

iSTART

START

Quick Start Guide

SSG 130

Outline

Outline	1
List of Figures	2
Chapter 1 START Set Up	1
1.1 START Install	1
1.2 START License Invoke	1
1.3 The Way to Confirm if START License be Invoked	1
1.4 Set Up Environment	2
1.5 Alias START	2
1.6 START License Update	2
Chapter 2 START BFL Flow	3
2.1 Untar Example Case	3
2.2 Check if START Is Executed Successfully	3
2.3 Create File-List File (*.f)	4
2.4 Memory Check with START (Optional)	4
2.5 Generating and Setting BFL File	4
2.6 Execute START with BFL File	7
2.7 Setting Memory Info File (Optional)	8
2.8 Execute START by BFL and Memory Info File	8
Chapter 3 Simulation	10
3.1 Self-Simulation	10
3.2 Inserted-Simulation	11
3.3 Simulation with Repair Function	12
3.4 Simulation with Fault Memory Models	13
Chapter 4 Synthesis	15
4.1 Synthesis	15
Appendix A	16
A.1 Memory Check with START memchecker	16

List of Figures

Figure 2-1 START Command	3
Figure 2-2 run.f File	4
Figure 2-3 Generate BFL File	5
Figure 2-4-1 BFL File Example	6
Figure 2-4-2 BFL File Example	6
Figure 2-5 Grouping Informatio	7
Figure 2-6 Auto Insertion Information	7
Figure 2-7 Memory Info Setting Information	8
Figure 2.8 memory_list Option	9
Figure 2-9 Grouping Information with Memory Info File	9
Figure 2-10 Auto Insertion Information with Memory Info File	9
Figure 3-1 Testbench Architecture of Self-Simulation	10
Figure 3-2 Delay Parameter Setting	11
Figure 3-3 Self-Simulation Result	11
Figure 3-4 Testbench Architecture of Insert-Simulation	12
Figure 3-5 Inserted-Simulation Result	12
Figure 3-6 simulation waveform of fault memory models	14
Figure 3-7 example of error bit definitions	14
Figure 4-1 Synthesis Output of top_default Controller	15
Figure A-1 memchecker Information	16

Chapter 1 START Set Up

1.1 START Install

To setup START tool, first at all, decompress START package in workstation environment.

- START execution file : `START-CentOS-6.5-x86_64-xxxxx.tar.gz`

Decompress method : Download whole compressed files at the same folder and execute the following command.

```
unix% tar -xvzf START-CentOS-6.5-x86_64-xxxxx.tar.gz
```

- START license file : `LM-CentOS-6.5-x86_64-xxxxx.bz2`

Decompress method : Execute the following command.

```
unix% tar jxf LM-CentOS-6.5-x86_64-xxxxx.bz2
```

After decompressing correctly, you will get the following files and folders :

- (1). `BRAINS_lic_2020xxxxx.dat`
- (2). `env_setup` folder
- (3). `lmInvoke`
- (4). `lmQry`

1.2 START License Invoke

Execute the following command under the folder `LM-CentOS-6.5-x86_64-xxxxx` :

```
unix% ./lmInvoke -f BRAINS_lic_2020xxxxx.dat
```

1.3 The Way to Confirm if START License Launched

```
unix% ./lmQry 4141 @your_IP
```

If START license is launched successfully, it will see like following message :

feature: BRAINS-DirectIO_1, version: 2016.10, num_lic: 999999, expired on: 2022/3/31;

feature: BRAINS-Repair, version: 2016.10, num_lic: 999996, expired on: 2022/3/31;

.....

1.4 Set Up Environment

Set up the environment with HOY_LIC_FILE for invoking the license server.

By Bash shell :

```
unix% export HOY_LIC_FILE=4141@hostname
```

or

```
unix% export HOY_LIC_FILE=4141@IP
```

By C shell (Tcsh) :

```
unix% setenv HOY_LIC_FILE 4141@hostname
```

or

```
unix% setenv HOY_LIC_FILE 4141@IP
```

1.5 Alias START

By Bash shell :

```
unix% alias start=/usr/home/tools/START-CentOS-6.5-x86_64-xxxxx/bin/start
```

By C shell (Tcsh) :

```
unix% alias start /usr/home/tools/START-CentOS-6.5-x86_64-xxxxx/bin/start
```

1.6 START License Update

Once license cannot invoke successfully, please confirm if the lmlnvoke is occupied by the previous license with the “top” command.

Once START license system is occupied by the previous license file, please kill exist license and re-lmlnvoke the new license.

```
unix% top -u user_name
```

```
unix% kill -9 lmlnvoke_PID
```

```
unix% ./lmlnvoke -f BRAINS_lic_2020xxxxx.dat
```

Chapter 2 START BFL Flow

If designer is first time to execute START, here is an example case for designer to understand BFL (BIST Feature List) flow of START.

