_gold_signature.txt` (See Figure 2-4) and in the meantime, a `top.v` file will be generated and replace the first one in the memory folder.

```
Rom_6144_64_verilog gold_signature = 7be4eb
```

Figure 2-4 Example of *_gold_signature.txt

2.5. Template File Generator

Usage: *--tempgen*

Description: This option is used to generate a template file of START. These template files include BII (MBIST/MBISR Integration Information) files, BFL (MBIST/MBISR Feature List) files and UDM file as Figure 2-5.

Example: *$ start --tempgen*

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

Figure 2-5 START Template Generator

2.6. Set BFL File in the GUI Mode

Usage: *--bflconfig BFL_FILE*

Description: This option is used to invoke a BFL file with the GUI mode. If BFL_FILE is not defined in commands, the initial name is `start_template`.

Example: *$ start --bflconfig [filename]*

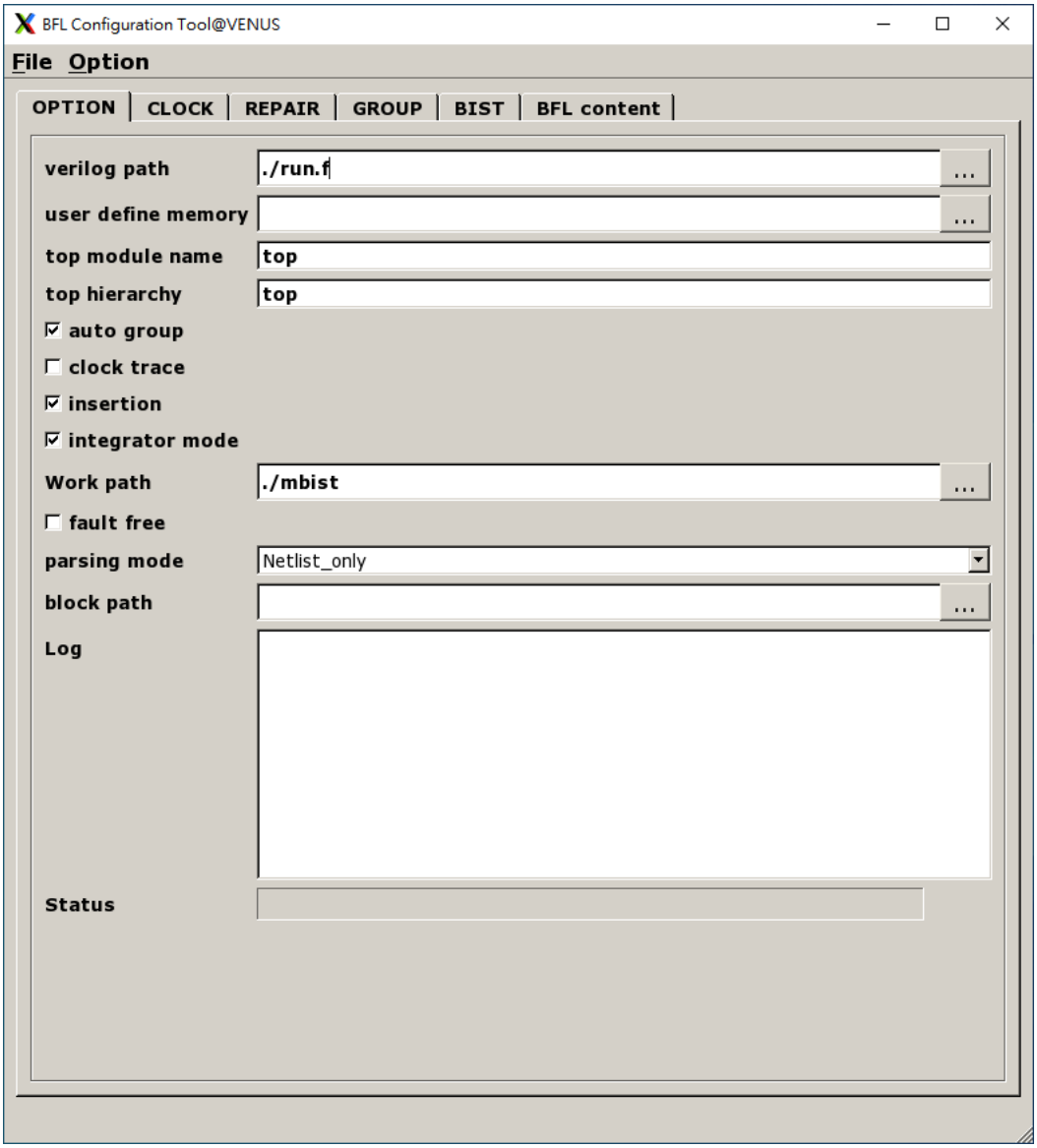


Figure 2-6 Perspective of BFL Configuration Tool

2.7. Input BFL File

Usage: *-bfl BFL_FILE*

Description: This option is used to define the BFL file for START.

Example: *$ start -bfl [filename].bfl -W [WORK_PATH]*

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

2.8. Insert MBIST/MBISR to Design

Usage: *-I, --insert*

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

Example: *$ start -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 generated MBIST/MBISR circuits into the original system designs. Users need to define a top module name with the *-top* option when using this option.

Example: *$ start -I -top [TOP_MODULE] -v [VLOG_PATH]/[file_1].v*

2.10. Input Integration SPEC File

Usage: *-integ*

Description: This option is used to import the integration spec file for integrator mode.

Example: *$ start -integ test_spec.integ*

2.11. Input UDM File

Usage: *-u UDM_FILE*

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

Example: *$ start -bfl [filename].bfl -u *.udm -W [WORK_PATH]*
START will read the BFL file and UDM files in the working directory. The output results will be saved into WORK_PATH.

2.12. Integrate Multiple MBIST/MBISR Circuits

Usage: *--integrator*

Description: This option is used to integrate multiple MBIST/MBISR circuits to save area cost.

Example: *$ start --integrator -bii [filename].bii -W [WORK_PATH]*
START will refer to BII file to integrate multiple MBIST/MBISR circuits and save output results into WORK_PATH.

2.13. 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 the 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: *$ start –memlib2udm -lv sram_512x8.memlib*
START will generate a UDM file for memory sram_512x8.

2.14. Generate UDM File with Configuration File

Usage: *--ug UDM_File config_file*

Description: This option is used to generate a UDM file based on the settings in the configuration file. Figure 2-7 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 the data width, and the fourth column defines the mux.

Example: *$ start --ug sram_512x8.udm config.file*
START 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 2-7 UDM Configuration File Example

2.15. 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: *$ start -pm Netlist_only -v example.v*
START will import `example.v` with nestlist format.

2.16. Fault Free

Usage: *--faultfree*

Description: This option is used to decide if the generated system designs are with or without fault memories. 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: *$ start -bfl start_template.bfl -I -W ./work*
START will generate an integrated system design with fault memory models.

Example 2: *$ start -bfl start_template.bfl -I --faultfree -W ./work*
START will generate integrated system designs with and without fault memory models, respectively.

2.17. 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 the output RCF file is random.

Example: *$ start --rcfg 32 8 example.rcf*
START will generate an example RCF file with the 32x8 matrix format.

2.18. STIL Format

Usage: *--STILloopformat*

Description: Change STIL file into the loop format.

Example: *$ start --STILloopformat*
START will generate a STIL file into the loop format.
If there are many repetitive testing commands, using the option will simplify the testing commands as the loop instruction.

3. START BFL Options

Users utilize START to generate MBIST/MBISR circuits and the MBIST/MBISR design is based on the BFL file settings. The BFL flow is recommended for users who use START in the first time. Users can execute START with the BFL flow to set the related options in the `*.bfl` file. This chapter will introduce the setting options in the BFL file.

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

define{function}

…

end_define{function}

User can find different options in each function block below.

3.1. Clock Sub Function Block

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

Table 3-1 Clock Information

Clock Information	Definition
sdc_file	Specify the path of the SDC file.
define{clock_name}	Set the clock domain name.
clock_cycle	Set operating period of the clock domain defined in “clock_name”.
clock_source_list	Set the source pin or the port of the clock domain defined in “clock_name”.

3.2. Option Function Block

Argument	Option
Description	
verilog_path	User defined
Set the Verilog file paths for START. The format can be set as *file_list.f (`*.f`)* here. Example: *set verilog_path = ./top.f*	
user_define_memory	User defined
Set UDM file paths for START. 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 = START.udm*	

Argument	Option
Description	
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 the MBIST/MBISR circuits in the design architecture. Example: *set top_hierarchy = top*	
clock_trace	No, Yes
This option is used for users to disable/enable the clock source tracing function. The default setting is “no”. No: Disable clock source tracing function. Yes: Enable clock source tracing function.	
auto_group	No, Yes
This option is for users to automatically group memory models based on the settings in GROUP function block. The default setting is “no”. No: Disable the auto-grouping function. Users should specify the `*.meminfo` file to memory_list. Yes: Enable the auto-grouping function.	
insertion	No, Yes
This option is used to integrate the generated MBIST/MBISR circuits and the original system designs. Figure 3-1 shows the block diagram of the inserted system design. No: Disable the insertion function. Yes: Enable the insertion function.	

