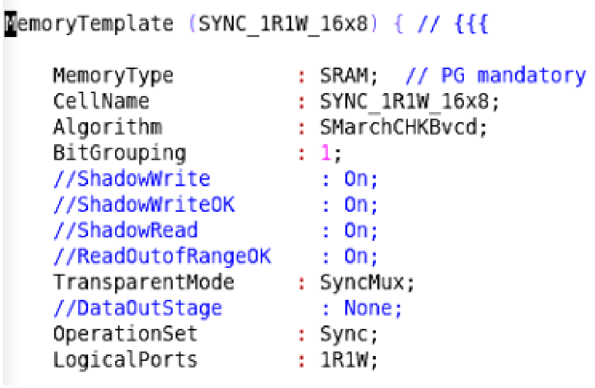
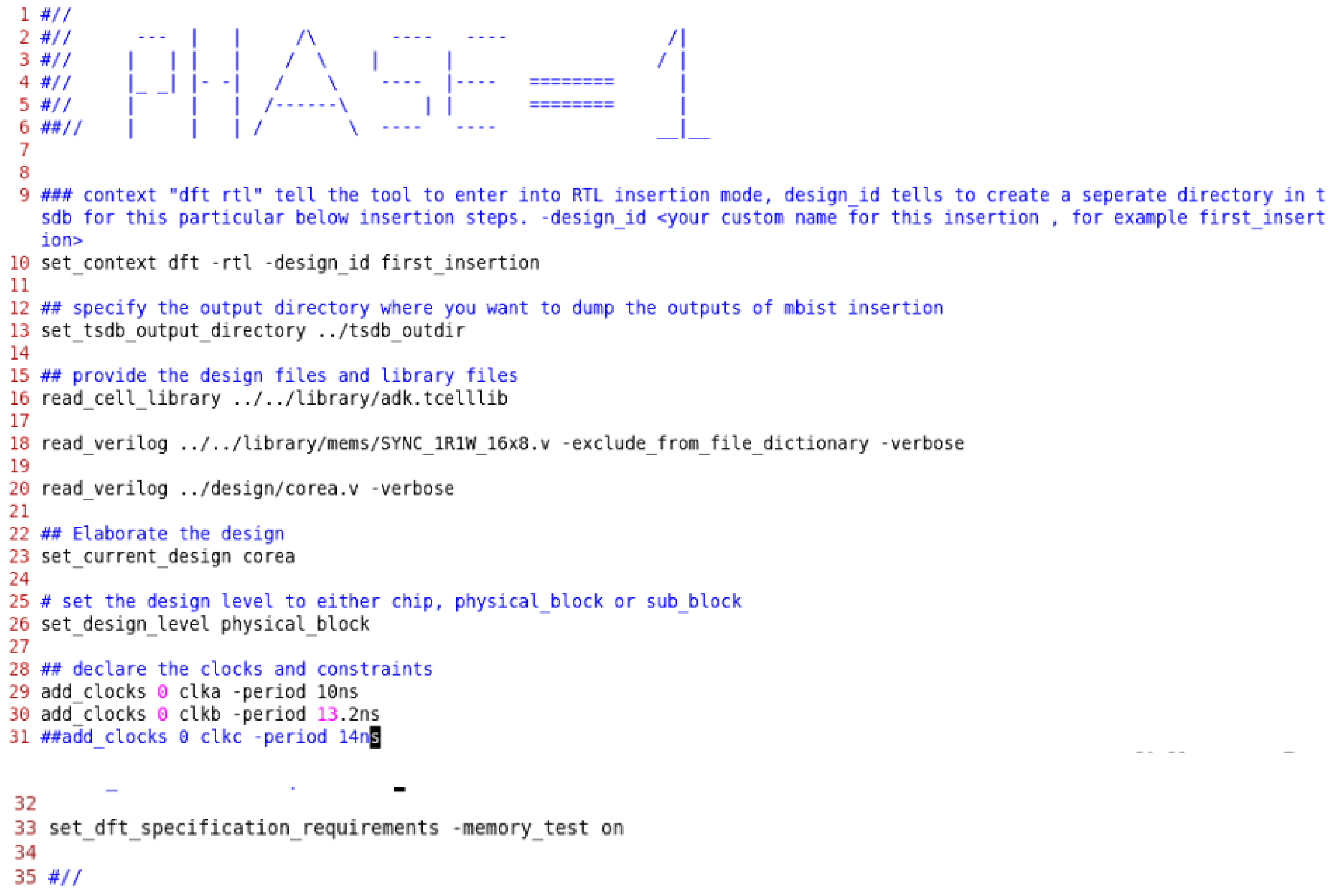
Lab 0 (It Contains COREA Design)

<u>Implementation of MBIST, Synthesis, Scan Insertion (Without Wrappers), ATPG & Simulations</u>



```
MemoryTemplate (SYNC_1R1W_16x8) { // {{{

    MemoryType            : SRAM;  // PG mandatory
    CellName              : SYNC_1R1W_16x8;
    Algorithm             : SMarchCHKBvcd;
    BitGrouping           : 1;
    //ShadowWrite           : On;
    //ShadowWriteOK         : On;
    //ShadowRead            : On;
    //ReadOutofRangeOK      : On;
    TransparentMode       : SyncMux;
    //DataOutStage          : None;
    OperationSet          : Sync;
    LogicalPorts          : 1R1W;
```

Sync Mux means adding a logic to bypass the memory
So memory has to be bypass during Scan, ATPG, EDT and tool will add a logic to bypass the memories



```
 1 #//
 2 #//     ---  |   |     /\       ----   ----          /|
 3 #//    |   | |   |    /  \     |      |             / |
 4 #//    |_ _| |- -|   /    \     ----  |----  ========   |
 5 #//    |     |   |  /------\        | |      ========   |
 6 ##//   |     |   | /        \   ----   ----           __|__
 7
 8
 9 ### context "dft rtl" tell the tool to enter into RTL insertion mode, design_id tells to create a seperate directory in t
   sdb for this particular below insertion steps. -design_id <your custom name for this insertion , for example first_insert
   ion>
10 set_context dft -rtl -design_id first_insertion
11
12 ## specify the output directory where you want to dump the outputs of mbist insertion
13 set_tsdb_output_directory ../tsdb_outdir
14
15 ## provide the design files and library files
16 read_cell_library ../../library/adk.tcelllib
17
18 read_verilog ../../library/mems/SYNC_1R1W_16x8.v -exclude_from_file_dictionary -verbose
19
20 read_verilog ../design/corea.v -verbose
21
22 ## Elaborate the design
23 set_current_design corea
24
25 # set the design level to either chip, physical_block or sub_block
26 set_design_level physical_block
27
28 ## declare the clocks and constraints
29 add_clocks 0 clka -period 10ns
30 add_clocks 0 clkb -period 13.2ns
31 ##add_clocks 0 clkc -period 14ns

32
33 set_dft_specification_requirements -memory_test on
34
35 #//
```

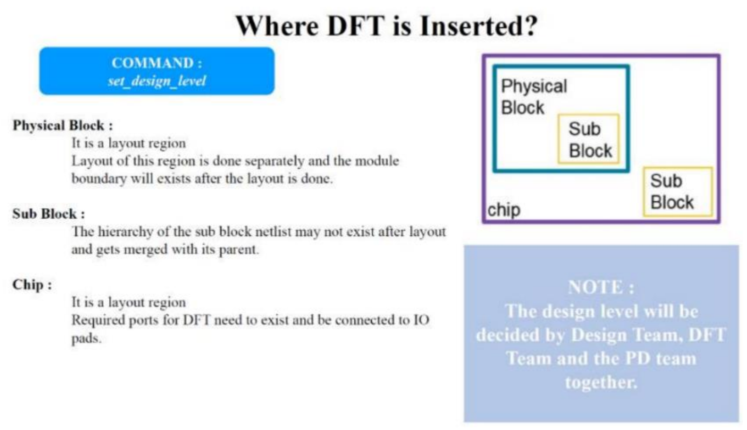
Line 10: Inside the tsdb output directory a sub-directory with this design id will be created and all the outputs with this particular run it will be stored into that particular directory

Line13: By default it will be created in current directory, so our current directory is MBIST Insertion. But for complete lab0 to have common tsdb output directory

Line 16,18,20: Reading the library files, the memory file and the corea we are reading.

Line 23: Setting the current design of corea and it will do elaboration, it will check whether all the module definition is available or not and whether module instantiation is correct or not.

Line 26: set_design_level physical_block



Where DFT is Inserted?

COMMAND :
set_design_level

Physical Block :
It is a layout region
Layout of this region is done separately and the module boundary will exists after the layout is done.

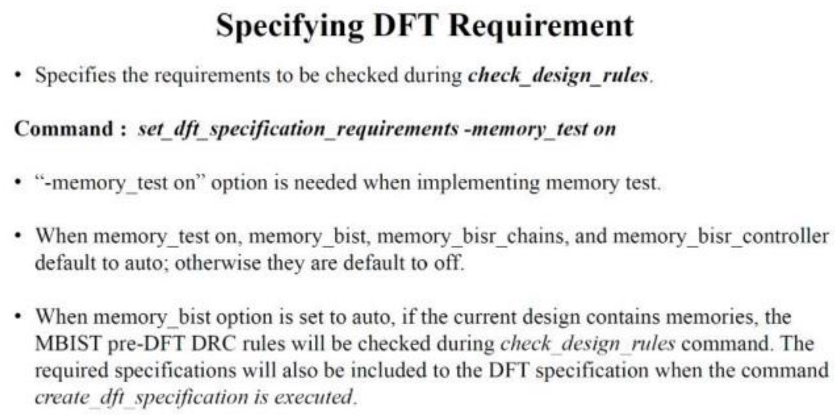
Sub Block :
The hierarchy of the sub block netlist may not exist after layout and gets merged with its parent.

Chip :
It is a layout region
Required ports for DFT need to exist and be connected to IO pads.

NOTE :
The design level will be decided by Design Team, DFT Team and the PD team together.

Line 33: -memory_test on means we are doing MBIST Insertion and if we -logic_test on means we are doing EDT and OCC Insertion.

Whatever the test hardware has inserted into MBIST insertion run that will be included into the dft specification

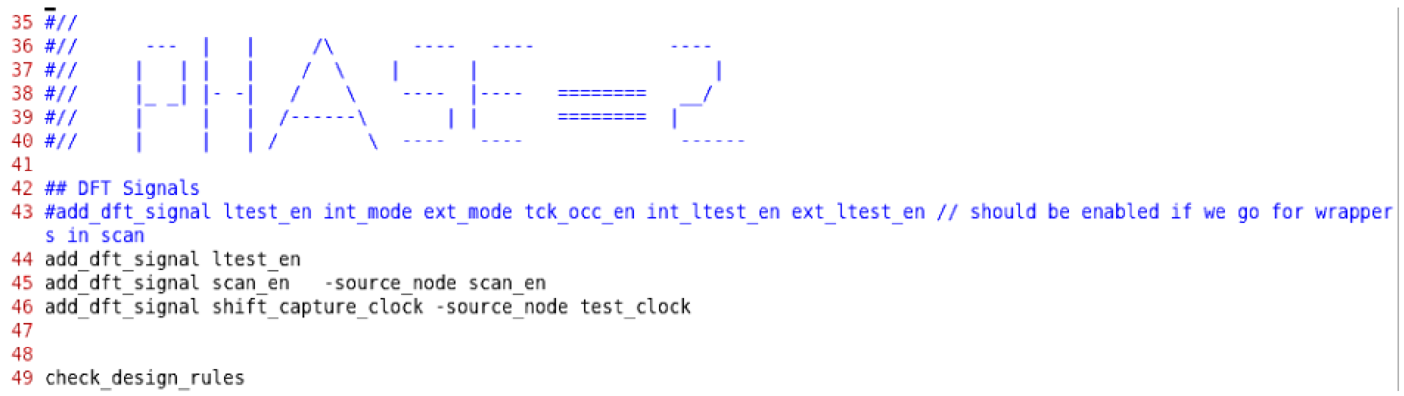


Specifying DFT Requirement

- Specifies the requirements to be checked during ***check_design_rules***.

Command : ***set_dft_specification_requirements -memory_test on***

- "-memory_test on" option is needed when implementing memory test.
- When memory_test on, memory_bist, memory_bisr_chains, and memory_bisr_controller default to auto; otherwise they are default to off.
- When memory_bist option is set to auto, if the current design contains memories, the MBIST pre-DFT DRC rules will be checked during *check_design_rules* command. The required specifications will also be included to the DFT specification when the command *create_dft_specification is executed.*



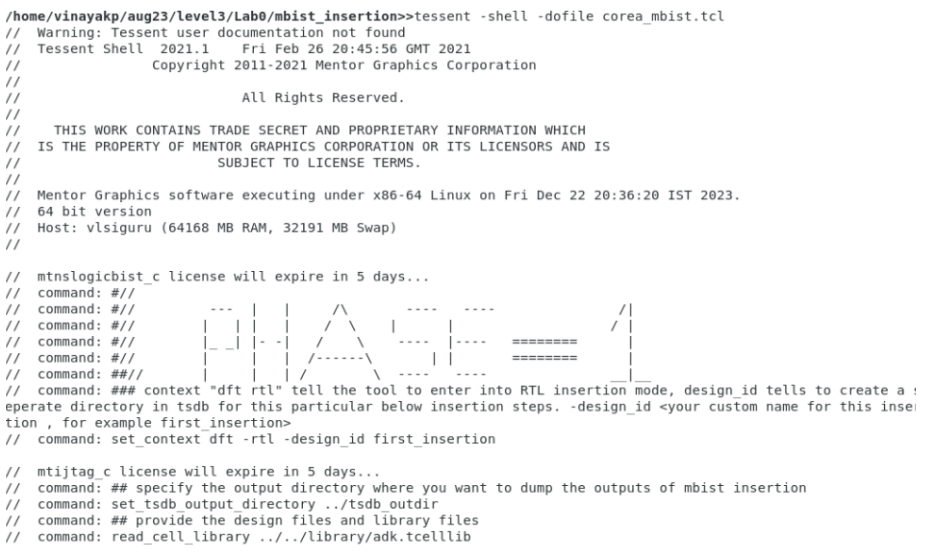
```
35 #//
36 #//      ---  |   |     /\       ----   ----            ----
37 #//     |   | |   |    /  \    |      |                     |
38 #//     |_ _| |- -|   /    \    ----  |----   ========   __/
39 #//     |     |   |  /------\       | |       ========  |
40 #//     |     |   | /        \  ----   ----              ------
41
42 ## DFT Signals
43 #add_dft_signal ltest_en int_mode ext_mode tck_occ_en int_ltest_en ext_ltest_en // should be enabled if we go for wrapper
   s in scan
44 add_dft_signal ltest_en
45 add_dft_signal scan_en   -source_node scan_en
46 add_dft_signal shift_capture_clock -source_node test_clock
47
48
49 check_design_rules
```