Please notice that this evaluation package, NDACase, is designed only for single clock domain design.

2.1 Untar example case

```
unix% tar xvzf NDACase.tgz
unix% cd NDACase
```

2.2 Check if START tool workable

Using following command under executing folder

```
unix% start --help
```

```
usage: start [-h] [-bii INTEGRATE_FILE] [-bfl BFL_FILE]
            [-f RUN_FILE [RUN_FILE ...]] [-v VERILOG_FILE [VERILOG_FILE ...]]
            [-W DIR] [-top MODULE] [-I] [-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]

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 BIST to design
-integ FILE [FILE ...]
                        input integ file(s)
-u FILE [FILE ...], -udm FILE [FILE ...]
                        Input User Defined Memory file(s)
-pm Verilog type, --parsingmode Verilog type
                        Specifies the Verilog input file types, RTL_only,
                        Netlist_only
--integrator            integrate multiple BIST
--faultfree            create file without faulty memory
--ug UDM_FILE config_FILE
                        User Defined Memory Generator
--rcfg Addr_length Data_width output_FILE
                        RCF Generator
--tempgen              BRAINS Defined Format Template generator
--memchecker           BRAINS auto memory identify (memory identify only)
--memlib2udm MEMLIB_FILE
                        Generate BRAINS udm file. (*.memlib -> *.udm, *.lvlib
                        -> *.udm)
--bflconfig [BFL_FILE]
                        GUI version of the BFL adjustment tool
```

Figure 2-1 The prompt after executing START help

2.3 Create File-List file (*.f)

The easiest way to execute START is providing a completed design and File-List file (*.f). The File-List file format are the same as NC-Verilog's file which is including the following items.

- Design.v (RTL or netlist)
- Memory.v
- Standard_cell.v (when your design is netlist)
- Parameter, e.g. +define+ \ +incdir+PATH/DIR ...

Figure 2-2 shows an example *.f file, run.f, for this test case, NDAcase. User should add `-v` option in front of each memory verilog 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

./top.v
```

Figure 2-2 run.f File

2.4 Memory checking by START (Optional)

START can assist to identify user's memory macros automatically by executing *memchecker* command. This command can check if user's memory models can be recognized by START. For detail, please refer to Appendix A.

If memory models can not to be recognized, designer can edit UDM (User Defined Memory) information to let START recognize these memory models. START provides a template *.udm file, users can modify it according to memory models. For detail, please refer to user manual chapter 7.

2.5 Generating and setting BFL file

```
unix% cd NDAcase
unix% start --tempgen
```

Please choose item 2 as Figure 2-3: MBIST/BISR Feature List (BFL) and the *start_template.bfl* file will be generated to the working folder.

```
This computer program constitutes or contains trade secrets and confidential
information of HOY-Technologies Inc. or its licensors. This computer program is
protected by copyright law and international treaties.

[17:16:02] START : ver. 2018.05 build 3001 : (c) Copyright 2009- HOY-Technologi
es. All rights reserved.

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

Figure 2-3 Generate BFL File

BFL file includes related requirements of MBIST/BISR circuit specification. User can modify it based on project requirement. Figure 2-4-1 and Figure 2-4-2 shows an example BFL setting for this test case. Users also can refer to *ref* folder in test case package to find an example BFL file.

- set verilog_path : ./run.f
- set top_module_name : top
- set top_hierarchy : top
- set clock_trace : no
- set auto_group : yes
- set insertion : yes
- set integrator_mode : no
- set work_path : ./mbist

In *define{BIST}* function block:

- set bist_interface : ieee1500
- set clock_swath_of_memory : yes

```

define{OPTION}
  set verilog_path      = ./run.f          # /absolute path/design.f
  set user_define_memory =               # /absolute path/memory.udm
  set top_module_name  = top              # design top
  set top_hierarchy    = top              # BIST top
  set clock_trace      = no               # yes, no (User group instances will all be un-group when setting yes)
  set auto_group       = yes              # yes, no
  set insertion        = yes              # yes, no
  set integrator_mode  = no               # yes, no
  set work_path        = ./mbist          # ./work
  set fault_free       = no               # yes, no
  set parsing_mode     = RTL_only         # RTL_only, Netlist_only
  set repair_prefix    = RP               # prefix for repair module

  define{CLOCK}
    set sdc_file      =                   # /absolute path/design.sdc
    define{100MHz}
      set clock_cycle = 10                # integer
      set clock_source_list = top CLK1    # top design1 CLK
    end_define{100MHz}
    define{10MHz}
      set clock_cycle = 100              #integer
      set clock_source_list = top CLK2    # top design2 CLK
    end_define{10MHz}
  end_define{CLOCK}

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

    define{PHYSICAL}
      set enable_physical = no            # yes, no
      set physical_location_file =         # /absolute path/design.def
      set distance_limit  = 1
      set physical_logical = 0.5
    end_define{PHYSICAL}
  end_define{GROUP}
end_define{OPTION}