Argument	Option
Description	

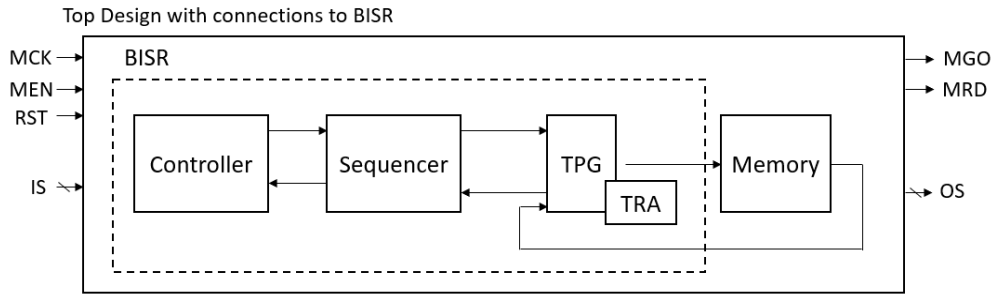


Figure 3-1 Block Diagram of System Design with MBISR Inserted

integrator_mode	No, Yes

This option is used for user to add the dedicated testing port in the top module of MBIST. As Figure 3-1 and Figure 3-2 show, when this option is set to “yes”, START will reserve signals internally in advance for testing only in the BFL flow. Because these testing ports are standard, like IEEE 1149.1, users can use the shared pin design to reduce the pin count. The default setting is “no”.

Note: The option must be set to “yes” when the clock tracing is turned on.

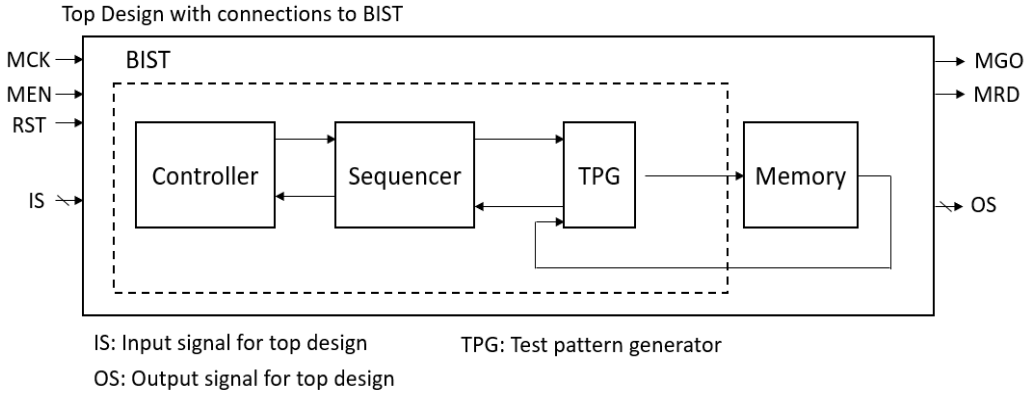


Figure 3-2 Block Diagram of System Design with MBIST Inserted

As the two figures above shows, the controller is responsible for controlling the test flow and communicating with ATE (Automatic Test Equipment). The sequencer is responsible for collecting all the test results from TPG and reporting the test results to the controller. The TPG (Test Pattern Generator) is responsible for generating a test vector to write patterns into memories, reading data from memories, and comparing with the expected data to generate the test results. Besides, if the repair mode is enabled, and the memory

Argument	Option
Description	
needs repair, it will generate TRA (Testing Redundancy Analyzer) to activate the redundant memory.	
work_path	User defined
Specify the path for saving the generated results in BFL flow.	
fault_free	No, Yes
When this option is set to “no”, START will generate an integrated system design with fault memory models. On the contrary, when this option is set to “yes”, START 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 is not uniquified, the parsing mode must be setting to “RLT_only”. Example: *set parsing_mode = RTL_only*	
repair_prefix	User defined
This option is to be filled in the specified prefix name of repair-related files. For example, when this option is set to “RP”, the output repair-related files will be named like `RP_[design]_INS.v` and `RP_[filename]_tb.v` etc. Example: *set repair_perfix = RP*	
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.	
block_path	User defined
While the design is implemented with 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*	

Argument	Option
Description	
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*	
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.	
rck_clock_cycle	User defined
This function is for users to modify the rck_clock_cycle of eFuse and OTP (One-time Programmable Memory). START will automatically calculate the corresponding timing and generate the corresponding circuit based on the rck_clock_cycle set by users in BFL. For example, the TSMC eFuse is supported, and the default value is 37ns.	
disable_auto_identify	No, Yes
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.	

3.2.1. Group Sub Function Block

START 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 with 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 clock domains of memory models. This ID indicates to which sequencer the memory models belong. The GRP_ID is classified by power consumption and number limitations of a single group.

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 to paths of memory info file (`*.meminfo`) if manual grouping is required. Figure 3-3 is an example of a memory info file. For more details, please refer to Application Notes.	
time_hierarchy	User defined
This option is for users to adjust the weight between the testing time and design hierarchy. The default value is 0.5. Example: *set time_hierarchy = 0* — START will assign memory grouping based on the optimized testing time. The testing time will be the highest priority. *set time_hierarchy = 1* — START will assign memory grouping by hierarchy relationships. The logical hierarchy will be the highest priority.	
lib_path	User defined
This option is for user to set the path of memory libraries. START will load power information of memory models from `*.lib` files and do memory grouping automatically based on the power criteria number by the **power_limit** option. Example: *set lib_path = ./lib* *set lib_path = ./lib.f*	
power_limit	User defined
Set the maximum limitation of power consumption in one group. The **power_limit** unit is mW and the value representation could be decimal and hexadecimal. Example:	

Argument	Option
Description	
set power_limit = 0.005	
hierarchy_limit	User defined
Set the maximum hierarchy number (no limitation) when doing auto-grouping. If the hierarchy number between memory models is larger than this number, START will not group these memory models into the same group. Default Value: 0 (no limitation of hierarchy number)	

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

- **Clock domain**: It shows the 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.

```
# if the instance is the alais type, the ** is .
[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=no, diag=no, q_pipe=yes, repair=no] sram_sp_1024x32      top u_t1 ram_sp_1
            [SP=1_1_2, byp=no, diag=no, q_pipe=yes, repair=no] sram_sp_1024x32      top u_t1 ram_sp_2
            [SP=1_1_3, byp=no, diag=no, q_pipe=yes, repair=no] sram_sp_1024x32      top u_t1 ram_sp_3

###Total mbist memory instance = 3
###Total SRAM/REGFILE = 3
###Total SRAM_2P = 0
###Total SRAM_DP = 0
###Total REPAIR SRAM/REGFILE = 0
###Total REPAIR SRAM_2P = 0
###Total REPAIR SRAM_DP = 0
###Total ROM = 0
###top_default algorithm is= (March C+,SOLID)
##################Not Group Memory List#############################################
```

Figure 3-3 Memory Info Setting Information

3.2.2. PHYSICAL Sub Function Block

To execute the auto-grouping function, designers should input two files, the DEF file (Design Physical Information) and the *.SCOPE file (User-Defined Memory Controller Scope) into START.

Argument	Option
Description	
enable_physical	No, Yes
If this option is set to "yes", START will auto-group based on the DEF file (Design Physical Information). No: Group based on *.meminfo or auto-group based on other settings in the GROUP sub function block. Yes: Auto-group based on the DEF file	
physical_location_file	User defined
Set the paths of DEF file. Example: *set physical_location_file = ./design.def*	
controller_scope	User defined
After editing the 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)	
physical_logical	0 <= value <= 1
This option is to adjust the weight between physical coordinates and values defined in the **time_hierarchy** option. Example: *set physical_logical = 0* START 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. *set physical_logical = 1* START will execute memory grouping based on the value of the **time_hierarchy** option.	

3.3. MBIST Function Block

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><td colspan="2">Description</td></tr>
<tr><td>STIL_test_bench</td><td>No, Yes</td></tr>
<tr><td colspan="2">Generate a test pattern with the STIL format (IEEE 1450-Standard Test Interface Language) for the tester machine. 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</td></tr>
<tr><td>WGL_test_bench</td><td>No, Yes</td></tr>
<tr><td colspan="2">Generate a test pattern with the WGL format (Waveform Generation Language).

No: Not generate the test pattern with the WGL format
Yes: Generate the test pattern with the WGL format</td></tr>
<tr><td>bist_interface</td><td>basic, basicIO, ieee1500, ieee1149.1, ieee1687</td></tr>
<tr><td colspan="2">Select the MBIST interface. This option supports basic, basicIO, ieee1500, ieee1149.1, and ieee1687. Users can choose one of them to meet their design architectures.

Note: For more details of bist_interface, please refer to IO_Pin_Definition.
Note: If the repair mode is adopted, START only supports IEEE 1500.
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.</td></tr>
<tr><td>add_address_y</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option defines the MBIST algorithm also supports the Y direction. The generated testbench supports the X and Y addressing modes (X stands for the row of the memory, 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 the X and Y directions.

<table>
<tr><td>X_Y = 00</td><td>Write the MBIST pattern in the X direction only.</td></tr>
<tr><td>X_Y = 01</td><td>Write the MBIST pattern in the X direction first, and then the Y direction.</td></tr>
<tr><td>X_Y = 10</td><td>Write the MBIST pattern in the Y direction first, and then the X direction.</td></tr>
<tr><td>X_Y = 11</td><td>Write the MBIST pattern in the Y direction only.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Argument</th><th>Option</th></tr>
<tr><td colspan="2">Description</td></tr>
<tr><td colspan="2">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.</td></tr>
<tr><td>algorithm_selection</td><td>No, Outside, Scan</td></tr>
<tr><td colspan="2">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.