Line 44: Only static signals can be controlled by TDR

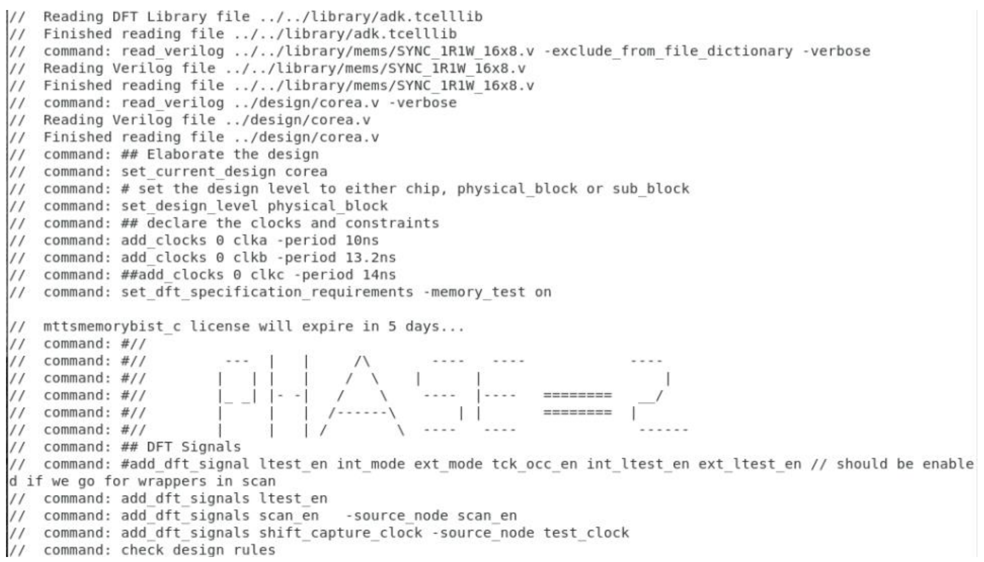
Line 45: Scan Enable will be controlled by the top level ports scan enable.
Source node that is scan_en will be controlled by top level port i.e. scan_en

Line 46: Shift capture clock it will be controlled by the top level port test clock

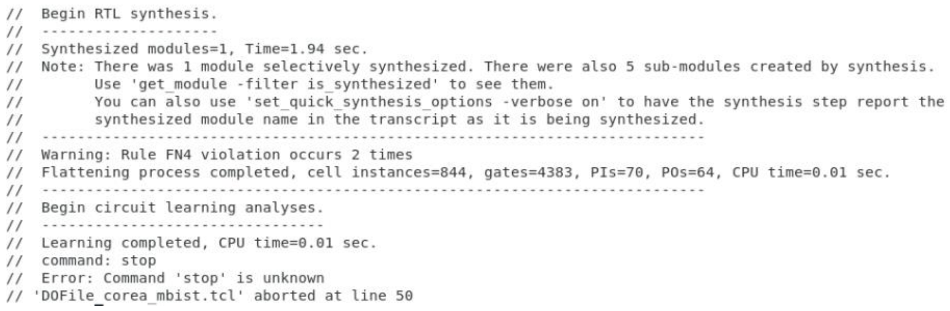
So we will run till line 49 in tessent tool



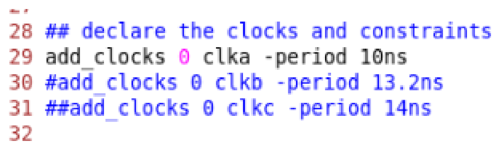
```
/home/vinayakp/aug23/level3/Lab0/mbist_insertion>>tessent -shell -dofile corea_mbist.tcl
//  Warning: Tessent user documentation not found
//  Tessent Shell  2021.1    Fri Feb 26 20:45:56 GMT 2021
//               Copyright 2011-2021 Mentor Graphics Corporation
//
//                           All Rights Reserved.
//
//    THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION WHICH
//  IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS AND IS
//                        SUBJECT TO LICENSE TERMS.
//
//  Mentor Graphics software executing under x86-64 Linux on Fri Dec 22 20:36:20 IST 2023.
//  64 bit version
//  Host: vlsiguru (64168 MB RAM, 32191 MB Swap)
//

//  mtnslogicbist_c license will expire in 5 days...
//  command: #//
//  command: #//         ---  |   |     /\       ----   ----              /|
//  command: #//        |   | |   |    /  \    |      |                 / |
//  command: #//        |_ _| |- -|   /    \    ----  |----   ========     |
//  command: #//        |     |   |  /------\       | |       ========     |
//  command: ##//       |     |   | /        \  ----   ----              __|__
//  command: ### context "dft rtl" tell the tool to enter into RTL insertion mode, design_id tells to create a s
eperate directory in tsdb for this particular below insertion steps. -design_id <your custom name for this inser
tion , for example first_insertion>
//  command: set_context dft -rtl -design_id first_insertion

//  mtijtag_c license will expire in 5 days...
//  command: ## specify the output directory where you want to dump the outputs of mbist insertion
//  command: set_tsdb_output_directory ../tsdb_outdir
//  command: ## provide the design files and library files
//  command: read_cell_library ../../library/adk.tcelllib
```



```
//  Reading DFT Library file ../../library/adk.tcelllib
//  Finished reading file ../../library/adk.tcelllib
//  command: read_verilog ../../library/mems/SYNC_1R1W_16x8.v -exclude_from_file_dictionary -verbose
//  Reading Verilog file ../../library/mems/SYNC_1R1W_16x8.v
//  Finished reading file ../../library/mems/SYNC_1R1W_16x8.v
//  command: read_verilog ../design/corea.v -verbose
//  Reading Verilog file ../design/corea.v
//  Finished reading file ../design/corea.v
//  command: ## Elaborate the design
//  command: set_current_design corea
//  command: # set the design level to either chip, physical_block or sub_block
//  command: set_design_level physical_block
//  command: ## declare the clocks and constraints
//  command: add_clocks 0 clka -period 10ns
//  command: add_clocks 0 clkb -period 13.2ns
//  command: ##add_clocks 0 clkc -period 14ns
//  command: set_dft_specification_requirements -memory_test on

//  mttsmemorybist_c license will expire in 5 days...
//  command: #//
//  command: #//         ---  |   |     /\       ----   ----            ----
//  command: #//        |   | |   |    /  \    |      |                     |
//  command: #//        |_ _| |- -|   /    \    ----  |----   ========   __/
//  command: #//        |     |   |  /------\       | |       ========  |
//  command: #//        |     |   | /        \  ----   ----              ------
//  command: ## DFT Signals
//  command: #add_dft_signal ltest_en int_mode ext_mode tck_occ_en int_ltest_en ext_ltest_en // should be enable
d if we go for wrappers in scan
//  command: add_dft_signals ltest_en
//  command: add_dft_signals scan_en   -source_node scan_en
//  command: add_dft_signals shift_capture_clock -source_node test_clock
//  command: check design rules
```

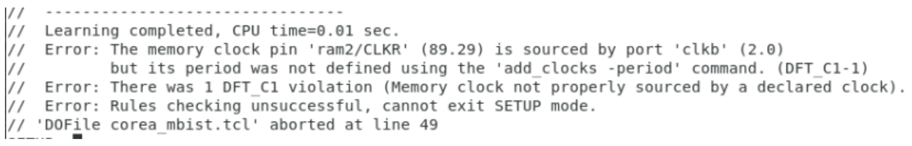


```
//  Begin RTL synthesis.
//  --------------------
//  Synthesized modules=1, Time=1.94 sec.
//  Note: There was 1 module selectively synthesized. There were also 5 sub-modules created by synthesis.
//        Use 'get_module -filter is_synthesized' to see them.
//        You can also use 'set_quick_synthesis_options -verbose on' to have the synthesis step report the
//        synthesized module name in the transcript as it is being synthesized.
//  ------------------------------------------------------------------------
//  Warning: Rule FN4 violation occurs 2 times
//  Flattening process completed, cell instances=844, gates=4383, PIs=70, POs=64, CPU time=0.01 sec.
//  ------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  --------------------------------
//  Learning completed, CPU time=0.01 sec.
//  command: stop
//  Error: Command 'stop' is unknown
// 'DOFile_corea_mbist.tcl' aborted at line 50
```

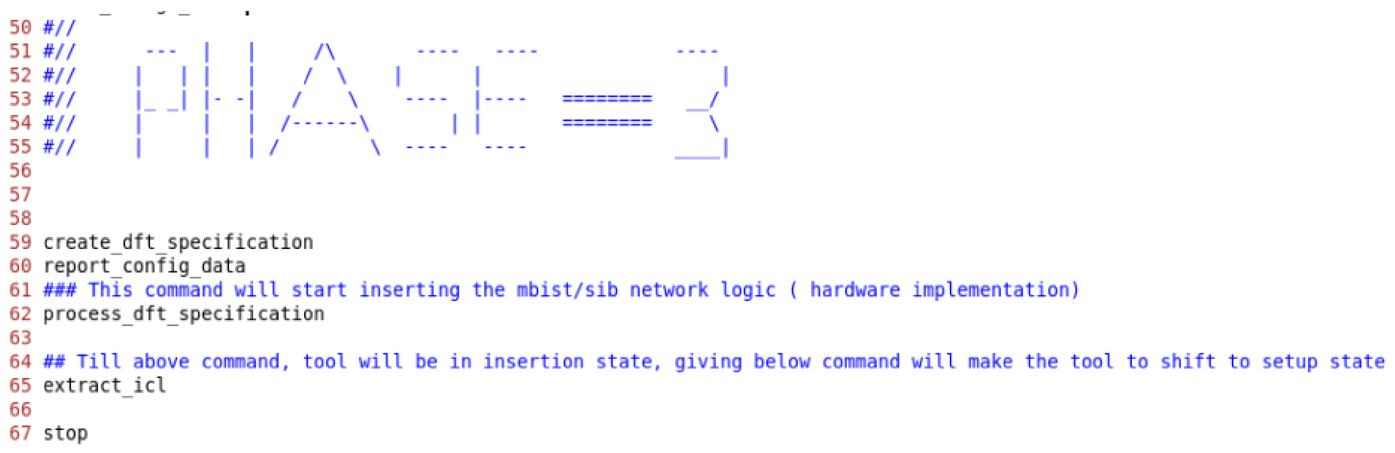


```
## declare the clocks and constraints
add_clocks 0 clka -period 10ns
#add_clocks 0 clkb -period 13.2ns
##add_clocks 0 clkc -period 14ns

```

If u have missed the clock declaration

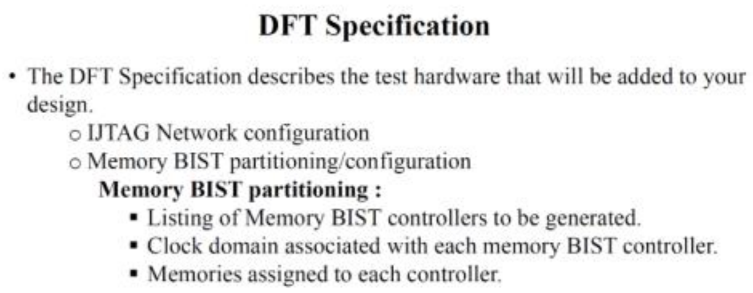


```
//  --------------------------------
//  Learning completed, CPU time=0.01 sec.
//  Error: The memory clock pin 'ram2/CLKR' (89.29) is sourced by port 'clkb' (2.0)
//         but its period was not defined using the 'add_clocks -period' command. (DFT_C1-1)
//  Error: There was 1 DFT_C1 violation (Memory clock not properly sourced by a declared clock).
//  Error: Rules checking unsuccessful, cannot exit SETUP mode.
// 'DOFile corea_mbist.tcl' aborted at line 49
```



```
50 #//
51 #//      ---  |   |     /\       ----   ----            ----
52 #//     |   | |   |    /  \    |      |                     |
53 #//     |_ _| |- -|   /    \    ----  |----   ========   __/
54 #//     |     |   |  /------\       | |       ========      \
55 #//     |     |   | /        \  ----   ----             ____|
56
57
58
59 create_dft_specification
60 report_config_data
61 ### This command will start inserting the mbist/sib network logic ( hardware implementation)
62 process_dft_specification
63
64 ## Till above command, tool will be in insertion state, giving below command will make the tool to shift to setup state
65 extract_icl
66
67 stop
```

Line 59: creat_dft_specification

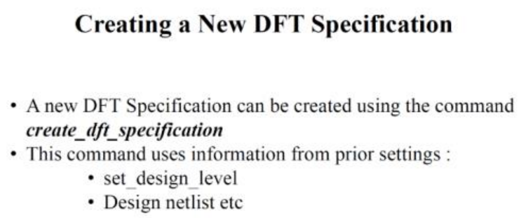


DFT Specification

- The DFT Specification describes the test hardware that will be added to your design.
  - IJTAG Network configuration
  - Memory BIST partitioning/configuration

    Memory BIST partitioning :
    - Listing of Memory BIST controllers to be generated.
    - Clock domain associated with each memory BIST controller.
    - Memories assigned to each controller.

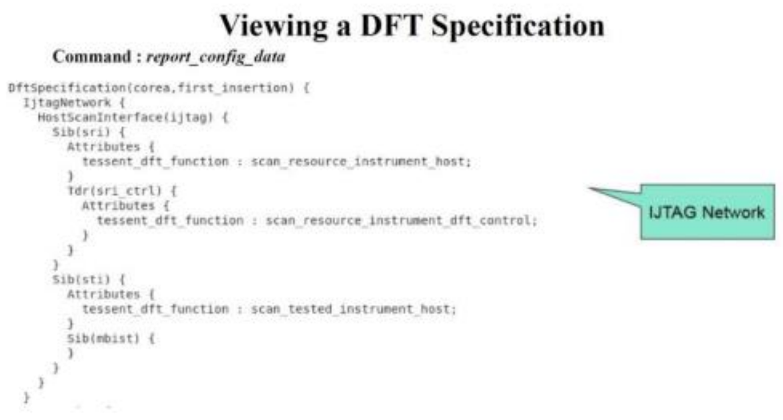
All the above information will be coming in dft specs.