```

Figure 2-4-1 BFL File Example

```

define{BIST}
  set STIL_test_bench      = no           # yes, no
  set asynchronous_reset   = yes          # yes, no
  set bist_interface       = ieee1500     # minibist, basic, ieee1149.1, ieee1500
  set address_fast_y       = no           # yes, no
  set program_algorithm    = no           # yes, no
  set algorithm_selection  = no           # no, outside, scan
  set background_style     = SOLID        # SOLID, WORD, 5A
  set background_bit_inverse = no         # yes, no
  set background_col_inverse = no         # yes, no
  set bypass_support       = no           # no, wire, reg
  set bypass_clock         = no           # yes, no
  set bypass_include_bist_pin = no        # yes, no
  set bypass_reg_sharing   = 1            # 1 ~ 1024
  set clock_function_hookup = no          # yes, no
  set clock_switch_of_memory = yes        # yes, no
  set clock_source_gated   = no           # yes, no
  set clock_source_switch  = no           # yes, no
  set clock_within_pll     = no           # yes, no
  set diagnosis_support    = no           # yes, no
  set diagnosis_data_sharing = no         # yes, no
  set diagnosis_memory_info = no          # yes, no
  set diagnosis_time_info  = no           # yes, no
  set diagnosis_faulty_items = algorithm, operation, element, seq_id, grp_id, address, ram_data,
  set parallel_on          = no           # yes, no
  set reduce_address_simulation = no      # yes, no
  set rom_half_access      = no           # yes, no
  set rom_result_shiftout  = no           # yes, no
  set specify_clock_mux    = no           # yes, no
  set specify_dt_port_value = no         # yes, no
  set Q_pipeline           = no           # yes, no
  set repair_mode          = yes          # yes, no. Repair mode with redundancy memory model.
  set soft_repair          = no           # yes, no. yes = soft repair, no = hard repair

```

Figure 2-4-2 BFL File Example

2.6 Execute START with BFL file

The command to execute START with BFL file is:

```
unix% start -bfl start_template.bfl
```

Please notice that if the location of files defined in BFL file is relative path instead of absolute path, the relative path is based on location of BFL file.

After executing commands above, user can see messages like Figure 2-5 and Figure 2-6. All of these results will also output to **mbist** folder. User can find **START_memory_spec.meminfo** file in **mbist** folder, which represents grouping architecture.

```
[16:36:33] [CHECK][GROUPING] top_default: seq 1, grp 1, 8 members
[16:36:33] [CHECK][GROUPING] top_default: seq 2, grp 1, 1 members
[16:36:33] [CHECK][GROUPING] top_default: seq 3, grp 1, 2 members
[16:36:33] [CHECK][GROUPING] top_default: seq 4, grp 1, 1 members
```

Figure 2-5 Grouping Information

```
[16:36:34] [INSERT]
[16:36:34] [INSERT] #===== BIST Insert Path =====#
[16:36:34] [INSERT] #
[16:36:34] [INSERT] #      ----- Controller -----      #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] # CTR(top_default) : top                #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] #      ----- Sequencer -----      #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] # SEQ 1 : top.u_t1                      #
[16:36:34] [INSERT] # SEQ 2 : top.u_t1                      #
[16:36:34] [INSERT] # SEQ 3 : top.u_t1                      #
[16:36:34] [INSERT] # SEQ 4 : top.u_t1                      #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] #      ----- TPG -----            #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] # TPG top_default_t_1_1_1 : top.u_t1    [sram_sp_1024x32] (ram_1) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_2 : top.u_t1    [sram_sp_1024x32] (ram_2) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_3 : top.u_t1    [sram_sp_1024x32] (ram_3) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_4 : top.u_t1    [sram_sp_1024x32] (ram_4) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_5 : top.u_t1    [sram_sp_1024x32] (ram_e) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_6 : top.u_t1    [sram_sp_1024x32] (ram_w) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_7 : top.u_t1    [sram_sp_1024x32] (ram_x) #
[16:36:34] [INSERT] # TPG top_default_t_1_1_8 : top.u_t1    [sram_sp_1024x32] (ram_y) #
[16:36:34] [INSERT] # TPG top_default_t_2_1_1 : top.u_t1    [rf_2p_24x28] (u_2p)    #
[16:36:34] [INSERT] # TPG top_default_t_3_1_1 : top.u_t1    [sram_dp_1024x64] (u_dp) #
[16:36:34] [INSERT] # TPG top_default_t_3_1_2 : top.u_t1    [sram_dp_1024x64] (u_dp2) #
[16:36:34] [INSERT] # TPG top_default_t_4_1_1 : top.u_t1    [rom_6144_64] (u_rom)  #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] #      ----- END -----            #
[16:36:34] [INSERT] #
[16:36:34] [INSERT] #=====
[16:36:34] [INSERT]
[16:36:34] [INSERT] Perform auto insertion ... done (0.11 sec)
[16:36:34] [INSERT] Create synthesis Testcase TCG ... /home/jisomy/TestCase/MPAapp/...
```