<table>
<tr><th>ALG</th><th>Selected Algorithm</th></tr>
<tr><td>0x00</td><td>Both algorithms (March 17N → Non-March BM)</td></tr>
<tr><td>0x01</td><td>March 17N only</td></tr>
<tr><td>0x10</td><td>Non-March BM only</td></tr>
<tr><td>0x11</td><td>NA</td></tr>
</table>
</td></tr>
<tr><td>clock_source_switch</td><td>No, Yes</td></tr>
<tr><td colspan="2">This option is used to select the testing frequency while 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.</td></tr>
<tr><td>clock_mux_method</td><td>No, async_reg, GCK_MUX, glitch_free</td></tr>
<tr><td colspan="2">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.</td></tr>
</table>

Argument	Option
Description	

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.

Please refer to Figure 3-4, Figure 3-5, and Figure 3-6 for the circuit schemes described above.

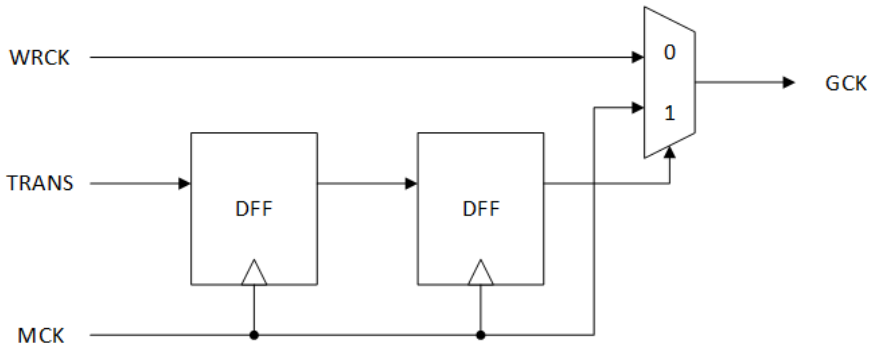


Figure 3-4 Scheme of the async_reg circuit

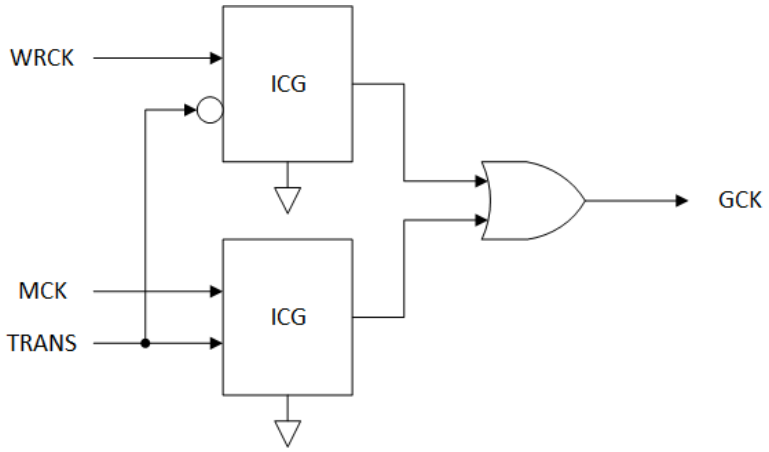
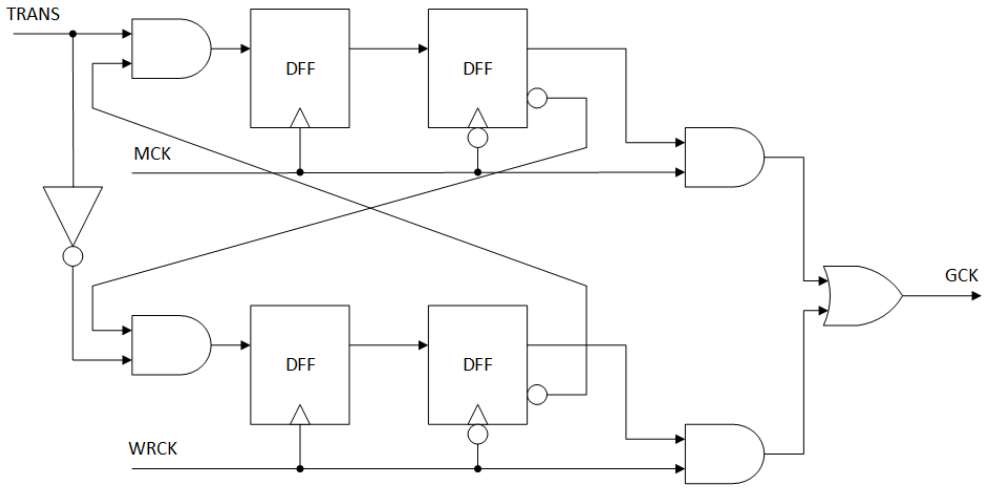


Figure 3-5 Scheme of the GCK_MUX circuit

Argument	Option
Description	
 **Figure 3-6 Scheme of the glitch_free circuit**	
clock_within_pll	No, Yes
This option is used to enable the MBIST/MBISR circuits for another clock input source, SCK. This SCK 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.	
diagnosis_method	No, Yes, Integer > 0
This option used to select the diagnosis mode can provide users with the failure time and failed memory information. No: Disable the diagnosis mode. Yes: Enable the diagnosis mode. Integer > 0: Enable the diagnosis mode and indicates the index of the error to report. If set to “0”, it will be treated as “no” and the diagnostic mode will be disabled.	
diagnosis_sharing	No, seq, ctr
When the **diagnosis_method** is enabled, this function can integrate the diagnosis circuits to reduce the area of MBIST circuits. No: Disable the sharing function. seq: Integrate the diagnosis circuits into the sequencer to share diagnosis storage, reducing the area of MBIST circuits.	

Argument	Option
Description	
ctr: The diagnosis circuits in the sequencer will be transferred to the controller to share diagnosis storage, reducing the area of MBIST circuits.	
diagnosis_faulty_items	algorithm, operation, element, seq_id, grp_id, address, ram_data, rom_data
Select the output items of the diagnosis result based on the failure analysis requirement. Example: *set diagnosis_faulty_items = algorithm, operation, element, seq_id, grp_id, address, ram_data, rom_data*	
rom_result_shiftin	No, Yes
This option is used to do ROM memory testing and import the signatures for internal verification. The scenario is used when the contents of 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.	
rom_result_shiftout	No, Yes
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 users set **rom_result_shiftout** to “yes”, the testing results of ROM memory will be transferred to the output interfaces via commands.	
Q_pipeline	No, Yes
This option is used to add an extra pipeline register to MBIST/MBISR logics. No: No extra register will be added to the data output of the memory model Yes: An extra register will be added to the data output of the memory model to enhance the operating timing of MBIST/MBISR logics.	
repair_method	No, soft_repair, hard_repair, hard_repair_with_buffer
The option is used for users to initiate the repair mode. No: Disable the repair method. soft_repair: Non-NVM repair. hard_repair: Store the repair-related information to NVM.	

Argument	Option
Description	
hard_repair_with_buffer:	Decide whether the repair information is read from NVM or from the buffer.

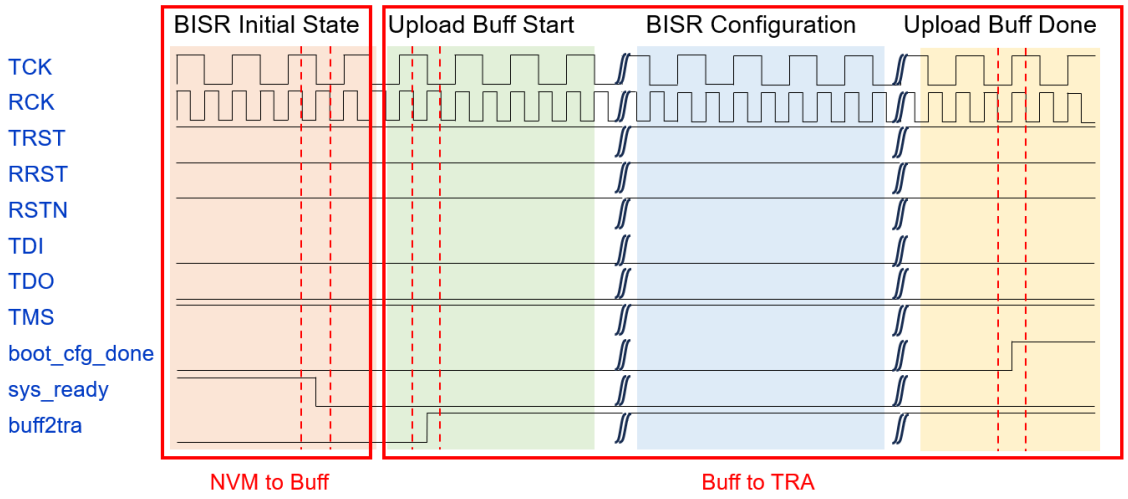


Figure 3-7 The Waveform of the hard_repair_with_buffer Option

asynchronous_reset	No, Yes

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.

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 3-8 is an example of synchronous/asynchronous circuit. 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.