Creating a New DFT Specification

- A new DFT Specification can be created using the command ***create_dft_specification***
- This command uses information from prior settings :
  - set_design_level
  - Design netlist etc

For example for IJTAG Network:-



Viewing a DFT Specification

Command : ***report_config_data***

```
DftSpecification(corea,first_insertion) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Tdr(sri_ctrl) {
          Attributes {
            tessent_dft_function : scan_resource_instrument_dft_control;
          }
        }
      }
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
      }
    }
  }
```

So in IJTAG network 3 SIBS will be created and 1 TDR SRI Control

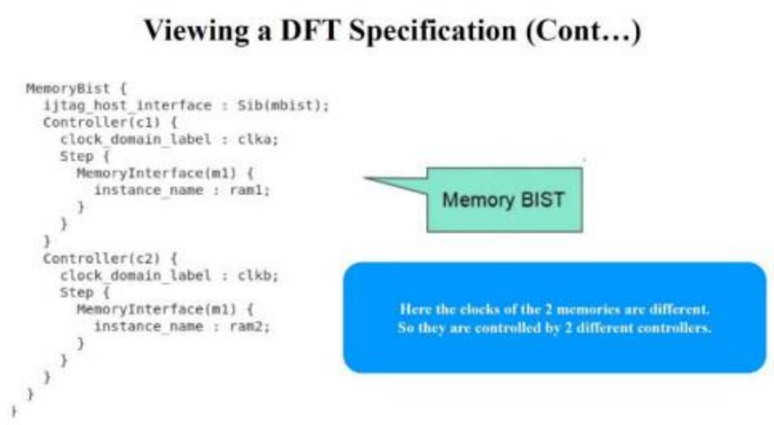
Inside 3 SIBS

- SIB STI (Scan Tested Instrument)
- SIB SRI (Scan Resource Instrument)

- SIB MBIST - It will only controls the MBIST Network

TDR SRI control
- This TDR (As of now it will controlling the once signal that is entest enable signal) but later in lab3 we will insert some more static signals like memory bypass enable, TCK OCC Enable, MBIST enable,

This are the things will be inserted during IJTAG Network

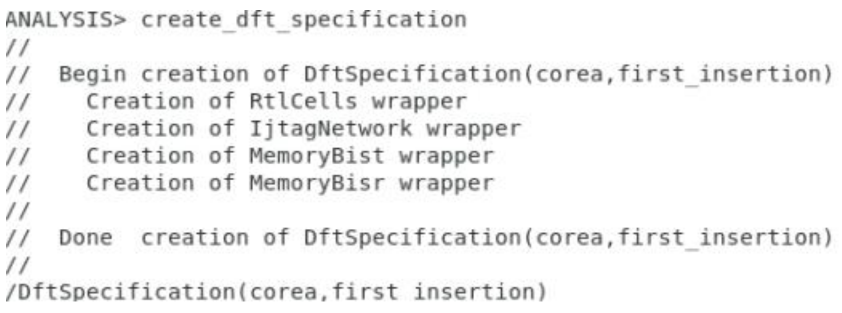


Viewing a DFT Specification (Cont…)

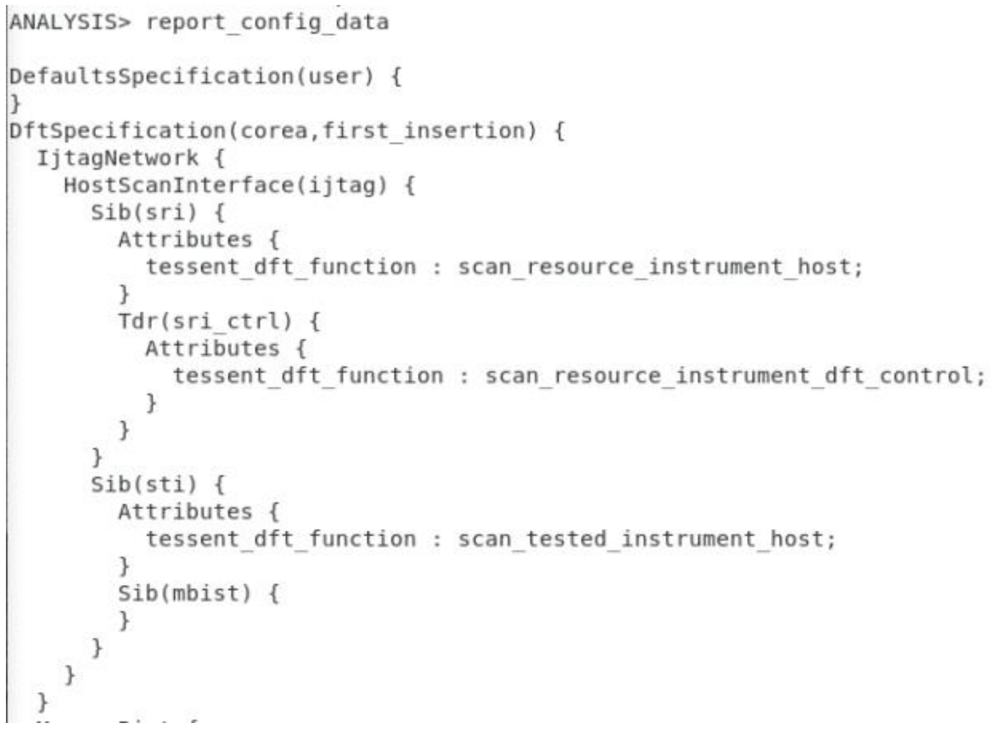
```
MemoryBist {
  ijtag_host_interface : Sib(mbist);
  Controller(c1) {
    clock_domain_label : clka;
    Step {
      MemoryInterface(m1) {
        instance_name : ram1;
      }
    }
  }
  Controller(c2) {
    clock_domain_label : clkb;
    Step {
      MemoryInterface(m1) {
        instance_name : ram2;
      }
    }
  }
}
}
```

Memory BIST
Here the clocks of the 2 memories are different. So they are controlled by 2 different controllers.

In this above code clka is going to ram1 and clkb is going to ram2, that is memory grouping so because of both this stamps having 2 different clocks.
So, both this clocks it has 2 different clocks and 2 different controllers.



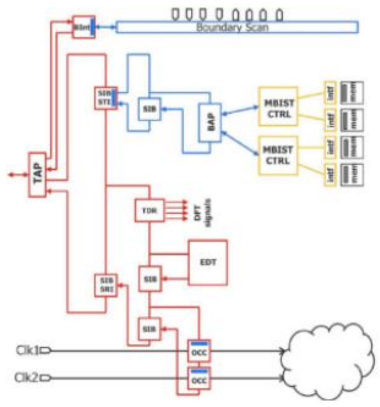
```
ANALYSIS> create_dft_specification
//
//  Begin creation of DftSpecification(corea,first_insertion)
//    Creation of RtlCells wrapper
//    Creation of IjtagNetwork wrapper
//    Creation of MemoryBist wrapper
//    Creation of MemoryBisr wrapper
//
//  Done  creation of DftSpecification(corea,first_insertion)
//
/DftSpecification(corea,first insertion)
```



```
ANALYSIS> report_config_data

DefaultsSpecification(user) {
}
DftSpecification(corea,first_insertion) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Tdr(sri_ctrl) {
          Attributes {
            tessent_dft_function : scan_resource_instrument_dft_control;
          }
        }
      }
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(mbist) {
        }
      }
    }
  }
  MemoryBist {
    ijtag_host_interface : Sib(mbist);
    Controller(c1) {
      clock_domain_label : clka;
      Step {
        MemoryInterface(m1) {
          instance_name : ram1;
        }
      }
    }
    Controller(c2) {
      clock_domain_label : clkb;
      Step {
        MemoryInterface(m1) {
          instance_name : ram2;
        }
      }
    }
  }
}
```

This the Network which the tool will be created

Diagrammatic Representation of DFT Specification



- A SIB is a special node in IJTAG that acts as a switch.
- Instruments that needs to be active during scan (EDT OCC) are inserted under 1 SIB and the ones that are scan tested such as MBIST controller are inserted under another SIB.

Types of SIBS

Scan Tested Instrument (STI) : The SIB STI provides access to the IJTAG network for MBIST controller (in this fig).

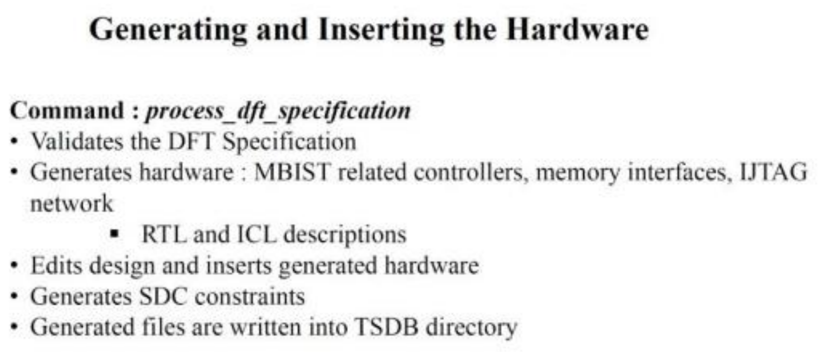
Scan Resource Instrument (SRI) : The SIB SRI provides access to the IJTAG network for logic instruments (EDT, OCC)

Difference between SIB - STI and SIB - SRI
And why are telling as a switch?

- Because It will decide whether to allow or not to allow the data to be return on to the TDR that's why SIB acts as a switch.

Instruments that need to be active during scan they are placed under one SIB and the instruments that has to be scan tested that is MBIST Controller It will be scanned tested and they will be placed under another SIB.

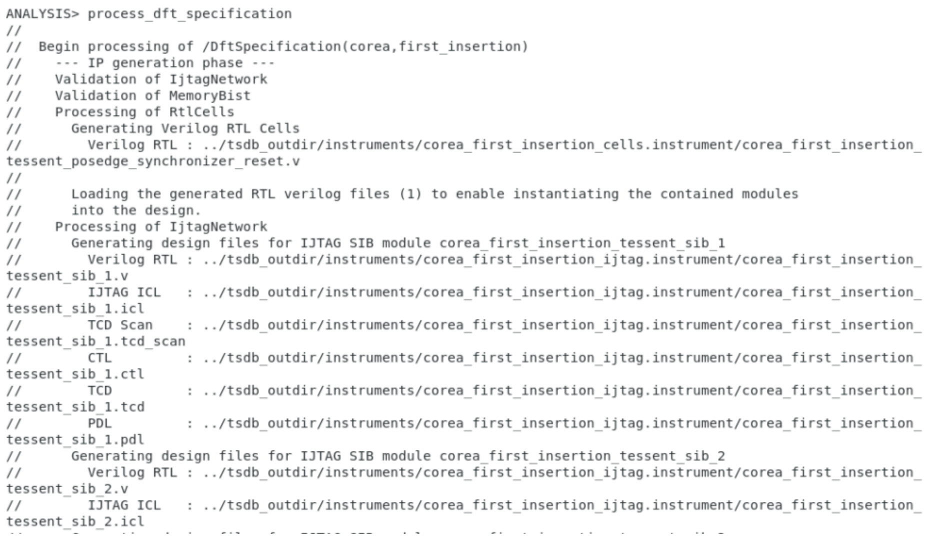
Line 62: It will validate whatever HW it has generated on top and it will inserted in our design and it will finally write out all the output files into the TSDB output directory.



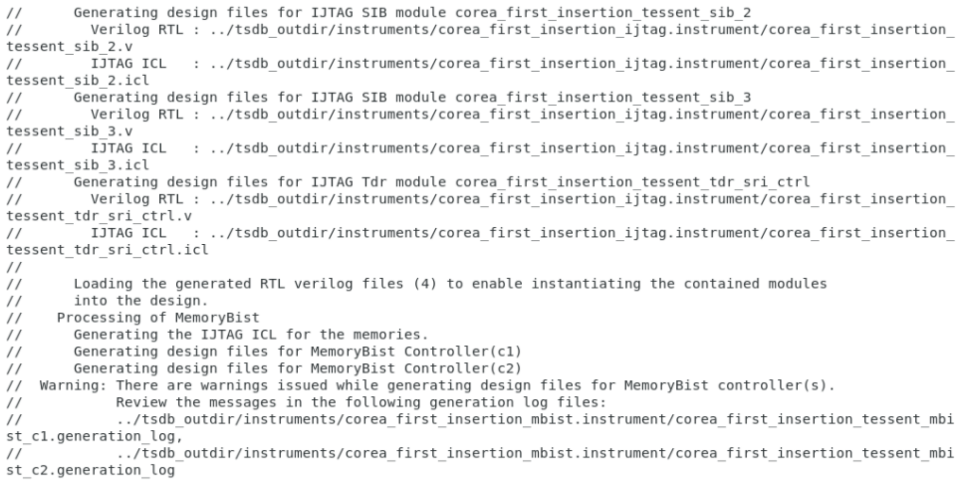
Generating and Inserting the Hardware

Command : *process_dft_specification*
- Validates the DFT Specification
- Generates hardware : MBIST related controllers, memory interfaces, IJTAG network
  - RTL and ICL descriptions
- Edits design and inserts generated hardware
- Generates SDC constraints
- Generated files are written into TSDB directory

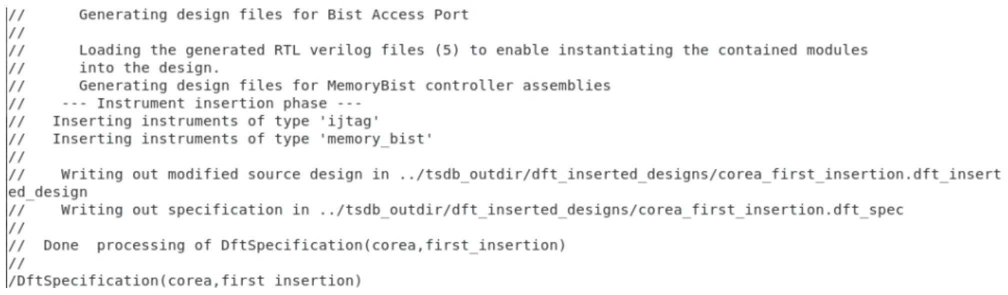
Line 65: extract_icl



```
ANALYSIS> process_dft_specification
//
//  Begin processing of /DftSpecification(corea,first_insertion)
//    --- IP generation phase ---
//    Validation of IjtagNetwork
//    Validation of MemoryBist
//    Processing of RtlCells
//      Generating Verilog RTL Cells
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_cells.instrument/corea_first_insertion_
tessent_posedge_synchronizer_reset.v
//
//      Loading the generated RTL verilog files (1) to enable instantiating the contained modules
//      into the design.
//    Processing of IjtagNetwork
//      Generating design files for IJTAG SIB module corea_first_insertion_tessent_sib_1
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.v
//        IJTAG ICL   : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.icl
//        TCD Scan    : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.tcd_scan
//        CTL         : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.ctl
//        TCD         : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.tcd
//        PDL         : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_1.pdl
//      Generating design files for IJTAG SIB module corea_first_insertion_tessent_sib_2
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_2.v
//        IJTAG ICL   : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_2.icl
```