Figure 2-6 Auto Insertion Information

2.7 Setting Memory Info File (Optional)

After executing START, the memory info file will be generated to **mbist** folder. Memory info file represents grouping architecture. If designer wants to adjust memory grouping according to your design requirement, designer can modify memory info files directly.

Memory info file includes following items.

- Clock domain : Memory clock domain and testing clock cycle
- Memory module : Memory module name and memory hierarchy
- Bypass/ Diagnosis : Setting values of bypass function and diagnosis function
- q_pipeline : Setting value of q_pipeline option
- Group Architecture : Grouping architecture information (including controller and sequencer)

Figure 2-7 shows the content of **START_memory_spec.meminfo**.

```
[DOMAIN=top_default, cycle=100.0ns]
[CTR] # Hier: top
[SEQ] # No.= 1, InstanceNo= 8, SEQ_max_addr_size= 1024, Hier: top u_t1
  [GROUP] # No.=1_1
    [SP=1_1_1, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_1
    [SP=1_1_2, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_2
    [SP=1_1_3, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_3
    [SP=1_1_4, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_4
    [SP=1_1_5, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_e
    [SP=1_1_6, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_w
    [SP=1_1_7, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_x
    [SP=1_1_8, byp=no, diag=no, q_pipe=no]sram_sp_1024x32      top u_t1 ram_y
[SEQ] # No.= 2, InstanceNo= 1, SEQ_max_addr_size= 24, Hier: top u_t1
  [GROUP] # No.=2_1
    [2P=2_1_1, byp=no, diag=no, q_pipe=no]rf_2p_24x28          top u_t1 u_2p
[SEQ] # No.= 3, InstanceNo= 2, SEQ_max_addr_size= 1024, Hier: top u_t1
  [GROUP] # No.=3_1
    [DP=3_1_1, byp=no, diag=no, q_pipe=no]sram_dp_1024x64      top u_t1 u_dp
    [DP=3_1_2, byp=no, diag=no, q_pipe=no]sram_dp_1024x64      top u_t1 u_dp2
[SEQ] # No.= 4, InstanceNo= 1, SEQ_max_addr_size= 6144, Hier: top u_t1
  [GROUP] # No.=4_1
    [ROM=4_1_1, byp=no, diag=no, q_pipe=no]rom_6144_64         top u_t1 u_rom
```

Figure 2-7 Memory Info Setting Information

2.8 Using memory info file as default memory grouping

If designer uses memory info file, **START_memory_spec.meminfo**, as memory grouping setting. Designer should turn off **“auto_group”** option and specify **memory_list** option to the path of **START_memory_spec.meminfo** in BFL configuration file as Figure 2-8.

After executing START BFL flow with Memory info file, START can automatically modify naming and operating frequency of BIST and BISR controller. It also assists designer to do grouping related setting according to designer’s requirement. There is example memory info file in **ref** folder of NDCase. Executing START with modified BFL file which includes modified memory info file and command like below, it will show the prompt like Figure 2-9 and Figure 2-10.

In this example case, there are one extra group for sequencer 1 and the name of BIST controller is changed to **testcase**.

unix% *start -bfl start_template.bfl*

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

Figure 2.8 memory_list Option

```
[17:39:24] [CHECK][GROUPING] testcase: seq 1, grp 1, 5 members
[17:39:24] [CHECK][GROUPING] testcase: seq 1, grp 2, 3 members
[17:39:24] [CHECK][GROUPING] testcase: seq 2, grp 1, 1 members
[17:39:24] [CHECK][GROUPING] testcase: seq 3, grp 1, 2 members
[17:39:24] [CHECK][GROUPING] testcase: seq 4, grp 1, 1 members
```