Argument	Option
Description	

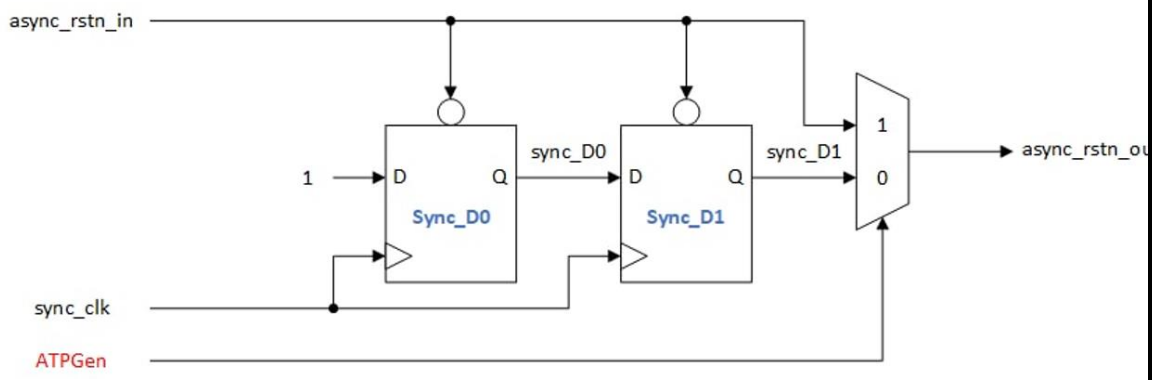


Figure 3-8 Example of Synchronous/Asynchronous Circuit

atpg_reset	No, Yes

This option is for users to reset the “Automatic Test Pattern Generation”. When the option is set to “yes”, the START tool will string all the reset signals under MBIST into a series of ATPG_rstn.

Note: In the BII flow, hookup the 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 3-9.

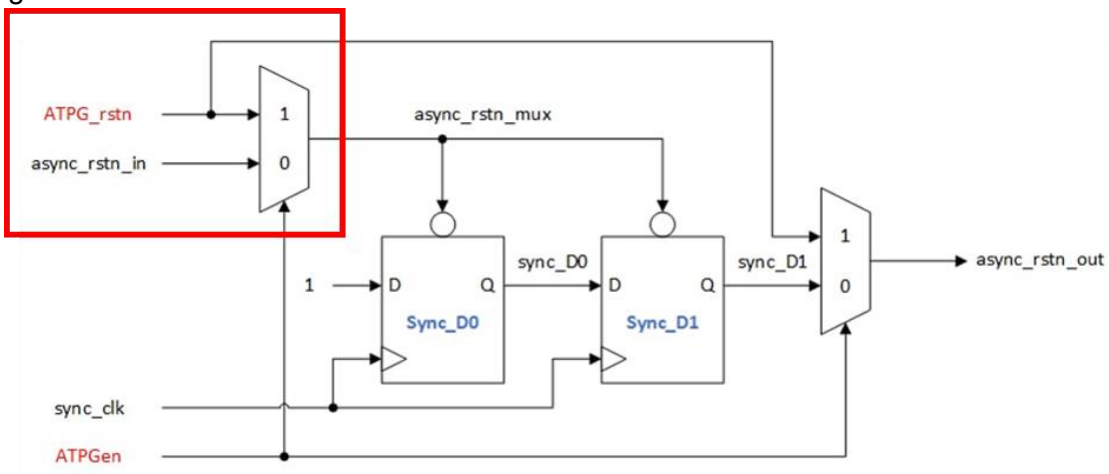


Figure 3-9 Example of ATPG Circuit

select_elem_testing	User defined

This option is for users to do testing with user-defined test algorithms rather than START 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.

Argument	Option
Description	

A programmable algorithm is presented as a PROG entry. Figure 3-10 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.

PROG	SEQ_ID	GRP_ID	MEB_ID	BG	ALG_CMD	ACTION_LOOP	MEN

SDI Command

Figure 3-10 Commands for Programmable Algorithm Function

Table 3-2 Commands for Programmable Algorithm

Command	Description
PROG	PROG = 0, executing the START built-in algorithm PROG = 1, executing the Test Element Change (TEC) function
PRL_ON	Sequencer ID for the memory
GRP_EN	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/MBISR circuits will execute user-defined algorithms. The width of ALG_CMD entry is based on March element definition.
ACTION_LOOP	ACTION_LOOP defines how many times an element is tested. This ACTION_LOOP entry is based on ALG_CMD, users can also define it. While PROG = 1, MBIST/MBISR circuits will execute TEC.
MEN	Enable the MBIST Testing

Argument	Option
Description	
Users can define the required test elements in define{algorithm_programmable} block: support_elements: r→ read w→ write n→ nop a→ background b→ inverse background max_prog_element: Set the number of elements for the programmed algorithm. If the built-in algorithm has more elements, the larger value will be used. max_elem_repeat: Set the maximum number of times an element can be repetitively tested. The default value is 1.	
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.	

Argument	Option
Description	
background_style	SOLID, 5A

The type of **background_style** can be set to “SOLID” and “5A” (Check Board), and the contents are defined in the bg_table file.
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.

Note: If users adopt “March Mdsn1” as an algorithm, **background_style** cannot be set to “5A”.

Table 3-3 BG Field Definition

BG [1:0]	Definition
00	SOLID + 5A
01	SOLID
10	5A
11	SOLID + 5A

background_bit_inverse	No, Yes

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.

Table 3-4 Example of Bit Inverse

Memory Address	SOLID BG Test Data	Description
0000_0000	0000_0000	testing data non-inversed
0000_0001	1111_1111	testing data inversed
0000_0010	0000_0000	testing data non-inversed
0000_0011	1111_1111	testing data inversed
…	…	…

Argument	Option
Description	
background_col_inverse	No, Yes

Column inverse means that BG testing data will be inversed based on 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

This option is for users to specify the background test pattern via the setting of user-defined background.
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 the 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_time	User defined

This option is used to define the retention time. The supported units of retention time are listed as Table 3-7 shows.

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, the March-RET algorithm is <(wb) (SLP) <(rb) >(wa) (SLP) >(ra). The (SLP) element indicates the sleep time is 1ms. If users want to extend the sleep time more than 1ms, they can specify the retention time by **retention_time**.

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

Note: The syntax differs depending on the retention time.

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.

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 the **retention_time** option in the BFL file or `testbech.v` file as shown in Figure 3-11.

Time: The retention time is fixed and can be 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 3-11 Example of Retention Time Option in testbech.v

Argument	Option
Description	
bypass_support	No, Wire, Reg

This option is used to define whether the bypass circuit is implemented by wire or register. 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.

No: Disable the bypass mode
Wire: Implementation of the bypass circuit through a wire as Figure 3-12
Reg: Implementation of the bypass circuit through a register as Figure 3-13

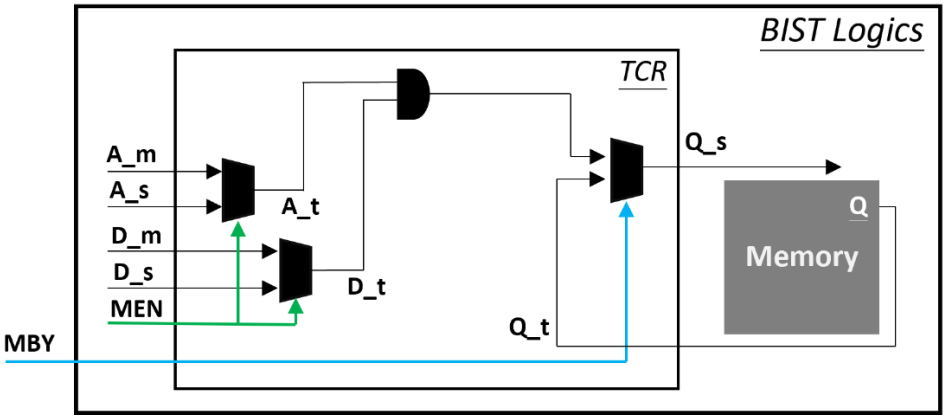


Figure 3-12 Implementation of Bypass Circuit through Wire

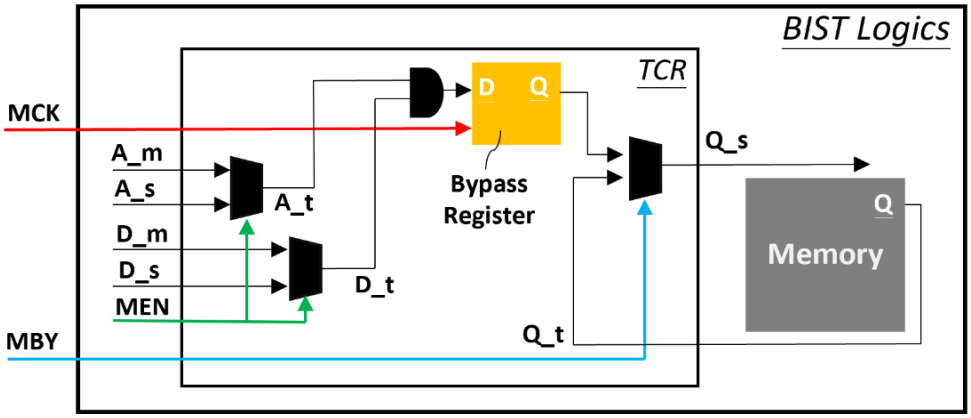


Figure 3-13 Implementation of Bypass Circuit through Register

Note: If the **bypass_support** option is enabled, the ATPG clock (Scan) will switch to the MBIST clock (MCK, Memory Clock) in the multi-source scenario.

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”. Moreover, the memory clock will be tied to “0” when **clock_switch_of_memory** is set to “yes”

No: The memory CS will be enabled. The clock remains fixed on the speedy clock (i.e., the MBIST clock, MCK), shown as Figure 3-14.

Yes: The memory chip select (CS) will be disabled (e.g., set to 1 if active low). The memory clock will be tied to 0, shown as Figure 3-15.