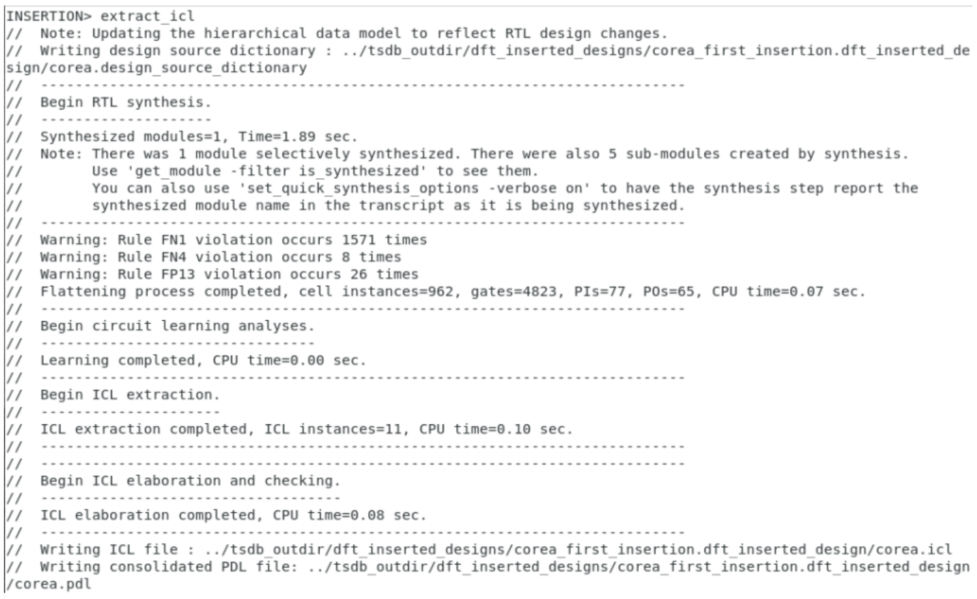


```
//      Generating design files for IJTAG SIB module corea_first_insertion_tessent_sib_2
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_2.v
//        IJTAG ICL   : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_2.icl
//      Generating design files for IJTAG SIB module corea_first_insertion_tessent_sib_3
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_3.v
//        IJTAG ICL   : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_sib_3.icl
//      Generating design files for IJTAG Tdr module corea_first_insertion_tessent_tdr_sri_ctrl
//        Verilog RTL : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_tdr_sri_ctrl.v
//        IJTAG ICL   : ../tsdb_outdir/instruments/corea_first_insertion_ijtag.instrument/corea_first_insertion_
tessent_tdr_sri_ctrl.icl
//
//      Loading the generated RTL verilog files (4) to enable instantiating the contained modules
//      into the design.
//    Processing of MemoryBist
//      Generating the IJTAG ICL for the memories.
//      Generating design files for MemoryBist Controller(c1)
//      Generating design files for MemoryBist Controller(c2)
//  Warning: There are warnings issued while generating design files for MemoryBist controller(s).
//           Review the messages in the following generation log files:
//           ../tsdb_outdir/instruments/corea_first_insertion_mbist.instrument/corea_first_insertion_tessent_mbi
st_c1.generation_log,
//           ../tsdb_outdir/instruments/corea_first_insertion_mbist.instrument/corea_first_insertion_tessent_mbi
st_c2.generation_log
```

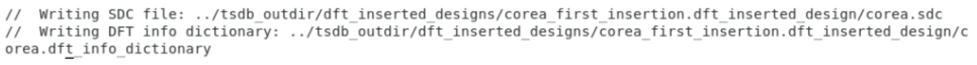


```
//      Generating design files for Bist Access Port
//
//      Loading the generated RTL verilog files (5) to enable instantiating the contained modules
//      into the design.
//      Generating design files for MemoryBist controller assemblies
//    --- Instrument insertion phase ---
//   Inserting instruments of type 'ijtag'
//   Inserting instruments of type 'memory_bist'
//
//    Writing out modified source design in ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_insert
ed_design
//    Writing out specification in ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_spec
//
//  Done  processing of DftSpecification(corea,first_insertion)
//
/DftSpecification(corea,first insertion)
```

So it was validating the design, whatever DT HW it has generated and it is inserting in our design and it is also writing all the output files into the tsdb output directory so individually for all the sibs, sri control tdr, verilog files and ICL description and even MBIST also, and control it will be creating individually all the three sibs.



```
INSERTION> extract_icl
//  Note: Updating the hierarchical data model to reflect RTL design changes.
//  Writing design source dictionary : ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_inserted_de
sign/corea.design_source_dictionary
//  ------------------------------------------------------------------------
//  Begin RTL synthesis.
//  -------------------
//  Synthesized modules=1, Time=1.89 sec.
//  Note: There was 1 module selectively synthesized. There were also 5 sub-modules created by synthesis.
//        Use 'get_module -filter is_synthesized' to see them.
//        You can also use 'set_quick_synthesis_options -verbose on' to have the synthesis step report the
//        synthesized module name in the transcript as it is being synthesized.
//  ------------------------------------------------------------------------
//  Warning: Rule FN1 violation occurs 1571 times
//  Warning: Rule FN4 violation occurs 8 times
//  Warning: Rule FP13 violation occurs 26 times
//  Flattening process completed, cell instances=962, gates=4823, PIs=77, POs=65, CPU time=0.07 sec.
//  ------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  -------------------------------
//  Learning completed, CPU time=0.00 sec.
//  ------------------------------------------------------------------------
//  Begin ICL extraction.
//  --------------------
//  ICL extraction completed, ICL instances=11, CPU time=0.10 sec.
//  ------------------------------------------------------------------------
//  ------------------------------------------------------------------------
//  Begin ICL elaboration and checking.
//  ----------------------------------
//  ICL elaboration completed, CPU time=0.08 sec.
//  ------------------------------------------------------------------------
//  Writing ICL file : ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_inserted_design/corea.icl
//  Writing consolidated PDL file: ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_inserted_design
/corea.pdl
```

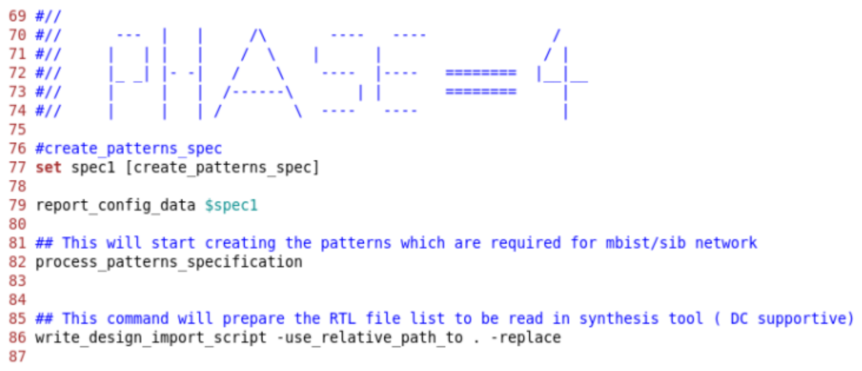


```
//  Writing SDC file: ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_inserted_design/corea.sdc
//  Writing DFT info dictionary: ../tsdb_outdir/dft_inserted_designs/corea_first_insertion.dft_inserted_design/c
orea.dft_info_dictionary
```

So the ICL, PDL, SDC File all these files will be created.

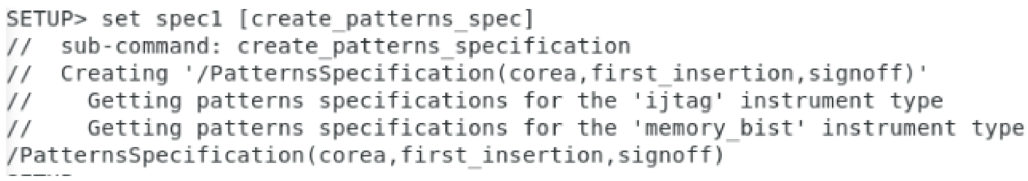
It will be check in the tsdb output directory

So, corea files will be created.



```
69 #//
70 #//      ---  |   |     /\       ----   ----              /
71 #//     |   | |   |    /  \    |      |                  / |
72 #//     |_ _| |- -|   /    \    ----  |----   ========  |__|__
73 #//     |     |   |  /------\       | |       ========     |
74 #//     |     |   | /        \  ----   ----                |
75
76 #create_patterns_spec
77 set spec1 [create_patterns_spec]
78
79 report_config_data $spec1
80
81 ## This will start creating the patterns which are required for mbist/sib network
82 process_patterns_specification
83
84
85 ## This command will prepare the RTL file list to be read in synthesis tool ( DC supportive)
86 write_design_import_script -use_relative_path_to . -replace
87
```

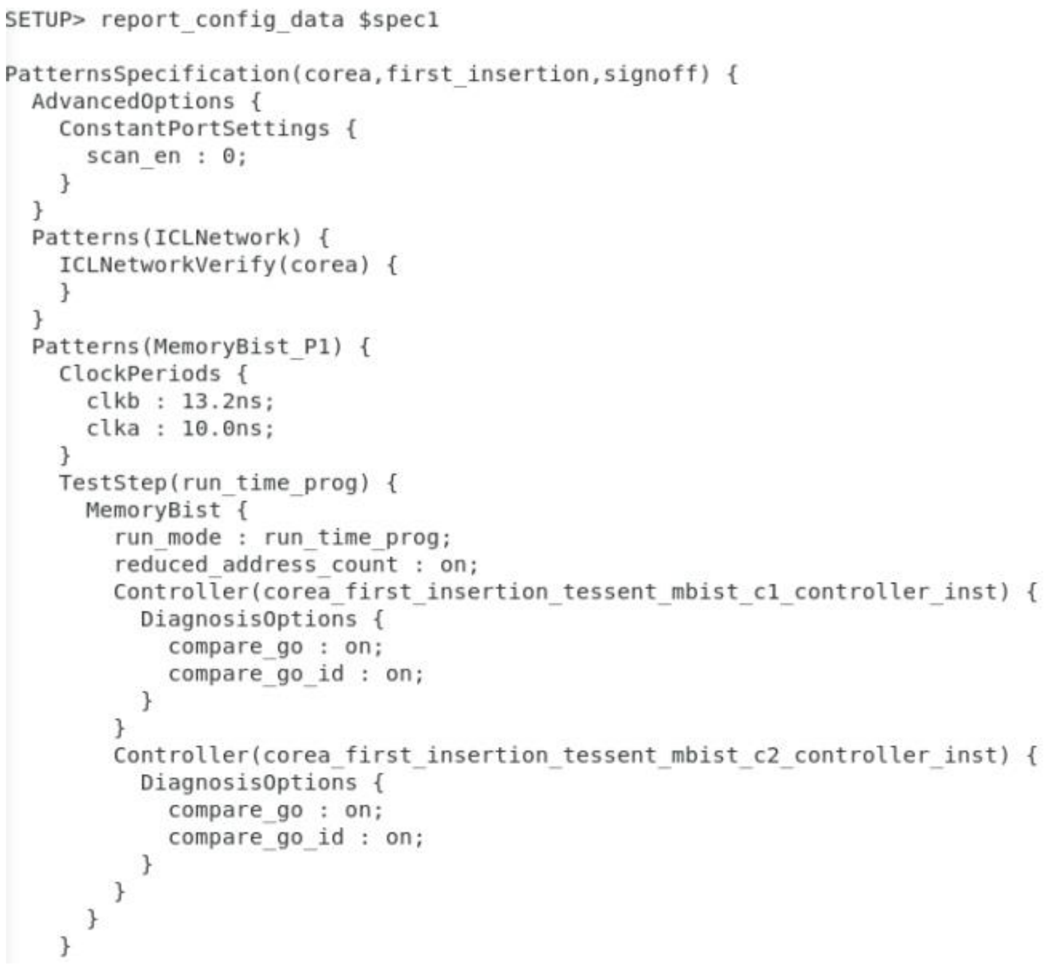
Line 77: It will create the patterns specs. And what is pattern specs?
It is similar to TCL command.



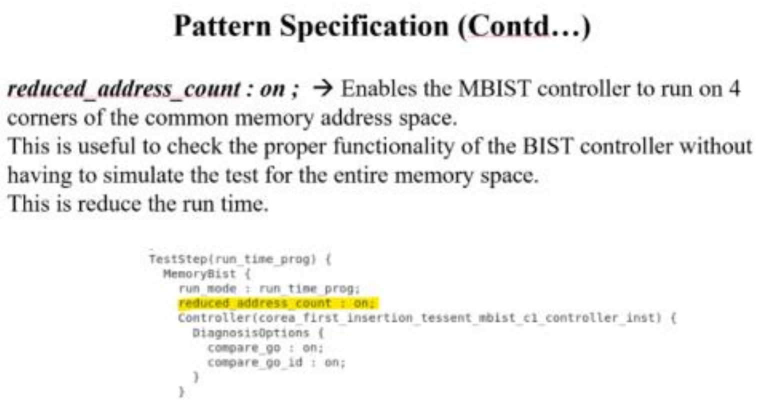
```
SETUP> set spec1 [create_patterns_spec]
//  sub-command: create_patterns_specification
//  Creating '/PatternsSpecification(corea,first_insertion,signoff)'
//    Getting patterns specifications for the 'ijtag' instrument type
//    Getting patterns specifications for the 'memory_bist' instrument type
/PatternsSpecification(corea,first_insertion,signoff)
```

Line 79: Whatever it store into spec1 variable that we will able to see.

$spec1 means what all are the dollar specifications which has created that we will get in this terminal.



```
SETUP> report_config_data $spec1

PatternsSpecification(corea,first_insertion,signoff) {
  AdvancedOptions {
    ConstantPortSettings {
      scan_en : 0;
    }
  }
  Patterns(ICLNetwork) {
    ICLNetworkVerify(corea) {
    }
  }
  Patterns(MemoryBist_P1) {
    ClockPeriods {
      clkb : 13.2ns;
      clka : 10.0ns;
    }
    TestStep(run_time_prog) {
      MemoryBist {
        run_mode : run_time_prog;
        reduced_address_count : on;
        Controller(corea_first_insertion_tessent_mbist_c1_controller_inst) {
          DiagnosisOptions {
            compare_go : on;
            compare_go_id : on;
          }
        }
        Controller(corea_first_insertion_tessent_mbist_c2_controller_inst) {
          DiagnosisOptions {
            compare_go : on;
            compare_go_id : on;
          }
        }
      }
    }
  }
  Patterns(MemoryBist_ParallelRetentionTest_P1) {
    ClockPeriods {
      clkb : 13.2ns;
      clka : 10.0ns;
    }
    TestStep(ParallelRetentionTest) {
      MemoryBist {
        run_mode : hw_default;
        parallel_retention_time : 0;
        reduced_address_count : on;
        Controller(corea_first_insertion_tessent_mbist_c1_controller_inst) {
          parallel_retention_group : 1;
          DiagnosisOptions {
            compare_go_id : on;
          }
        }
        Controller(corea_first_insertion_tessent_mbist_c2_controller_inst) {
          parallel_retention_group : 1;
          DiagnosisOptions {
            compare_go_id : on;
          }
        }
      }
    }
  }
}
```

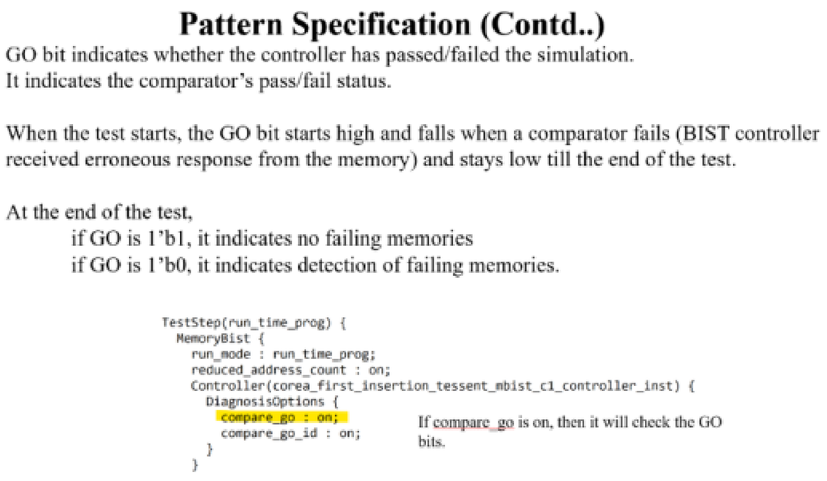


Pattern Specification (Contd…)

reduced_address_count : on ; → Enables the MBIST controller to run on 4 corners of the common memory address space.
This is useful to check the proper functionality of the BIST controller without having to simulate the test for the entire memory space.
This is reduce the run time.

```
TestStep(run_time_prog) {
  MemoryBist {
    run_mode : run_time_prog;
    reduced_address_count : on;
    Controller(corea_first_insertion_tessent_mbist_c1_controller_inst) {
      DiagnosisOptions {
        compare_go : on;
        compare_go_id : on;
      }
    }
```

reduced_address_count : on (This is done before manufacturing validation)



Pattern Specification (Contd..)