Figure 2-9 Grouping Information with Memory Info File

```
[17:39:25] [INSERT]
[17:39:25] [INSERT] #===== BIST Insert Path =====#
[17:39:25] [INSERT] #
[17:39:25] [INSERT] #      ----- Controller -----
[17:39:25] [INSERT] #
[17:39:25] [INSERT] # CTR(testcase) : top
[17:39:25] [INSERT] #
[17:39:25] [INSERT] #      ----- Sequencer -----
[17:39:25] [INSERT] #
[17:39:25] [INSERT] # SEQ 1 : top.u_t1
[17:39:25] [INSERT] # SEQ 2 : top.u_t1
[17:39:25] [INSERT] # SEQ 3 : top.u_t1
[17:39:25] [INSERT] # SEQ 4 : top.u_t1
[17:39:25] [INSERT] #
[17:39:25] [INSERT] #      ----- TPG -----
[17:39:25] [INSERT] #
[17:39:25] [INSERT] # TPG testcase_t_1_1_1 : top.u_t1 [sram_sp_1024x32] (ram_1)
[17:39:25] [INSERT] # TPG testcase_t_1_1_2 : top.u_t1 [sram_sp_1024x32] (ram_2)
[17:39:25] [INSERT] # TPG testcase_t_1_1_3 : top.u_t1 [sram_sp_1024x32] (ram_3)
[17:39:25] [INSERT] # TPG testcase_t_1_1_4 : top.u_t1 [sram_sp_1024x32] (ram_4)
[17:39:25] [INSERT] # TPG testcase_t_1_1_5 : top.u_t1 [sram_sp_1024x32] (ram_e)
[17:39:25] [INSERT] # TPG testcase_t_1_2_1 : top.u_t1 [sram_sp_1024x32] (ram_w)
[17:39:25] [INSERT] # TPG testcase_t_1_2_2 : top.u_t1 [sram_sp_1024x32] (ram_x)
[17:39:25] [INSERT] # TPG testcase_t_1_2_3 : top.u_t1 [sram_sp_1024x32] (ram_y)
[17:39:25] [INSERT] # TPG testcase_t_2_1_1 : top.u_t1 [rf_2p_24x28] (u_2p)
[17:39:25] [INSERT] # TPG testcase_t_3_1_1 : top.u_t1 [sram_dp_1024x64] (u_dp)
[17:39:25] [INSERT] # TPG testcase_t_3_1_2 : top.u_t1 [sram_dp_1024x64] (u_dp2)
[17:39:25] [INSERT] # TPG testcase_t_4_1_1 : top.u_t1 [rom_6144_64] (u_rom)
[17:39:25] [INSERT] #
[17:39:25] [INSERT] #      ----- END -----
[17:39:25] [INSERT] #
[17:39:25] [INSERT] #=====
[17:39:25] [INSERT]
[17:39:25] [INSERT] Perform auto insertion ... done (0.08 sec)
```

Figure 2-10 Auto Insertion Information with Memory Info File

Chapter 3 Simulation

3.1 Self-Simulation

Figure 3-1 shows the architecture of Testbench for self-simulation. This self-simulation is used to verify function correctness of BIST and BISR circuit only. This system design does not include in self-simulation. The simulation environment is built by “*make*” language. User can refer to *Makefile.top_default* file. This file defines commands and parameters for executing simulation.

If designer wants to debug with waveform file. designer can turn on related dump parameters in *top_default.f* file.