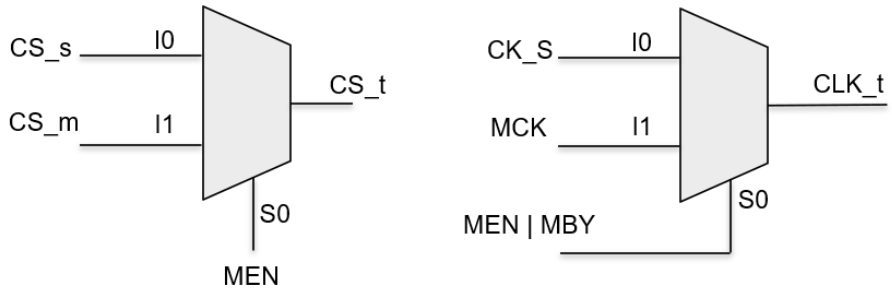


Figure 3-14 Implementation of CS & Clock Circuits (Option = No)

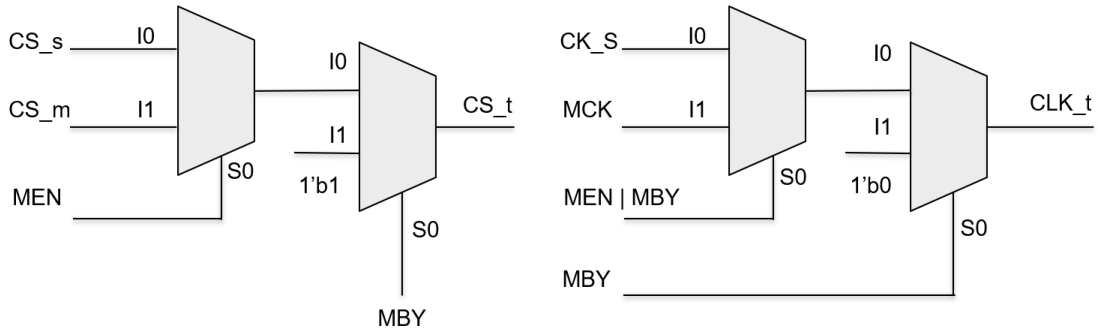


Figure 3-15 Implementation of CS & Clock Circuits (Option = Yes) (clock_switch_of_memory = Yes)

bypass_reg_sharing	1 <= (value) <= 1024

Users can set this option to define the register sharing number of bypass registers when **bypass_support** is set to “reg”. The range is between “1 ~1024”. START 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 shown in Figure 3-16.

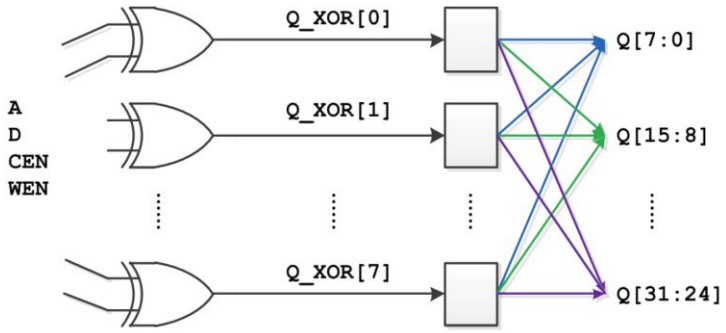
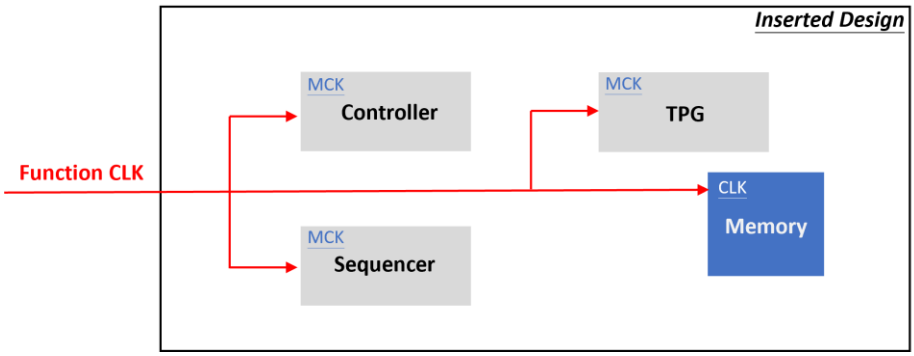


Figure 3-16 Example of Register Sharing

Argument	Option
Description	
bypass_clock	No, Yes
If users decide to implement bypass circuits through the “reg” method, they can turn on this option, too. In this case, MBIST circuits will have a dedicated input port BCK for bypass register and users can define the frequency of BCK based on the project requirement.	
clock_function_hookup	No, Yes
This option is used 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 workable only when the clock tracing is turned on. Figure 3-17 shows the clock architecture of this option.  **Figure 3-17 Clock Architecture of clock_function_hookup Option**	

Argument	Option
Description	
clock_switch_of_memory	No, Yes

Figure 3-18 shows the clock architecture of this option. The MCK also can be driven by the internal testing clock. Users can hookup the internal clock signal in the BII mode. This option must be set to “yes” if clock tracing is turned off.

No: The clock signal of the memory model will use the function CLK.

Yes: The clock signal of the memory model will be changed to MCK by the clock multiplexer in the test mode. The clock signal of the memory model is running at the same frequency as specified by users.

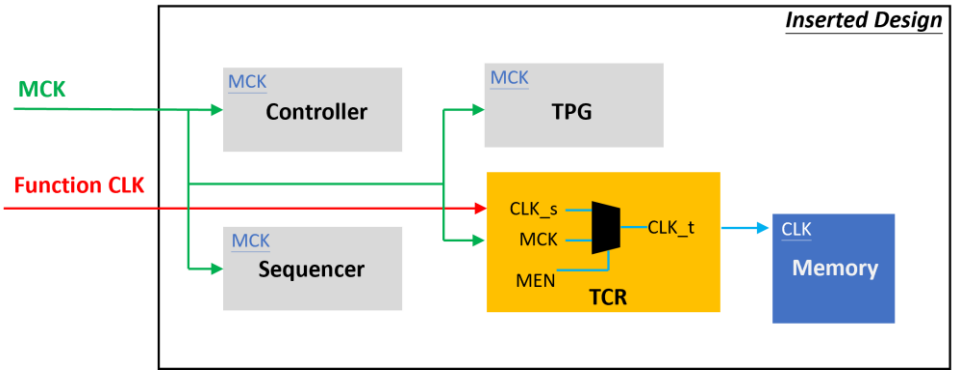


Figure 3-18 Clock Architecture of clock_switch_of_memory Option

diagnosis_memory_info	No, Yes

START will generate MBIST circuits with the N-bits width LATCH_GO output signal when this option is turned on. N means the number of memory models and each bit of the LATCH_GO signal indicates one memory model. Figure 3-19 shows the waveforms of LATCH_GO signals. When the signal turns from high to low, it indicates that the memory has failed.

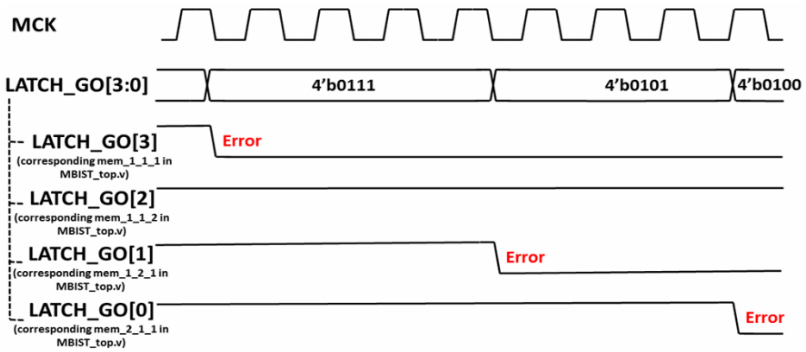


Figure 3-19 Diagnosis Failed Memory Information

Argument	Option
Description	
parallel_on	No, Yes
No: Not support parallel testing. Yes: It supports parallel testing. When this option is set to “yes” and users assign the testbench parameter PRL_ON to “1”, all memories under a controller will launch the testing simultaneously.	
reduce_address_simulation	No, Yes
START executes testing with the fixed four memory addresses as Table 3-8. It speeds up simulation by reducing the 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 Address

Memory Address Row	Memory Address Column
…000000	000000…
…000000	111111…
…111111	000000…
…111111	111111…

Table 3-9 Fixed Two Memory Address

Memory Address Row
…000000
…111111

Argument	Option
Description	
ecc_function	No, Yes

This option is for users to add ECC (Error Correction Code) circuits into customers' memories. ECC algorithm, adopting SECDED (Signal Error Correction and Double Error Detection), is capable of repairing the1-bit error. The 2-bit error can also be detected, yet it cannot be repaired.

No: Disable the ECC function
Yes: Enable the ECC function

To calculate the total required memory data width, the formula $2^{Parity-1} > Parity + Data\ Bit$ can be used. Table 3-10 is the example. When the data bit is more than 256, please calculate the memory data width according to the formula.
For example, If the memory data width is 16, the needed memory data width will be 22 (16 + 6). The extra 6 bits is for ECC circuit.

Table 3-10 Exampled Variables of Memory Data Width

Data Bit	Parity
8	5
16	6
32	7
64	8
128	9
256	10
…	…
M	N

M: Memory Data Width N: Parity

one_bit_for_ecc	No, Yes

BIST circuits will detect single-bit errors in the memory. Users can then determine whether ECC can be used to correct these errors.
Add ECC_MGO to the testbench:

test_result = {MGO, MRD, ECC_MGO};

Table 3-11 Testing Status

MGO	ECC_MGO	Meaning
1	1	Memory without defects
0	1	Can be fixed by ECC
0	0	Can't be fixed by ECC

pot	No, basic, hw_rom, rom, ram

When the system requires the Power_On testing, the following options are available. For more details, please refer to Chapter 9 in Application Notes.