GO bit indicates whether the controller has passed/failed the simulation.
It indicates the comparator's pass/fail status.

When the test starts, the GO bit starts high and falls when a comparator fails (BIST controller received erroneous response from the memory) and stays low till the end of the test.

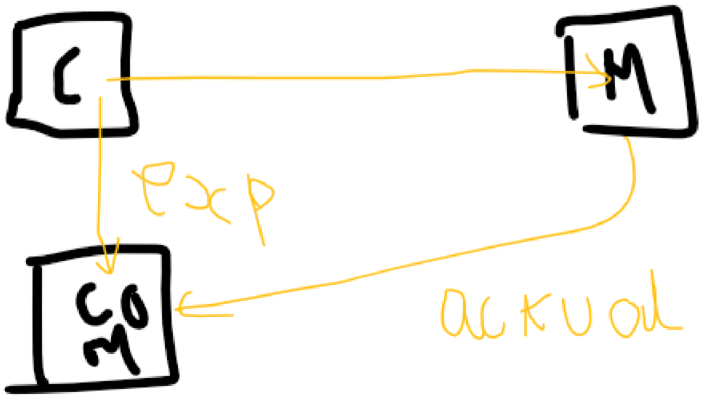
At the end of the test,
if GO is 1'b1, it indicates no failing memories
if GO is 1'b0, it indicates detection of failing memories.

```
TestStep(run_time_prog) {
  MemoryBist {
    run_mode : run_time_prog;
    reduced_address_count : on;
    Controller(corea_first_insertion_tessent_mbist_c1_controller_inst) {
      DiagnosisOptions {
        compare_go : on;
        compare_go_id : on;
      }
    }
```

If compare_go is on, then it will check the GO bits.

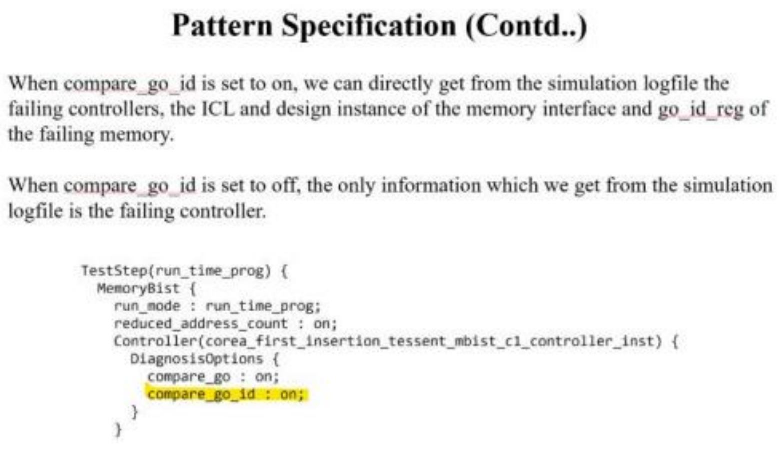
FSM and signal which is present inside the controller.

1. so it will write a value memory which is present inside a controller (0 is written onto a memory)
2. Same value will be write it out into a comparator.



3. Output of the memory will be given to the comparator and it will be checked.

- If both are same, comparator will be expected value and output of the memory will be actual value, then the go bit will be high only.
- If suppose coupling fault is there in memory, 0 has got change to 1, then we got erroneous message in memory when both these things are compared then go bit will go low and it will stay low thoughout the test.



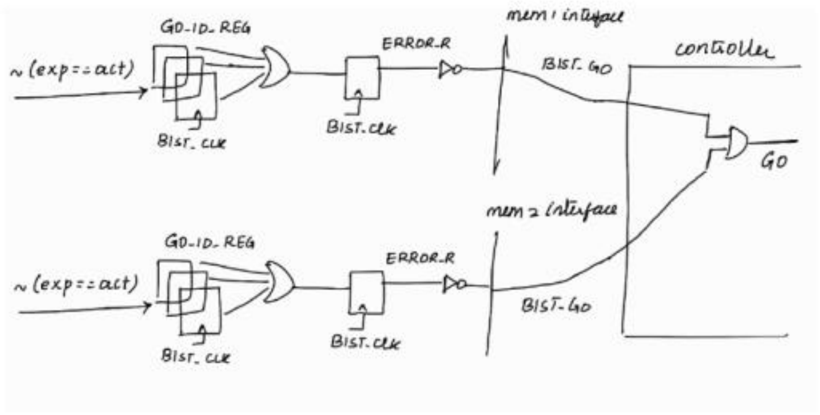
Pattern Specification (Contd..)

When compare_go_id is set to on, we can directly get from the simulation logfile the failing controllers, the ICL and design instance of the memory interface and go_id_reg of the failing memory.

When compare_go_id is set to off, the only information which we get from the simulation logfile is the failing controller.

```
TestStep(run_time_prog) {
  MemoryBist {
    run_mode : run_time_prog;
    reduced_address_count : on;
    Controller(corea_first_insertion_tessent_mbist_c1_controller_inst) {
      DiagnosisOptions {
        compare_go : on;
        compare_go_id : on;
      }
    }
```

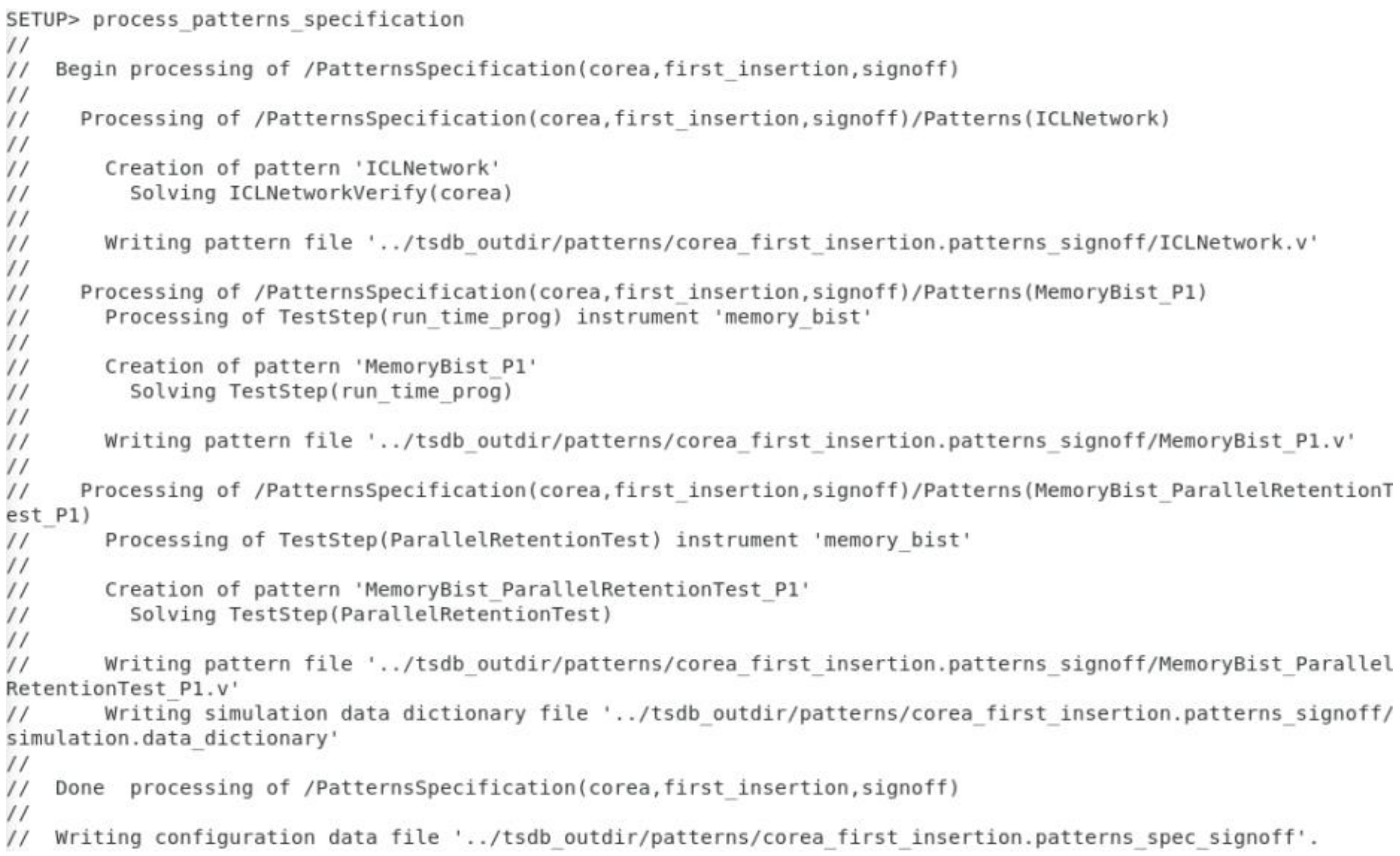
- If some memories are failing then the simulation log file will tell which controller is failing and then ICL and design instance of the memory interface and go id ref of the failing memory, if compare go id is turned on.



- Expected value is written to the comparator, and actual value from the memory that is actual patterns that is output value from the memory ~(exp == act) it's a relational operator
- The number of bits of the go id reg it will be equal to the number of dout in the memory.

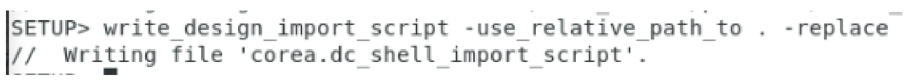
- If a single controller it is controlling more than one memory where it will be there in memory grouping concept, the BIST_GO which is specific for 1st memory and for 2nd memory again I will be having another BIST_GO, so this two things, it will inside the controller it will be added and the output will be GO

Line 82: It will validate all the patterns specifications and it will finally write out the TB and patterns files into the tsdb output directory/patterns directory.

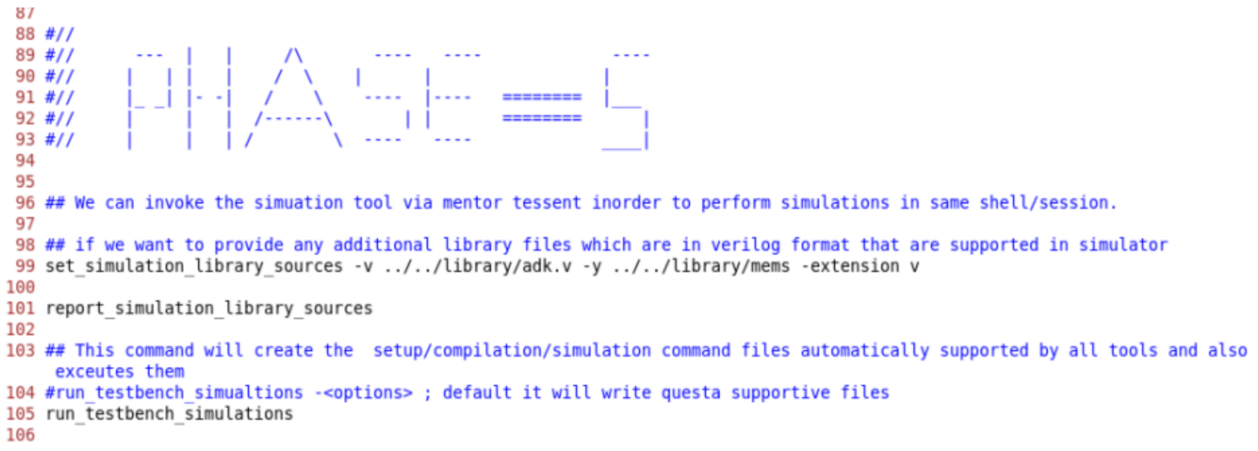


```
SETUP> process_patterns_specification
//
//  Begin processing of /PatternsSpecification(corea,first_insertion,signoff)
//
//    Processing of /PatternsSpecification(corea,first_insertion,signoff)/Patterns(ICLNetwork)
//
//      Creation of pattern 'ICLNetwork'
//        Solving ICLNetworkVerify(corea)
//
//      Writing pattern file '../tsdb_outdir/patterns/corea_first_insertion.patterns_signoff/ICLNetwork.v'
//
//    Processing of /PatternsSpecification(corea,first_insertion,signoff)/Patterns(MemoryBist_P1)
//      Processing of TestStep(run_time_prog) instrument 'memory_bist'
//
//      Creation of pattern 'MemoryBist_P1'
//        Solving TestStep(run_time_prog)
//
//      Writing pattern file '../tsdb_outdir/patterns/corea_first_insertion.patterns_signoff/MemoryBist_P1.v'
//
//    Processing of /PatternsSpecification(corea,first_insertion,signoff)/Patterns(MemoryBist_ParallelRetentionT
est_P1)
//      Processing of TestStep(ParallelRetentionTest) instrument 'memory_bist'
//
//      Creation of pattern 'MemoryBist_ParallelRetentionTest_P1'
//        Solving TestStep(ParallelRetentionTest)
//
//      Writing pattern file '../tsdb_outdir/patterns/corea_first_insertion.patterns_signoff/MemoryBist_Parallel
RetentionTest_P1.v'
//      Writing simulation data dictionary file '../tsdb_outdir/patterns/corea_first_insertion.patterns_signoff/
simulation.data_dictionary'
//
//  Done  processing of /PatternsSpecification(corea,first_insertion,signoff)
//
//  Writing configuration data file '../tsdb_outdir/patterns/corea_first_insertion.patterns_spec_signoff'.
```

Line 86: After MBIST Insertion we are not having here EDT and OCC insertion, directly we will do synthesis, then tool will directly write out the RTL file list which has to be read into the synthesis whichever files has to be inputs of synthesis that will be automatically written out by the tool.