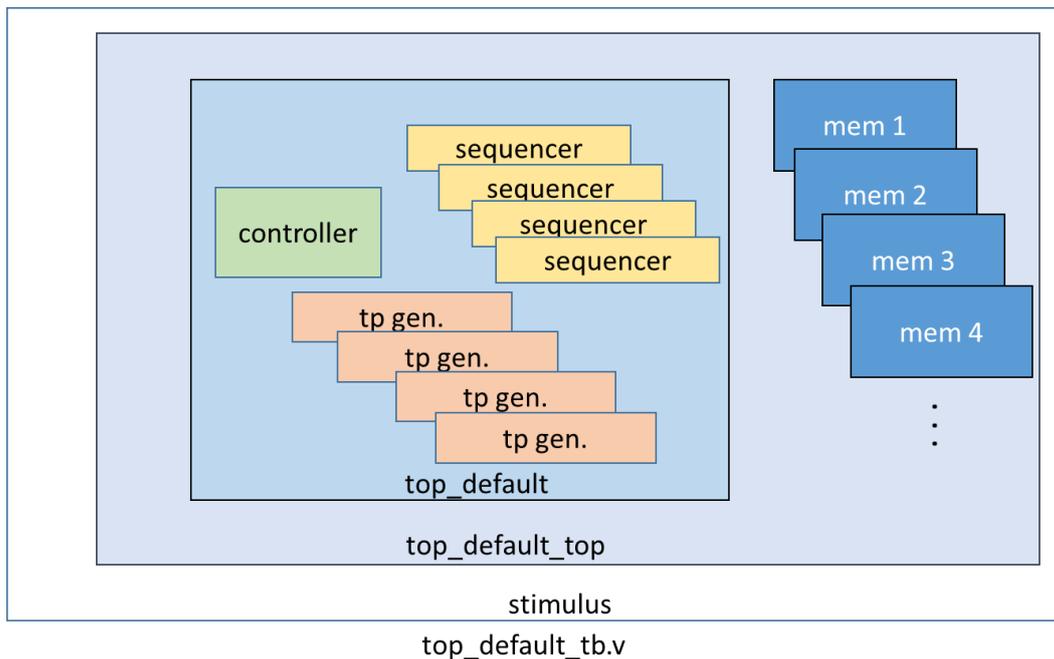


Figure 3-1 Testbench Architecture of Self-Simulation

If designer adjust clock domain, designer can check the difference of output file in *mbist* folder. In test case, the controller name of default clock domain is *top_default*.

Command for self-simulation is:

```
unix% make top_default FUNC=tb
```

If see time out message, “*Simulation time-out!*”, during self-simulation. Designer can modify the delay parameter of block “*ifndef FAULT*” in *top_default_tb.v* file as Figure 3-2 showed.

This delay parameter is generated by START and design for preventing infinite-loop situation. Figure 3-3 shows simulation results of self-simulation.

Note: if design includes ROM memory inside. Please check the path setting of ROM code file before executing simulation.

```

`ifndef FAULT
  initial begin
    #(cyc*414611);

    $display("\nSimulation time-out!\n");
    $finish;
  end
`endif

```

Figure 3-2 Delay Parameter Setting

```

Writing initial simulation snapshot: worklib.stimulus.v
Loading snapshot worklib.stimulus.v ..... Done
*Verdi3* Loading libsscore_ius142.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/IUS/cur/tools/inca/files/ncsimrc
ncsim> run

Test Result: Pass!

Simulation complete via $finish(1) at time 39383600 NS + 0
./top_default_tb.v:154 $finish;
ncsim> exit
make[1]: Leaving directory `/home/jeremy/TestCase/NDACase/mbist'

```

Figure 3-3 Self-Simulation Result

3.2 Inserted-Simulation

Figure 3-4 shows the architecture of Testbench for inserted-simulation. This inserted-simulation verify function correctness of inserted design which combines BIST circuits and user's system design. The simulation environment is built by "make" language. Designer can refer to **Makefile.top_default** file. This file defines commands and parameters for executing simulation.

If wants to debug with waveform file, designer can turn on related dump parameters in **top_default_INS_FAULT.f** file.

The same as self-simulation, if design has several clock domains, each clock domain should be passed when doing inserted-simulation.

Command of inserted-simulation is:

```
unix% make top_default FUNC=tb_INS
```

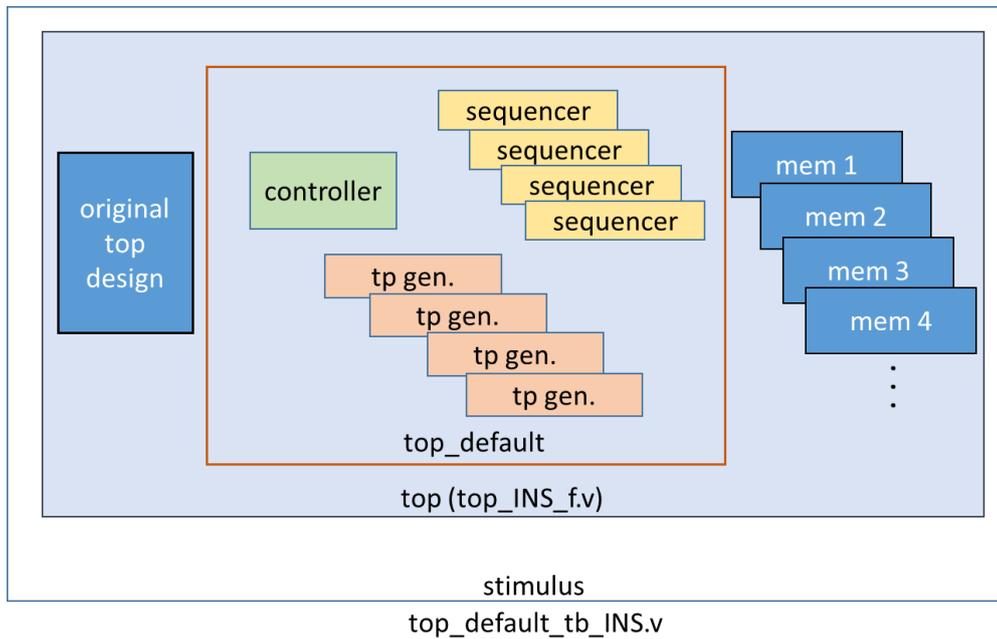


Figure 3-4 Testbench Architecture of Insert-Simulation

If shows time out message, **“Simulation time-out!”**, during executing simulation. designer can modify the delay parameter of block **“ifndef FAULT”** in **top_default_tb_INS.v**.