No:	Disable the POT function.
Basic:	It indicates supporting some generic signals to enable or disable MBIST/MBISR and the test results. This function only supports the RAM test.
hw_rom:	It indicates that the POT testing commands will be designed to hardwired circuits. This function supports the ROM test.
Rom:	It indicates that the POT testing commands will be stored in the ROM. This function supports the ROM test.
Ram:	An RAM interface is supported for POT. It enables users to dynamically adjust MBIST test instructions.

RPCT	No, Yes

This option is designed to improve the timing from the NVM to each repair.

During the integration process, the `.integ` file is loaded, and a buffer is constructed for the RPCT option. Buff_count counts in reverse order to facilitate the state change between Read and Write operations.

In the BFL setting, the RPCT option is defined as "yes" (Figure 3-20). This allows the MBIST circuit to record the RPCT parameter when generating the `.integ` file, for the integrator's use.

define{BIST}
…
set RPCT = yes

Regarding the BII behaviors, when the RPCT option is enabled, it can be read in the `.integ` file. The following actions will be performed:

1. Create a buffer.
2. Based on buff_width, create buff_cnt and accumulate counts until the number of bits in memory is reached, triggering a state change.

```
set diagnosis_support              = no        # yes, no
set diagnosis_data_sharing         = no        # yes, no
set diagnosis_seq_sharing          = no
set diagnosis_ctr_sharing          = no
set diagnosis_memory_info          = no      # yes, no
set diagnosis_time_info            = no        # yes, no

set diagnosis_faulty_items         = algorithm, operation,
address, ram_data, rom_data
set parallel_on                    = no       # yes, no
set reduce_address_simulation      = yes        # yes, no
set RPCT                           = yes        # yes, no
set rom_result_shiftout            = yes        # yes, no
set Q_pipeline                     = no       # yes, no
set repair_mode                    = yes        # yes, no.
set soft_repair                    = no       # yes, no. y
set retention                      = time      # handshake
set retention_time                 = 0.0
```

Figure 3-20 The RPCT Option in the BFL Setting

NVM sharing	No, Yes

NVM sharing allows multiple memories to share a single NVM space, reducing overall NVM storage requirements. As shown in Figure 3-21, two memories with the same module name share one signature to reduce the requirement for NVM storage space.

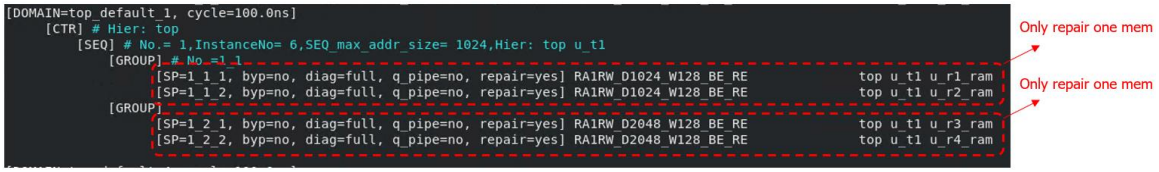


Figure 3-21 NVM Sharing

ecc_algorithm	2d1c, 1d1c

When **ecc_function** is set to “yes”, **ecc_algorithm** can be set to “1d1c” or “2d1c”. This setting has no effect when the ECC function is disabled.

2d1c:	Single Error Correcting Double Error Detecting.
1d1c:	Single Error Correcting Single Error Detecting.