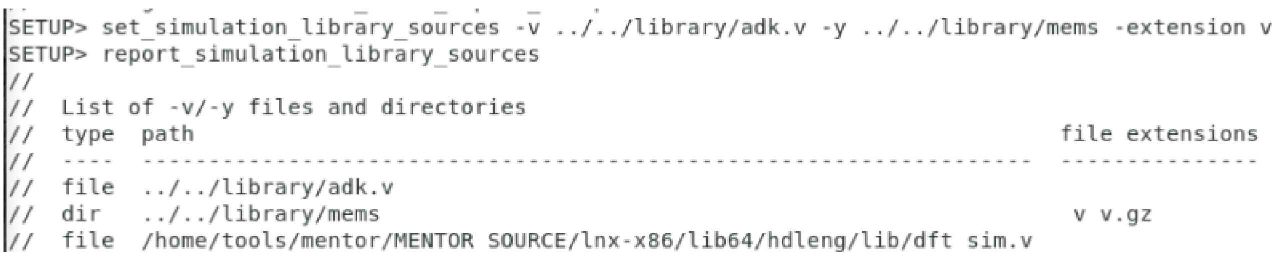
```
SETUP> write_design_import_script -use_relative_path_to . -replace
//  Writing file 'corea.dc_shell_import_script'.
```



```
87
88 #//
89 #//       ---  |   |     /\       ----    ----             ----
90 #//      |   | |   |    /  \    |       |                 |
91 #//      |_ _| |- -|   /    \    ----   |----   ========  |___
92 #//      |     |   |  /------\        | |       ========      |
93 #//      |     |   | /         \ ----    ----             ___|
94
95
96 ## We can invoke the simuation tool via mentor tessent inorder to perform simulations in same shell/session.
97
98 ## if we want to provide any additional library files which are in verilog format that are supported in simulator
99 set_simulation_library_sources -v ../../library/adk.v -y ../../library/mems -extension v
100
101 report_simulation_library_sources
102
103 ## This command will create the  setup/compilation/simulation command files automatically supported by all tools and also
    exceutes them
104 #run_testbench_simualtions -<options> ; default it will write questa supportive files
105 run_testbench_simulations
106
```

Line 99: Whichever the library file we want to read during simulation time i.e. adk.v library file we are reading and then inside this mems directory whichever having file is .v extension inside the mems directory those files will be read that is the verilog file of the memory.

Line 101: Whatever the library it has been read that will be displayed here.

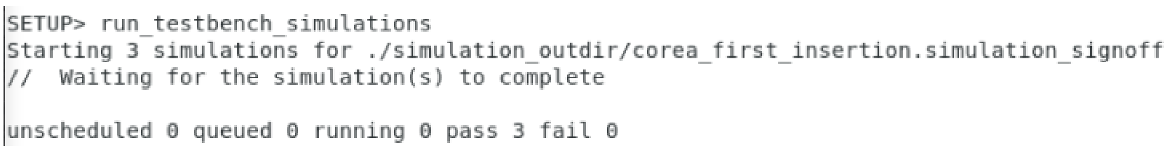


```
SETUP> set_simulation_library_sources -v ../../library/adk.v -y ../../library/mems -extension v
SETUP> report_simulation_library_sources
//
//  List of -v/-y files and directories
//  type  path                                                           file extensions
//  ----  -------------------------------------------------------------  --------------
//  file  ../../library/adk.v
//  dir   ../../library/mems                                               v v.gz
//  file  /home/tools/mentor/MENTOR SOURCE/lnx-x86/lib64/hdleng/lib/dft sim.v
```

Inside the mems, whatever having the .v extension those files will be read. It describes the functionality of the memory

Line 105: run test bench simulations

- It will automatically do the sims scripts and it will tell the status

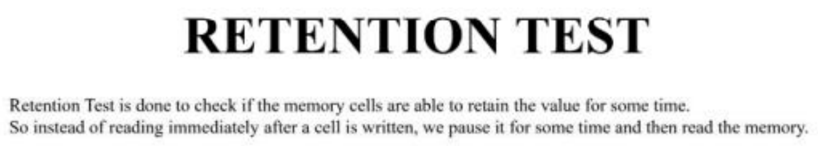


```
SETUP> run_testbench_simulations
Starting 3 simulations for ./simulation_outdir/corea_first_insertion.simulation_signoff
//  Waiting for the simulation(s) to complete

unscheduled 0 queued 0 running 0 pass 3 fail 0
```

The 3 simulations are:

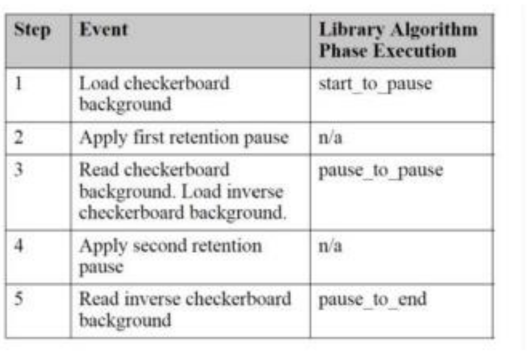
- One is to validate the IJTAG-ICL network
- One is to validate the MBIST network
- One is to validate the Memory Retention Test Network

So all the simulations has been passed

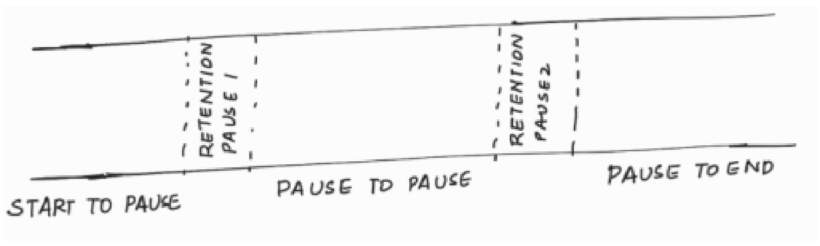


RETENTION TEST

Retention Test is done to check if the memory cells are able to retain the value for some time.
So instead of reading immediately after a cell is written, we pause it for some time and then read the memory.



Step	Event	Library Algorithm Phase Execution
1	Load checkerboard background	start_to_pause
2	Apply first retention pause	n/a
3	Read checkerboard background. Load inverse checkerboard background.	pause_to_pause
4	Apply second retention pause	n/a
5	Read inverse checkerboard background	pause_to_end



Step 1: Start to pause phase: tool will load the checkerboard algorithm, It means alternatively 0's and 1's will be return.

Step 2: Apply first retention pause

Step 3: Whatever the checkerboard pattern is written here that will be read, and the inverse checkerboard pattern will be loaded. Like 1st for Black Cells in station it is representing like 1 is written then all white cells 0 is written that is checkerboard pattern.
Inverse Checkerboard pattern it is just the opposite that is black cells 0 will be written, and for white cells 1 will be written.

Step 4: Apply second retention pause

Step 5: Inverse checkerboard pattern will be read.

- **So this is what happens in memory retention test. This is retention test, to check if the memory cells we are able to retain the value for sometime or not. So instead of reading the memory immediately after it has written we will read the memory after some time.**
- **So write the memory pause it for sometime and afterwards will read the memory.**

Extracting the ICL

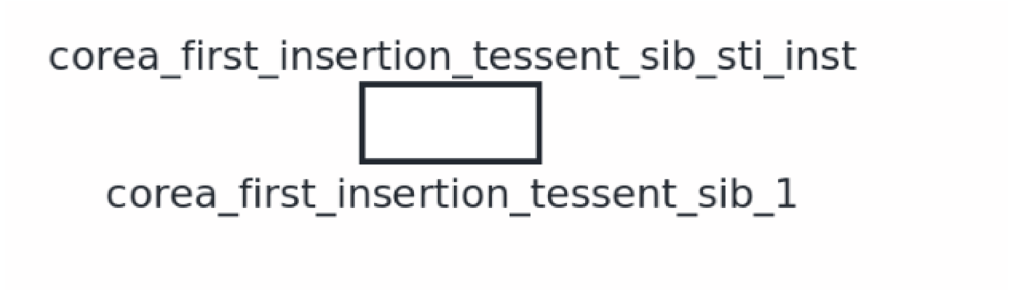
```
## Till above command, tool will be in insertion state, giving below command will make the tool to shift to
setup state
extract_icl

stop
```

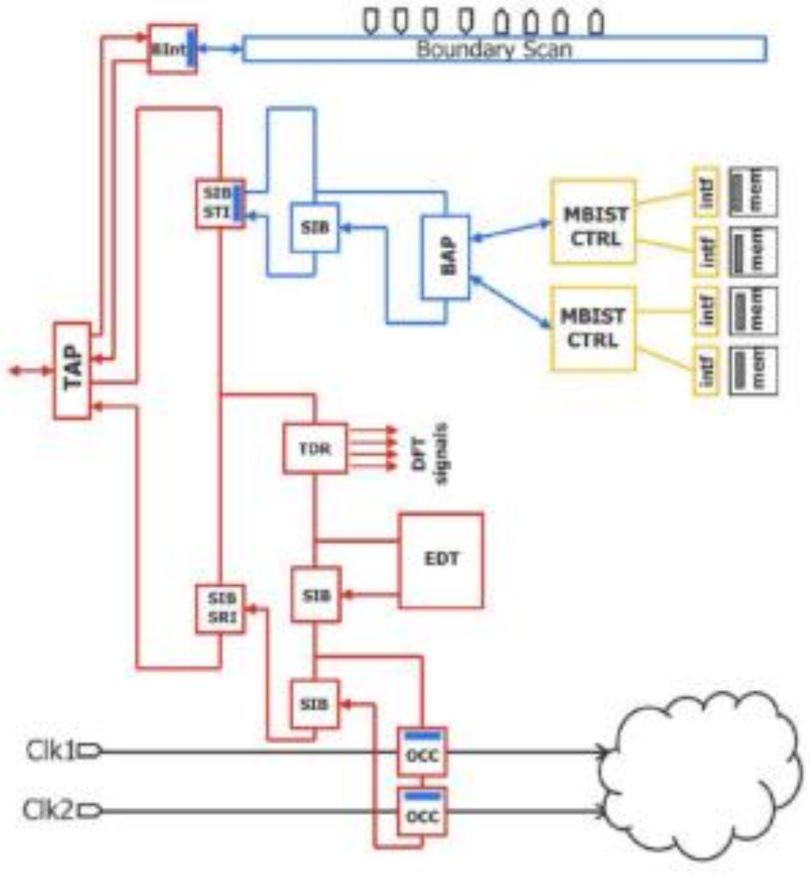
To get the complete path

```
SETUP> get_instance *sib* -hier
{corea_first_insertion_tessent_sib_mbist_inst corea_first_insertion_tessent_sib_sri_inst corea_first_insertion_t
essent_sib_sti_inst corea_first_insertion_tessent_mbist_bap_inst/corea_first_insertion_tessent_mbist_controller_
sib_ctl_bypass_inst corea_first_insertion_tessent_mbist_bap_inst/corea_first_insertion_tessent_mbist_controller_
sib_inst0 corea_first_insertion_tessent_mbist_bap_inst/corea_first_insertion_tessent_mbist_controller_sib_inst1
corea_first_insertion_tessent_mbist_bap_inst/corea_first_insertion_tessent_mbist_controller_sib_tdr_bypass_inst}
```

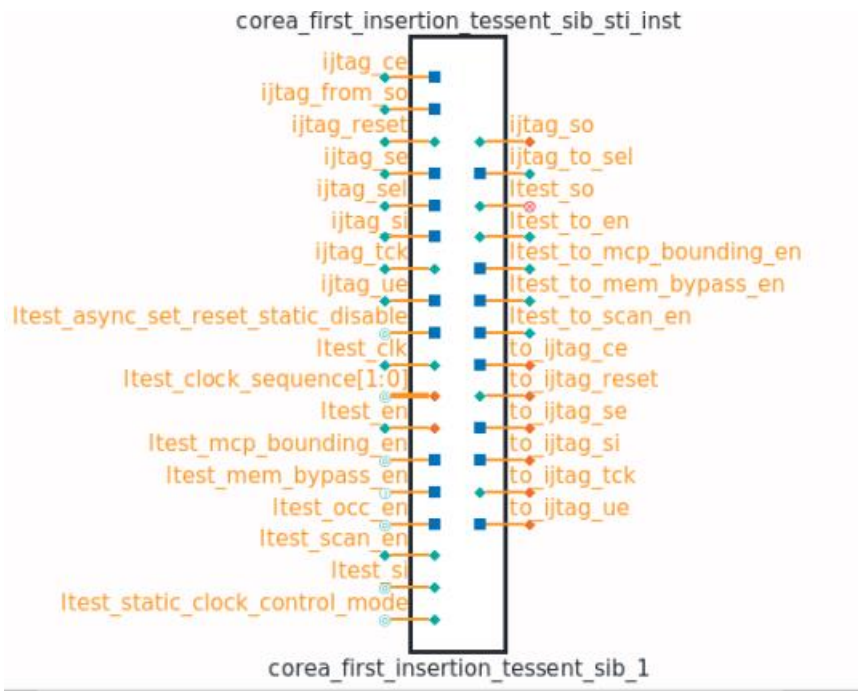
Add schematic objects in 2021 version and it was similar to add display instances in the 2016 tessent version

```
SETUP> a_s_o corea_first_insertion_tessent_sib_sti_inst -disp hier
```