Figure 3-5 shows simulation prompt of inserted-simulation.

```

Loading snapshot worklib.stimulus:v ..... Done
*Verdi3* Loading libsscore_ius142.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/IUS/cur/tools/inca/files/ncsimrc
ncsim> run

Test Result: Pass!

Simulation complete via $finish(1) at time 39383600 NS + 0
./top_default_tb_INS.v:188 $finish;
ncsim> exit
make[1]: Leaving directory `/home/jeremy/TestCase/NDAcase/mbist'
    
```

Figure 3-5 Inserted-Simulation Result

3.3 Simulation with Repair Function

Designer can do inserted-simulation with repair function when repair mode is enabled. The simulation environment is built by **“make”** language. Designer can refer to **Makefile.RP_default** file. This file defines commands and parameters for executing simulation.

If debugging with waveform file. Designer can turn on related dump parameters in **RP_default_INS_FAULT.f** file.

The same as general inserted-simulation, if design has several clock domains, each clock domain should be passed when doing inserted-simulation.

Command of inserted-simulation is:

```
unix% make RP_default FUNC=tb_INS_RP
```

3.4 Simulation with Fault Memory Models

START can generate fault memory models for verifying functional correctness of BIST circuits. These fault memory models are generated automatically by START tool. Designer can find these models in **FAULT_MEMORY** directory.

Designer executes simulation with these models by using command below. These operations will use **fault_memory.f** in **FAULT_MEMORY** folder.

For self-simulation:

```
unix% make top_default FUNC=tb_f
```

For inserted-simulation

```
unix% make top_default FUNC=tb_INS_f
```

When executing this kind of simulation, it will show fail prompt. This is caused by pre-defined error bits in fault memory models. Designer can observe simulation waveform to understand the behavior of START BIST designs and fault memory models.

Figure 3-6 shows an example for running simulation with fault memory models. In this case, designer can find the access sequence of memories in group 1 (1_1_8, *sram_sp_640x32* memory model).

- A. Write access with data 32'hfff_ffff to address 10'h350
- B. Read access from address 10'h350
- C. Read data is 32'hfffd_ffff

The data of reading does not equal data of writing in A and this wrong behavior brought on simulation fail.

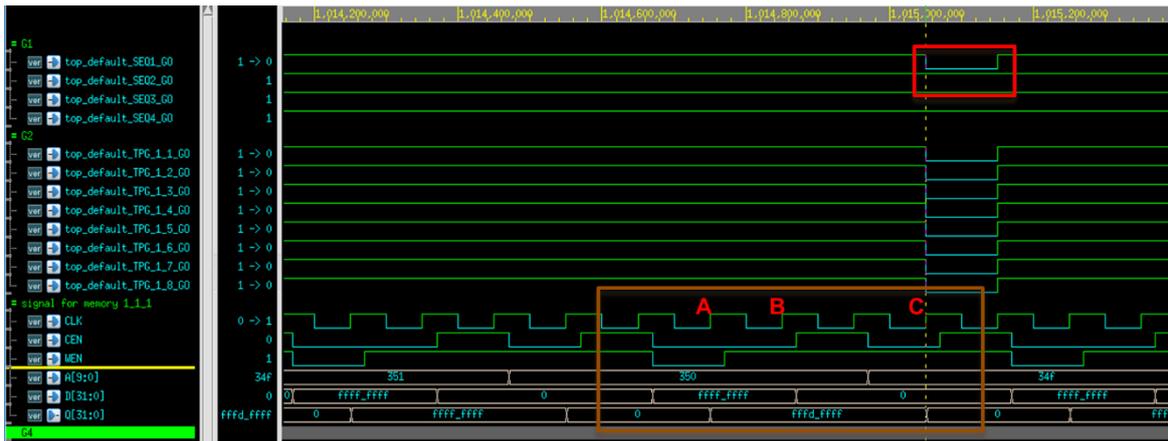


Figure 3-6 simulation waveform of fault memory models

Designer can find pre-defined error bits in fault memory models. Figure 3-7 is an example of *sram_sp_1024x32* memory model in **FAULT_MEMORY** directory.

```

module sram_sp_1024x32_f(
    Q,
    CLK,
    CEN,
    WEN,
    A,
    D,
    EMA,
    RETN
);
    integer _addr;
    parameter _BITS = 32;
    parameter _sa_fault = 1'b0; // sa0
    parameter _faulty_bit = 17;
    parameter _faulty_addr = 10'h350;

```

Figure 3-7 example of error bit definitions

Chapter 4 Synthesis

4.1 Synthesis

START also provides synthesis script for BIST circuits. Users can find it in output directory, named [design_name].tcl. Before executing synthesis, designer should build up related settings including library path, standard cell type and path of memory library file. If design has different clock domain, each clock domain should do synthesis task.