3.3.1. Default Algorithm Sub Function Block

START provides various testing algorithms for users to choose according to different testing requirements. Figure 3-22 shows the default setting of single-port memories is the March C+ algorithm. If users want to add more testing algorithms into MBIST/MBISR 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 which START provides.

```
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 3-22 Default Algorithm Function Block

3.3.2. Programmable Algorithm 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 3-23), 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 3-12), 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 contents 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 3-12. Users can also find the definition in the `march_command.alias` file.

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

Table 3-12 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

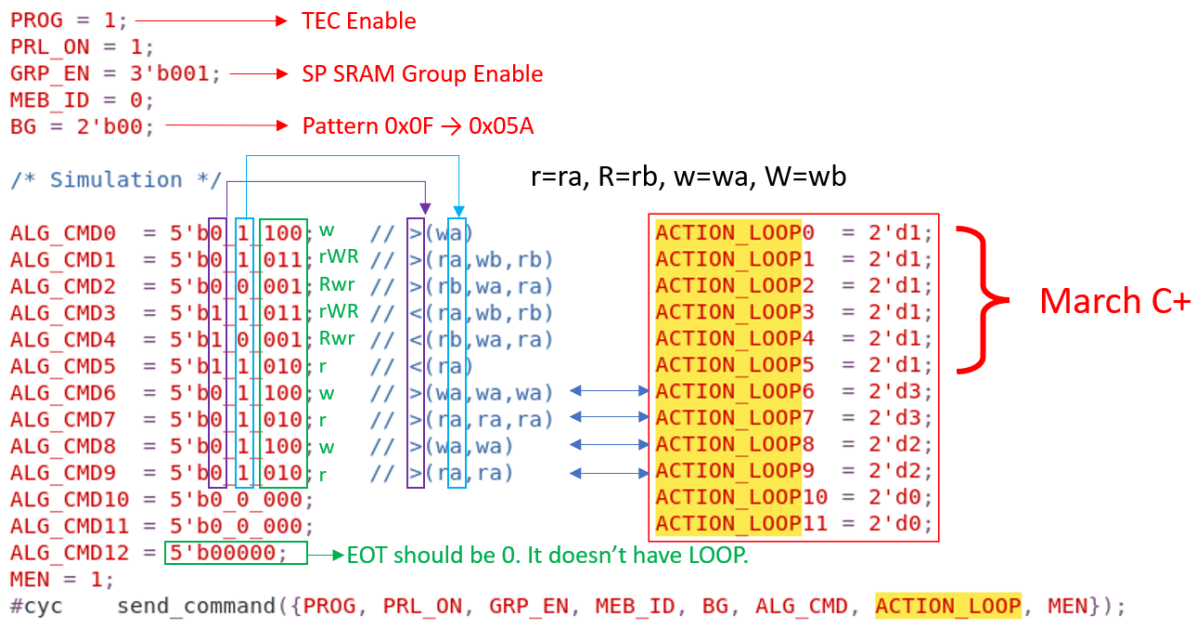


Figure 3-23 ACTION_LOOP & ALG_CMD in testbench

ACTION_LOOP is the number of times an element needs to be tested. Each ALG_CMD corresponds to a specific ACTION_LOOP setting. However, due to the EOT (End of Testing), which signifies the last command, the command execution will be terminated. Therefore, the last ALG_CMD will not have a corresponding ACTION_LOOP setting.

ACTION_LOOP = {ACTION_LOOP5, ACTION_LOOP4, … , ACTION_LOOP1, ACTION_LOOP0}

As the example illustrated in Figure 3-23, ALG_CMD0 to ALG_CMD5 represent the March C+ algorithm. Each element in this algorithm is tested only once. ALG_CMD6 and ALG_CMD7 involve three consecutive write and read operations, so the values of ACTION_LOOP6 and ACTION_LOOP7 are 3. ALG_CMD8 and ALG_CMD9 involve two consecutive write and read operations, so the values of ACTION_LOOP8 and ACTION_LOOP9 are 2.

4. START Output Files

This chapter introduces START tool's output files and their usage. These output files are divided into Self-MBIST, Self-MBISR, Inserted-MBIST and Inserted-MBISR part. Users can use these generated files to verify MBIST and MBISR circuits only, and also verify the MBIST and MBISR circuits integrated with customers' own logic design.

4.1. Self-MBIST Related Files

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

Table 4-1 Self-MBIST Related Files

	START output	Description
Project file (.bid)	[filename]_spec.bid	This is a START project file and includes all settings of START.
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 the original system design. `[filename].v` is HDL file of MBIST circuits.
Test bench (.v)	[filename]_tb.v	This is the 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. Self-MBISR Related Files

The generated self-MBIST related files include circuits (`.v`), a repair test bench (`.v`), a repair file-list file (`.f`), a repair synthesis script (`.tcl`) and a repair brief introduction file (`.html`). While users run a simulation with these output files in this section, it only simulates between MBISR circuits and memories only. For the following description, it uses RP as repair_prefix to explain the repair related files.

Table 4-2 Self-MBISR Related Files

	START output	Description
Repair self MBIST circuits (.v)	RP_[filename]_top.v RP_[filename].v	`RP_[filename]_top.v` includes memory models, fault memory models and MBISR circuits. It is integrated with MBISR circuits and memory models of original system design. `RP_[filename].v` is a HDL file of MBISR circuits.
Repair test bench (.v)	RP_[filename]_tb.v	This is a test bench for `RP_[filename].v`.
Repair file list (.f)	RP_[filename].f	This repair file-list file includes `RP_[filename]_top.v`, `RP_[filename].v` and memory models. This file is used for simulation.
Repair synthesis script (.tcl)	RP_[filename].tcl	This is a script file for users to do synthesis of MBISR circuits.

4.3. Insert MBIST Related Files

START can insert MBIST circuits, inserted-MBIST related, into customers' design. Users can verify inserted-MBIST with their own system circuit. Table 4-3 shows the related files for inserting MBIST circuits.

Table 4-3 Insert MBIST Related Files

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

4.4. Insert MBISR Related Files

When START inserts MBISR circuits and integrates with customer's system designs, it will generate files as Table 4-4. (In the following description, it uses RP as repair_prefix to explain that they are repair related files).

Table 4-4 Insert MBISR Related Files

	START output	Description
Inserted repair MBISR circuits (.v)	RP_[filename]_INS.v	RP_[filename]_INS integrates MBISR circuits with user's system designs. The [design] is the name of users' system design.
Repair test bench (.v)	RP_[filename]_tb_INS.v	`RP_[filename]_tb_INS.v` is the test bench for testing `RP_[design]_INS.v`.
Repair test bench (.v)	RP_[filename]_tb_RP_INS.v	`RP_[filename]_tb_RP_INS.v` is the test bench for the testing repair function.
Repair file list (.f)	RP_[filename]_INS.f RP_[filename]_INS_FAULT.f	File list `RP_[filename]_INS.f` records `RP_[design]_INS.v`, memory models and repair MBIST circuits. Different from file list `RP_[filename]_INS.f`, `RP_[filename]_INS_FAULT.f` also includes fault memory models.

4.5. “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 4-5 “Generate for Loop” Related Files

START 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.

4.6. Generate Folders

The following table shows the generated folders when executing START.

Table 4-6 Generated Folder

	START 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 the memory are tied to 0 or 1. This is used to verify the functional correctness of MBIST circuits. 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`.

4.7. Makefile

The START tool also generates a Makefile which includes related commands of simulation and synthesis for users to verify their designs. Using a Makefile, it can easily run various simulations along with MBIST/MBISR circuits.
Table 4-7 shows the commands of the Makefile.

Table 4-7 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 by 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 the 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 Inserted MBIST simulation with `[bistname]_tb_INS.v` and `[bistname]_INS_FAULT.f`, the simulation results that will be printed out in the command line window. This command is available while the BFL option **insertion** is “yes”.

	Command	Description
Inserted MBISR simulation	make [bistname] FUNC=tb_INS_RP	It is used to run Inserted MBISR simulation with `RP_[bistname]_tb_RP_INS.v` and `RP_[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 memory models.
Check syntax of inserted-MBIST circuits with nLint	make [bistname] FUNC=lint_INS	It is used to run a syntax check with `[bistname]_INS_FAULT.f` by using nLint. This checking result will be saved to the 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`.

4.8. 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 clcok skews, it is preferable to integrate clock cells from the standard library.

Note: Please change each module in the macro file into the coresponding standard cell. Figure 4-1 is an example of a clock gating module. Here "ctr_name" means the prefix that comes from the controller's 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.

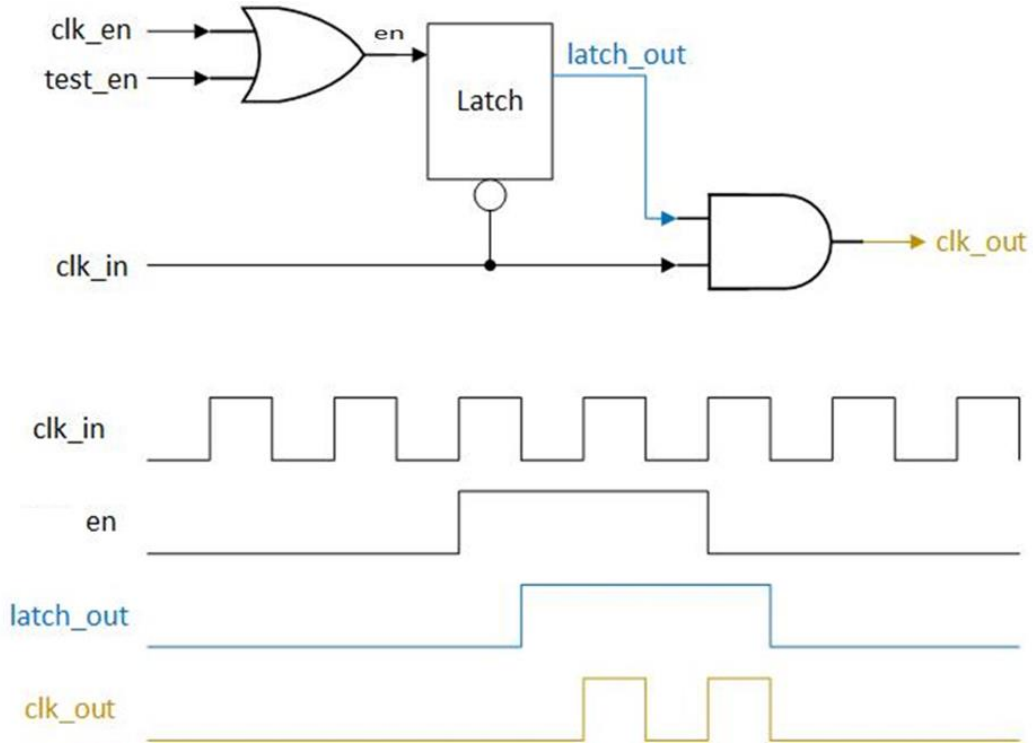


Figure 4-2 Clock Gating Cell with Waveform

5. START BII Files

START BII (MBIST/MBISR Integration Information) is a convenient and powerful tool that helps users integrate all controllers in a very short time. Using the brief BII configuration format, users could plan their own integration modules. This tool integrates all MBIST/MBISR controllers which are using IEEE 1500 interface by using an integrator module. Then, this integration module only uses one output, IEEE 1500, to communicate with an ATE (Automatic Test Equipment) machine. This design reduces the pin counts of the chip level.

5.1. Integrator Function Block

Design engineers could define the mapping of hookup pins and order the different MBIST/MBISR controllers in the following definition block.

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

The parameter [Name] (the default name is “INTEG”), can be modified by users and it will become the prefix name of the generated integrator files, for example, `INTEG.v` and `INTEG_INS.f`. This integrator module is used to integrate the WSI (Wrapper Serial Input) signal and WSO (Wrapper Serial Output) signal of each MBIST/MBISR controller.

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

Argument	**Option**
Description	
group_order	User defined
This option is for users to define the ordering of the MBIST/MBISR controller by setting a 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 specify the top-level module of the design.	

Argument	Option
Description	
TAP_hierarchy	User defined
This option is for users to define the hierarchy of the 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 file_list file is `*_INS.f`. If the BFL option, **fault_free** is set to “no”, the generated file_list 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 working directory of the BII flow. All the 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 file by using the vertical bar “|”. Example: *bist_1_spec.integ*	*bist_2_spec.integ*	*bist_3_spec.integ.*	
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 is not uniquified, the parsing mode must be setting to “RLT_only”. Example: *set parsing_mode = RTL_only*	
nvm_ctr	User defined
This option is for users to assign the nvm (non-volatile memory) controller of hard repair. If the customer has hard repair, users should consult iSTART to get this assignment.	
nvm_initial_address	User defined
This option is for users to set the initial address to access the NVM. Please fill in with the decimal format. Default value: 0 Example:	

Argument	Option
Description	
set nvm_initial_address = 8	
skip_include_check	No, Yes
No: Transform all the included paths in the output files into the absolute paths Yes: Only transform the included paths in the modified files (which are named with the keyword "_INS") into the absolute paths	
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 the memory under more than one controller one time. By using **serial_order**, users can assign the controller group priority testing order, and the controller group contains one or more controllers. For example, to set the priority testing order to [top_default0 & top_default1] => [RP_default0] => [RP_default1], users can refer to the example as follows, and set **parallel_on** in the BFL file to "yes". Example: *set serial_order = top_default0, top_default1 | RP_default0 | RP_default1* Note: Each memory controller under a group separated by "," is tested at the same priority order. Each testing controller group is separated by a vertical bar "|".	
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.	
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 MBIST logics insertion for the top module. Example: *set postfixinfo = ./mbist_top/EZ-BIST_top.postfixinfo*	
upload_repair_info	No, Yes
When this option is set to "yes", a buff2tra control signal is added to transfer buffer data to the TRA module. After the NVM is read and the data is stored in the buffer, subsequent reads use the buffer contents instead of reading from NVM again. The buff2tra control signal is to support loading buffer data to TRA only. This feature is mainly used for designs with multiple power domains. When a power domain is powered	

Argument	Option
Description	
on or off, it eliminates the need to reload NVM data, thereby accelerating the repair mapping process.	

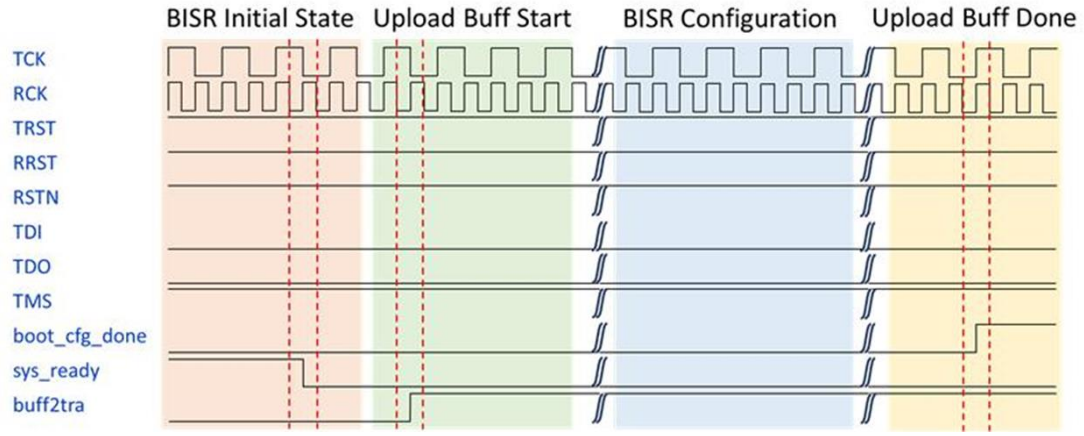


Figure 5-1 The NRLT Waveform

5.1.1. Hookup Sub Function Block

To reduce pin counts, the START tool does not use unique pins to control or communicate with MBIST/MBISR. The START tool could share debug pins like JTAG interface, IEEE 1149.7. Through the brief hookup pin setting in the BII file, START could achieve pin hookup automatically.

The following syntax defines a hookup pin and mapped pin in the hookup sub function block.