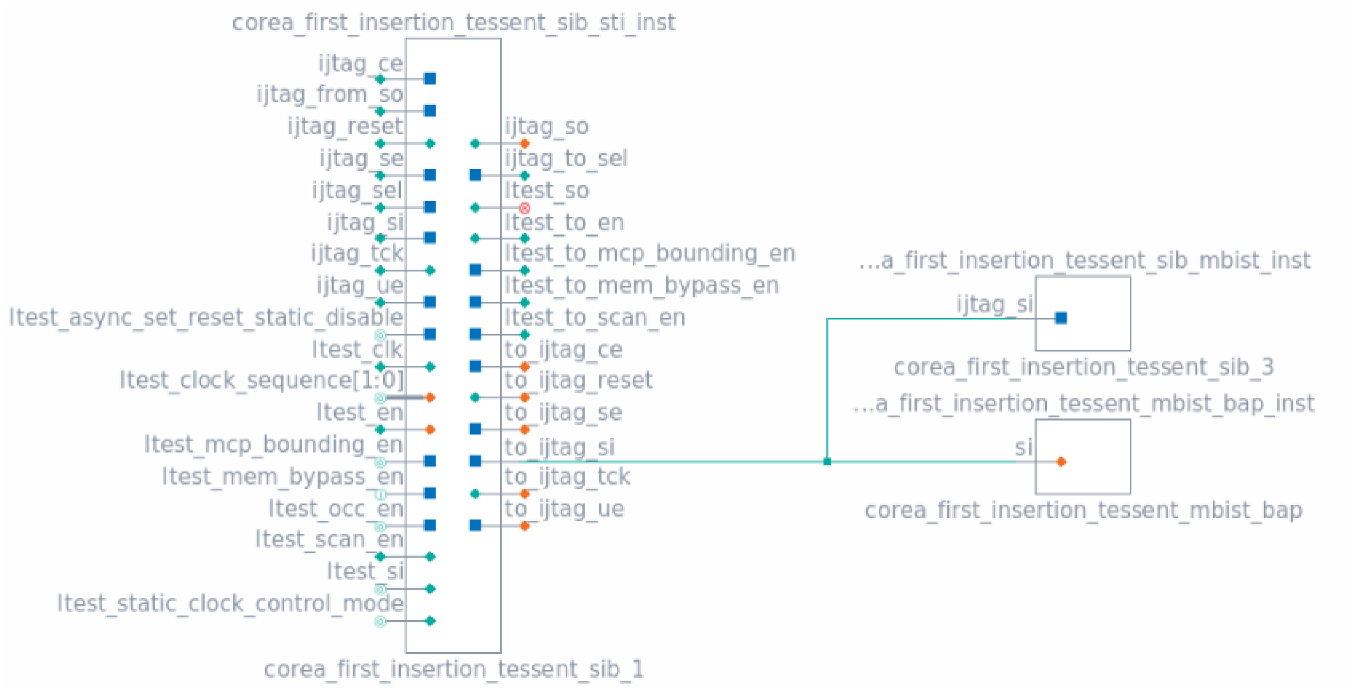

corea_first_insertion_tessent_sib_sti_inst
corea_first_insertion_tessent_sib_1



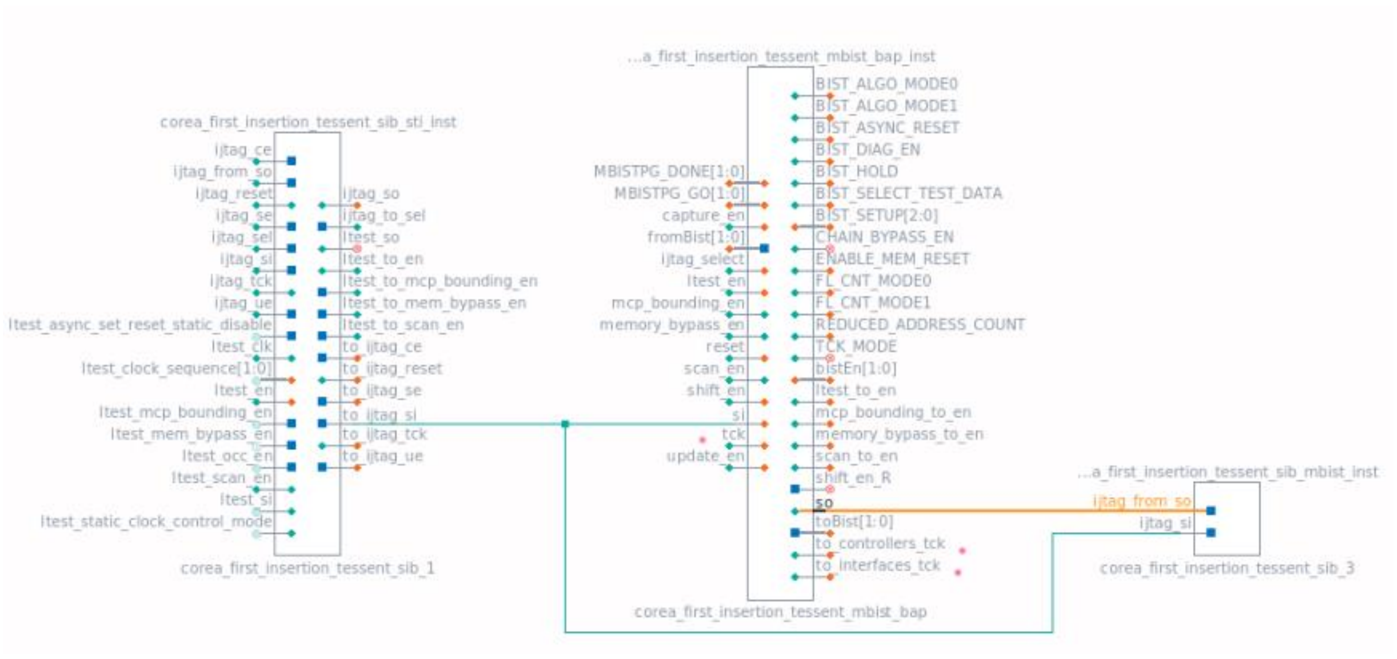
```
SETUP> a_s_o corea_first_insertion_tessent_sib_sti_inst -disp hier
```



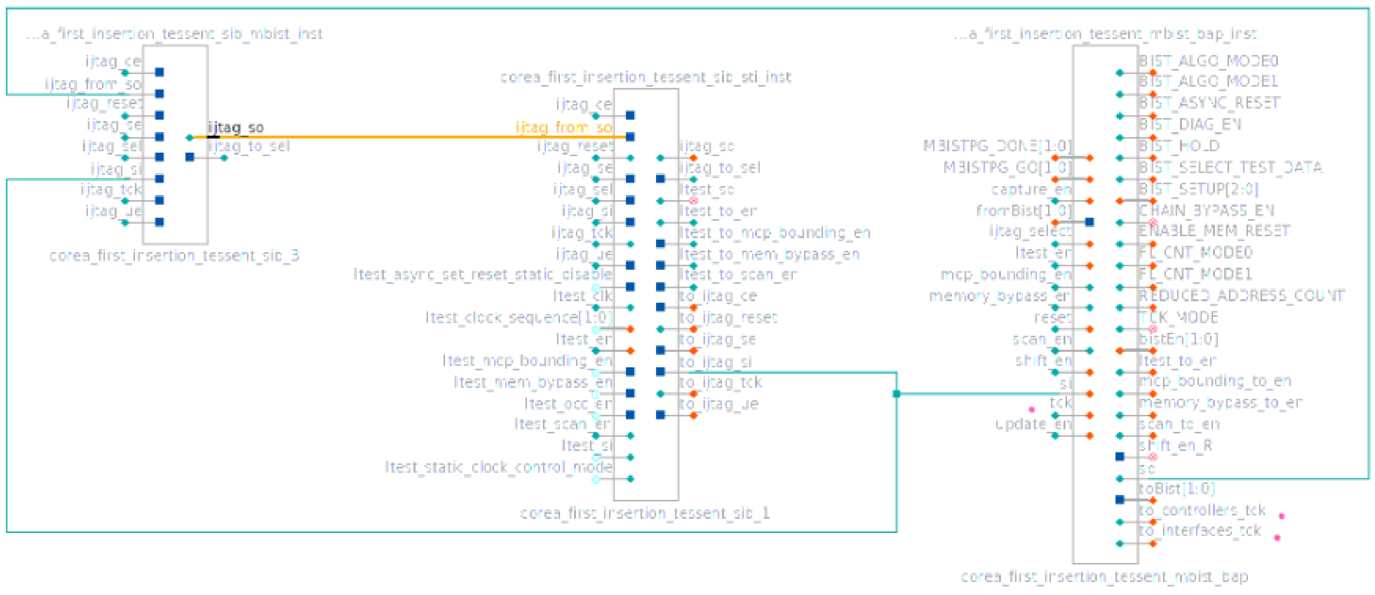
a. IJTAG to SI it will be going to SIB MBIST and another fanout will be going to BAP



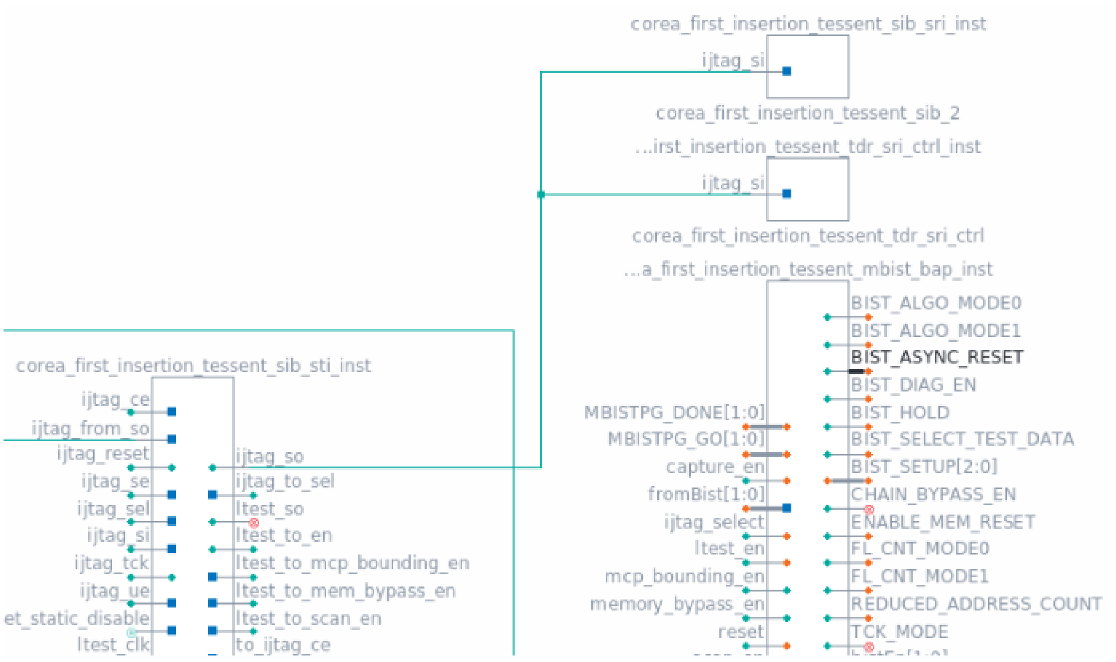
b. BAPS output it will got SIB MBIST



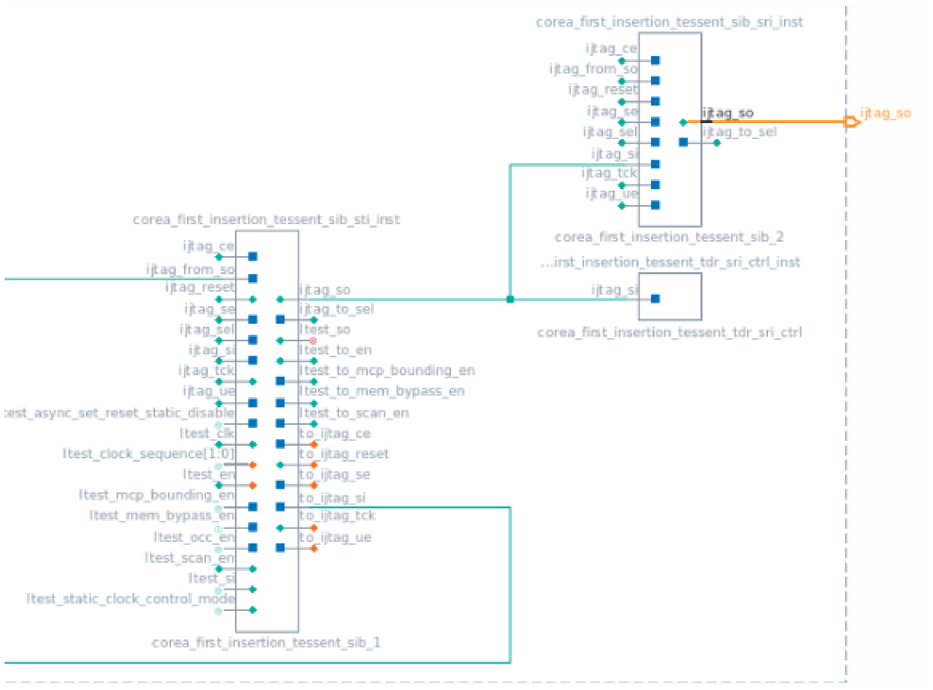
c. SIB MBIST it will go SIB STI



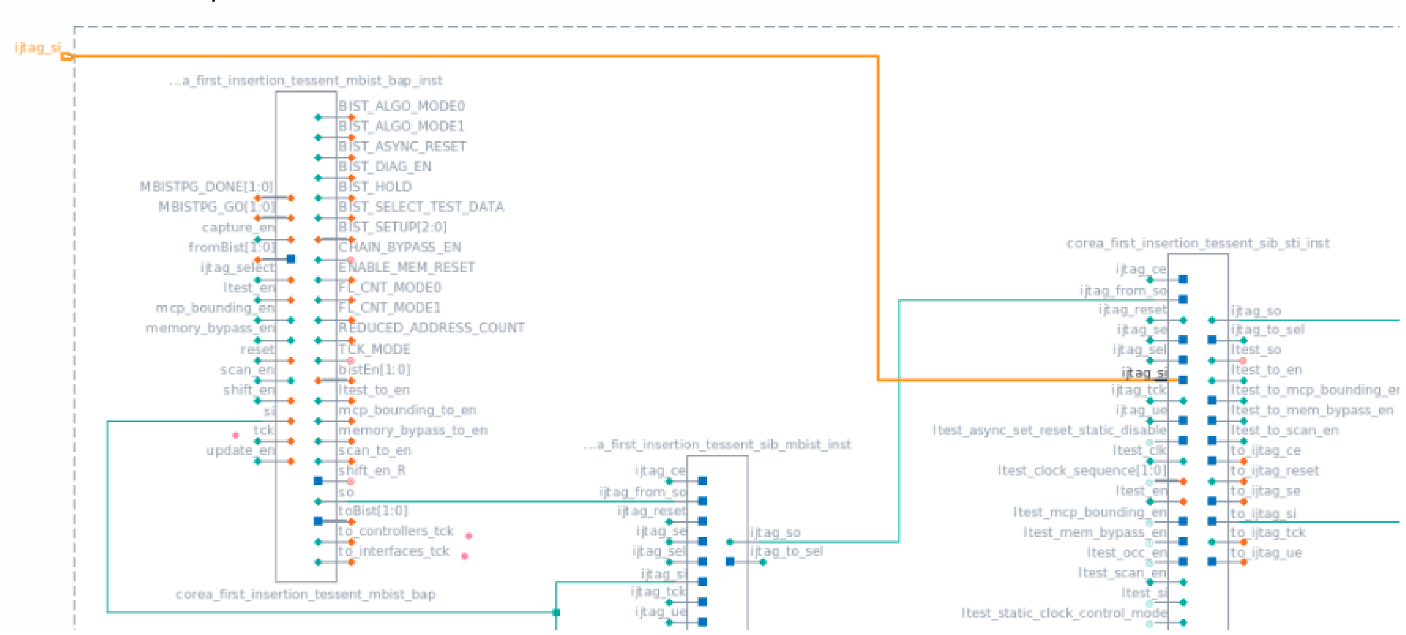
d. SIB STI Output will go SIB SRI and another fanout will go to TDR



e. SIB SRI output will go to Top level Port, but later if we go in COREB level it will controlled by another SIB

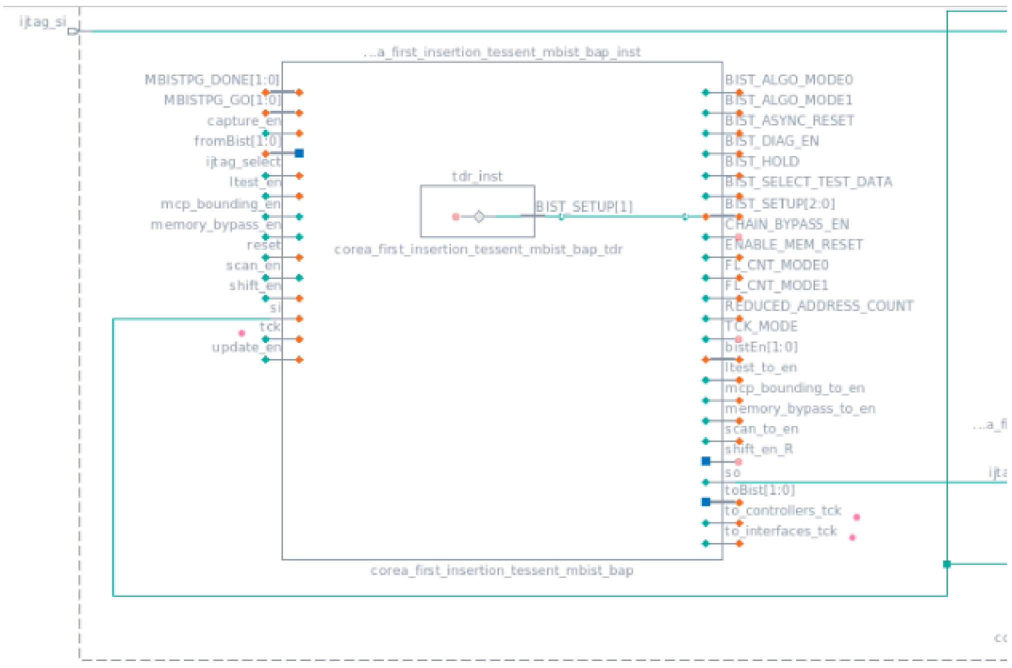


f. SIB STI Input is coming from IJTAG Design which is a TOP level port but later if we go for COREB level, it will be controlled by another SIB.