START provides a reference synthesis script in **mbist** folder. Command of synthesis is:

```
unix% make top_default FUNC=dc
```

Figure 4-1 shows the prompt during executing synthesis command. Designer can find synthesis results including area and timing reports in **REPORT** folder after synthesis is done.

```
write_sdc ${WORK_PATH}/${TOP}_netlist.sdc
1
#/****** worst case timing report *****/
redirect ${WORK_PATH}/REPORT/${TOP}_maxtiming.rpt { report_timing -nets -delay max -max_paths 5 -transition_time -nosplit }
redirect ${WORK_PATH}/REPORT/${TOP}_mintiming.rpt { report_timing -nets -delay min -max_paths 5 -transition_time -nosplit }
redirect ${WORK_PATH}/REPORT/${TOP}_looptim.rpt { report_timing -loops -max_paths 5 }
#/****** area report *****/
redirect ${WORK_PATH}/REPORT/${TOP}_area.rpt { report_area -hier }
redirect -append ${WORK_PATH}/REPORT/${TOP}_area.rpt { report_reference }
redirect ${WORK_PATH}/REPORT/${TOP}_power.rpt {report_power}
redirect ${WORK_PATH}/REPORT/${TOP}_qor.rpt {report_qor}
#/****** write_script *****/
write_script -output ${WORK_PATH}/REPORT/${TOP}_scrip.rpt
1
#/****** all violation *****/
redirect ${WORK_PATH}/REPORT/${TOP}_constraint.rpt { report_constraint -all_violators -verbose -nosplit }
#/****** check design *****/
#redirect ${WORK_PATH}/REPORT/${TOP}_check_design.rpt { check_design }
#/****** check test *****/
#redirect ${WORK_PATH}/REPORT/${TOP}_check_test.rpt { check_design }
#/****** check timing *****/
#redirect ${WORK_PATH}/REPORT/${TOP}_check_timing.rpt { check_timing }
exit

Memory usage for main task 226 Mbytes.
Memory usage for this session 226 Mbytes.
CPU usage for this session 13 seconds ( 0.00 hours ).

Thank you...
make[1]: Leaving directory `/mnt/raid/home/jeremy/LAB/NDACase/work'
```

Figure 4-1 Synthesis Output of top_default Controller

Appendix A

This appendix will introduce how to do memory checking with START --memchecker option. This can make sure whether customer's memory models can be recognized by START tool.

A.1 Memory Check with START memchecker

START assist to identify memory macros in customer's design by executing *memchecker* command. Here is an example to identify memories and output results into *memck* folder.

```
unix% cd NDAcase
unix% start --memchecker -N -W memck -v run.f
```

Designer can also identify memories in *memory* folder directly.

```
unix% cd NDAcase/memory
unix% start --memchecker -N -W memck -v *.v
```

Figure A-1 shows the output message of *memchecker* command.

```
Input file(s):
[1] /home/jeremy/LAB_e/NDAcase/memory/rom_6144_64.v
[2] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_4096x64.v
[3] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_640x32.v
[4] /home/jeremy/LAB_e/NDAcase/memory/rf_sp_128x22.v
[5] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_72x14.v
[6] /home/jeremy/LAB_e/NDAcase/top.v
[7] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_2048x64.v
[8] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_64x64.v
[9] /home/jeremy/LAB_e/NDAcase/memory/sram_dp_1024x64.v
[10] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_24x28.v
[11] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_24x56.v
[12] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_1024x32.v

Valid file(s):
[1] /home/jeremy/LAB_e/NDAcase/memory/rom_6144_64.v
[2] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_4096x64.v
[3] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_640x32.v
[4] /home/jeremy/LAB_e/NDAcase/memory/rf_sp_128x22.v
[5] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_72x14.v
[6] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_2048x64.v
[7] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_64x64.v
[8] /home/jeremy/LAB_e/NDAcase/memory/sram_dp_1024x64.v
[9] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_24x28.v
[10] /home/jeremy/LAB_e/NDAcase/memory/rf_2p_24x56.v
[11] /home/jeremy/LAB_e/NDAcase/memory/sram_sp_1024x32.v

Unrecognized file(s):
[1] /home/jeremy/LAB_e/NDAcase/top.v
```

Figure A-1 memchecker Information