```
define{hookup}[signal]
    set dedicate_port = …
    set mapping_port = …
end_define{hookup}
```

Argument	Option
Description	
hookup	User defined
It indicates a hookup sub function block.	
signal	User defined

It indicates the signals on an 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 can be connected to MBIST by replacing the port. The hierarchy must be specified and can be separated by a space bar. Figure 5-2 is an 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}

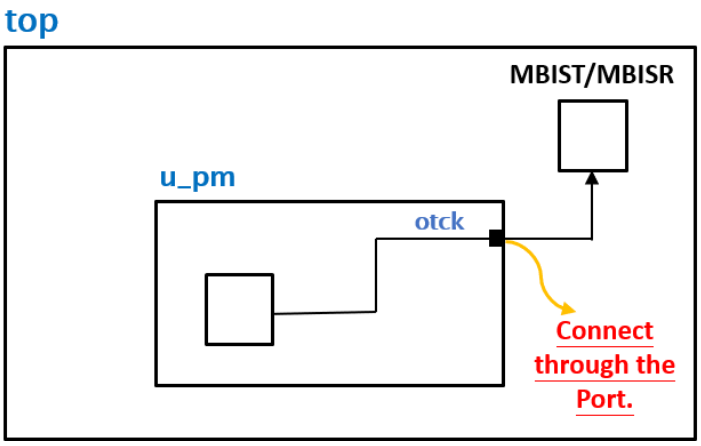


Figure 5-2 The Example of Port Connection

Argument	Option
Description	
mapping_wire	User defined

Connect to MBIST through the wire assignment. The hierarchy must be specified and can be separated by a space bar. Figure 5-3 is an example of wire connection.

The following is an 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.

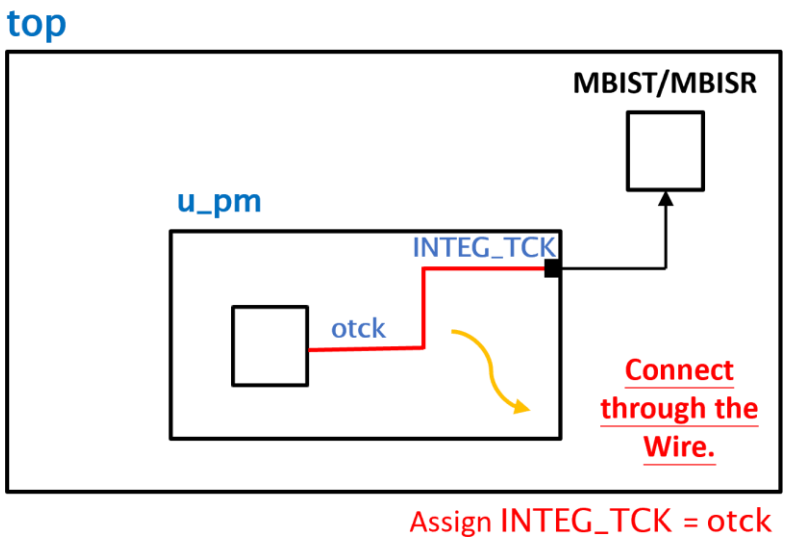


Figure 5-3 The Example of Wire Connection

5.1.2. Hookup Port Type

There are various ports that need to be connected during the BII stage. To build these connections, users could refer to Table 5-1 to set the [signal] assignment in the BII file.
[Controller name] means that the prefix name comes from the controller's name in customers' design.

In the Multi chain port type, the "#" in signal "integ_sys_ready_pd#" stands for the grouping number.

Table 5-1 Descriptions of Hookup Port Type

Port Type	Port Name	Signal Name in BII
CLK port	Controller MCK	[Controller name] + _MCK
Reset port	RSTN	RSTN
Repair port	RCK	RCK
	RRST	RRST
Hard repair port	SYS_READY	SYS_READY
	BOOT_CFG_DONE	BOOT_CFG_DONE
Bypass port	BCK	BCK
	MBY	MBY
IEEE 1149.1	TCK	TCK
	TRST	TRST
	TMS	TMS
	TDI	TDI
	TDO	TDO
IEEE 1149.7	TCKC	TCKC
	PWR_RST	WR_RST_N
	TMSC_IN	TMSC_IN
	TMSC_OUT	TMSC_OUT
	TMSC_OUT_EN	TMSC_OUT_EN
Power_on test	SYS_POT	SYS_POT

Port Type	Port Name	Signal Name in BII
	Controller MGO	[Controller name] + _ MGO
	Controller MRD	[Controller name] + _ MRD
	Controller RGO	[Controller name] + _ RGO
Multi chain	SYS_READY	integ_sys_ready_pd# (# stands for grouping number)
basic	SCK	SCK
	SRST	SRST
	SEN	SEN
	SDI	SDI
	SDO	SDO
basicIO interface (rom_result_shiftout turn on)	Controller SCK	[Controller name] + _SCK
	Controller SRST	[Controller name] + _SRST
	Controller SEN	[Controller name] + _SEN
	Controller SDO	[Controller name] + _SDO
basicIO function (According to and reference the hookup information table in the integ file)	Controller DIAG_D	[Controller name] + _DIAG_D
	Controller ALG_D	[Controller name] + _ALG_D
	Controller PRL_ON_D	[Controller name] + _ PRL_ON_D
	Controller SEQ_ID_D	[Controller name] + _SEQ_ID_D
	Controller GRP_EN_D	[Controller name] + _GRP_EN_D
	Controller MEB_ID_D	[Controller name] + _MEB_ID_D
	Controller BG_D	[Controller name] + _BG_D
	Controller RAT_D	[Controller name] + _RAT_D
	Controller X_Y_D	[Controller name] + _X_Y_D
	Controller RET_done_D	[Controller name] + _RET_done_D
	Controller RET_state_D	[Controller name] + _RET_state_D
	Controller MEN_D	[Controller name] + _MEN_D
	Controller MGO_D	[Controller name] + _MGO_D
	Controller MRD_D	[Controller name] + _MRD_D
	Controller SRD_D	[Controller name] + _SRD_D

Port Type	Port Name	Signal Name in BII
	Controller LATCH_GO_D	[Controller name] + _LATCH_GO_D

5.1.3. Group Sub Function Block

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

The following syntax defines the Group sub function block.

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

Argument	Option
Description	
group_name	User defined
Define the name which is listed in the **group_order** parameter.	
bist_order	User defined
This option is for user to establish the connection order of controllers in a chain. For example, set **bist_order** to “bist1_controller, bist2_controller, bist3_controller”. Then, 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 such as 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}

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. MBIST circuits will be reset after these stable cycles. Default Value: 100000	
reset_cycle	User defined
This option defines the waiting cycles to reset MBIST circuits. While PLL is stable, MBIST/MBISR circuits will be reset after the period of **reset_cycle**.	
file_format	STIL format, WGL format, Verilog
This option defines the output format of the testbench. Default Setting: “Verilog”	

5.2.1. Initial_sequence Sub Function Block

The Initial_sequence sub function defines signals on the top level that can force the system into the testing mode. In a real chip, users may use some signals to switch the function or testing mode. To run the MBIST mode simulation, the START tool will force these signals into 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}
```

Argument	Option
Description	
width	User defined
This parameter defines the width of the signal. On the top level, users will use pins to switch the 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 which is 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.	

6. Appendixes

6.1. “Include” Case

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

6.2. Parsing Mode

If the design is RTL, please make sure it could be synthesized. Otherwise, START cannot parse the design for inserting memory MBIST circuits to the design.

Due to the diverse syntax of RTL, we suggest users to use netlist as input if there are parsing issues with the RTL.

6.3. *.rcf File

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

6.4. Supported Testing Algorithm

Table 6-1 Testing Algorithms for SRAM in START

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)

Memory Type	Name	Fault Detection	Algorithm
	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)

Memory Type	Name	Fault Detection	Algorithm
	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 START
**: Support two port memory only
***: Support dual port memory only

±1: used to increase/ decrease the memory address

|: used to separate the operation of different ports
>: indicates the address count from o to the highest address in a memory
<: indicates the address count from the highest address to 0 in a memory
a: indicates the test pattern
b: indicates inverse “a” test pattern

Table 6-2 Testing Algorithms for ROM (ROM Test) in START

Memory Type	Name	Address sequence	Operation	Description
ROM	*ROM Test	LFSR	(rc)	Reads, compresses the ROM’s content, and 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.

*: Default algorithm of START

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 the BFL file: `default.bfl`

Table 6-4 The Default Settings of the 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
bypass_reg_sharing	1
clock_function_hookup	no
clock_switch_of_memory	yes
clock_source_switch	no
clock_within_pll	no
diagnosis_method	no
diagnosis_sharing	no
diagnosis_memory_info	no
diagnosis_faulty_items	all

BFL File Column	Default Value
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
top_default_ter_1_1_15	402.840007
top_default_ter_1_1_16	402.840007

Referenced Library	Total Area
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 option **asynchronous_reset** to “cs”, and 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** is 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 the option **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
```