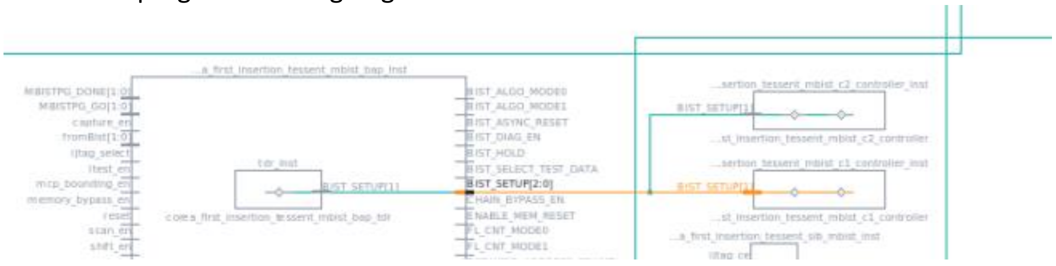


g. Inside the BAP it has a TDR logic, so the output of BAP will be controlling the signals

- It requires MBIST PG Enable, MBIST Setup signal, these signals has to be controlled in order to start the MBIST operation, these signals will be controlled by BAP so it has TDR inside a BAP that will be controlling the signals.

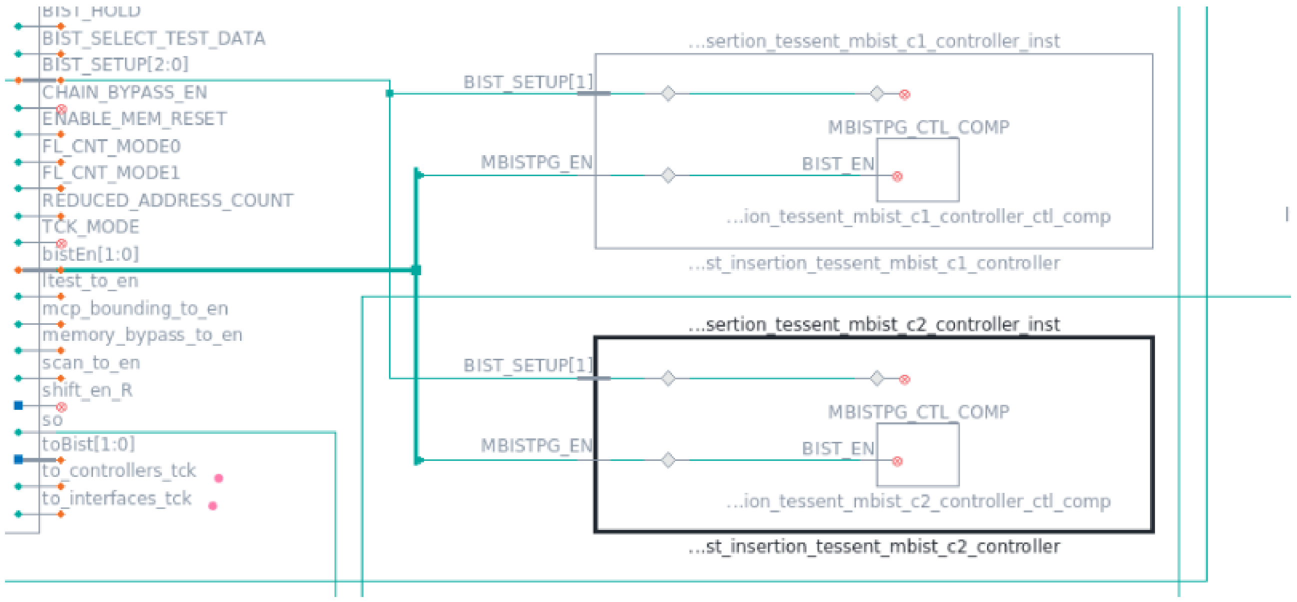


- BIST setup signals will be going to the controller C1 and C2


...a_first_insertion_tessent_mbist_bap_inst
tdr_inst
BIST_SETUP[1]
corea_first_insertion_tessent_mbist_bap_tdr
BIST_SETUP[2:0]
...sertion_tessent_mbist_c2_controller_inst
...st_insertion_tessent_mbist_c2_controller
...sertion_tessent_mbist_c1_controller_inst
...st_insertion_tessent_mbist_c1_controller
...a_first_insertion_tessent_sib_mbist_inst

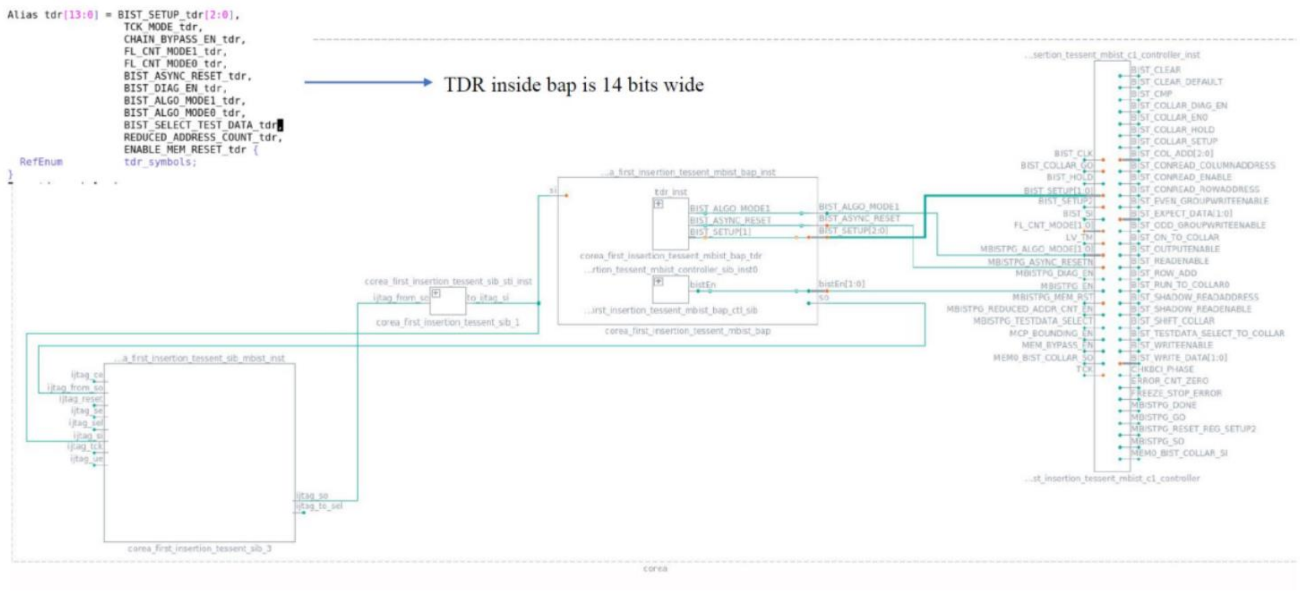
h. BIST Enable signal will be controlling the controller C1 and controller C2, so these so signals are necessary for initiating the test,

- So BIST setup has to be 1 0 and MBISTPG_EN has to be 1
- BIST_SETUP will be controlled by BAP
- MBISTPG_EN will be going to the input of the controller.



i. BAP will be controlling 14 other signals.

- TDR which is present inside the BAP

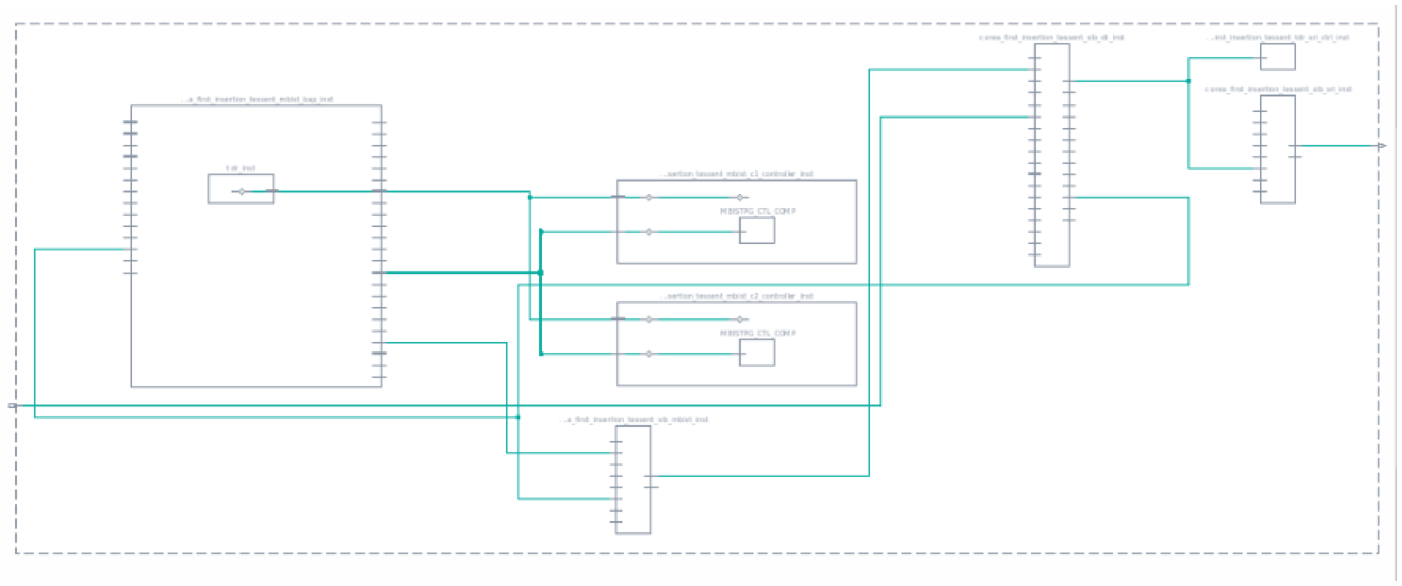


```
Alias tdr[13:0] = BIST_SETUP_tdr[2:0],
                  TCK_MODE_tdr,
                  CHAIN_BYPASS_EN_tdr,
                  FL_CNT_MODE1_tdr,
                  FL_CNT_MODE0_tdr,
                  BIST_ASYNC_RESET_tdr,
                  BIST_DIAG_EN_tdr,
                  BIST_ALGO_MODE1_tdr,
                  BIST_ALGO_MODE0_tdr,
                  BIST_SELECT_TEST_DATA_tdr,
                  REDUCED_ADDRESS_COUNT_tdr,
                  ENABLE_MEM_RESET_tdr {
  RefEnum         tdr_symbols;
}
```

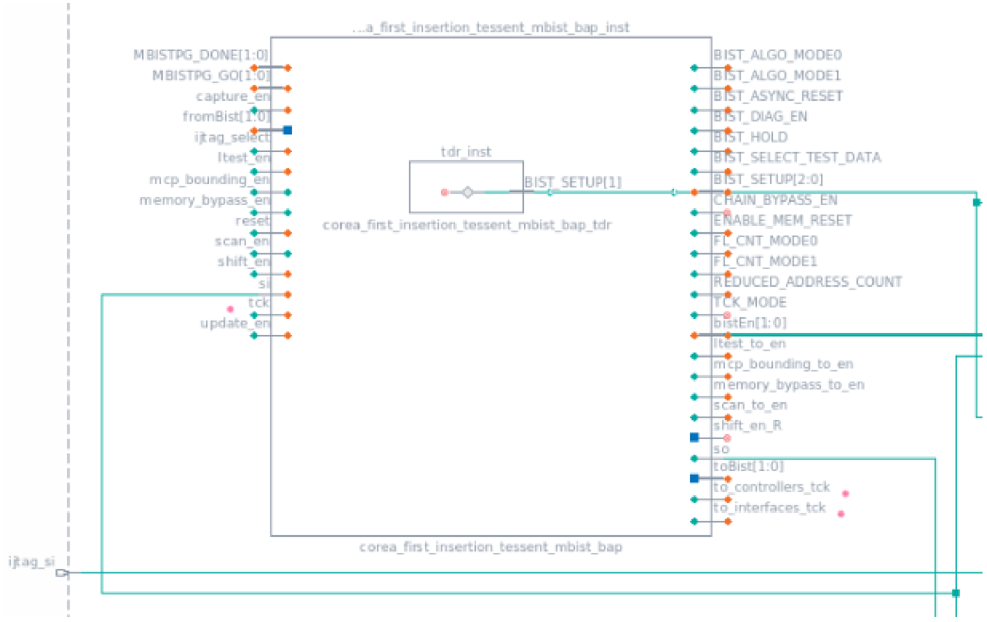
TDR inside bap is 14 bits wide
...sertion_tessent_mbist_c1_controller_inst
...a_first_insertion_tessent_mbist_bap_inst
tdr_inst
BIST_ALGO_MODE1
BIST_ASYNC_RESET
BIST_SETUP[1]
corea_first_insertion_tessent_mbist_bap_tdr
corea_first_insertion_tessent_sib_sti_inst
ijtag_from_si
to_ijtag_si
corea_first_insertion_tessent_sib_1
...irst_insertion_tessent_mbist_bap_ctl_sib
corea_first_insertion_tessent_mbist_bap
...a_first_insertion_tessent_sib_mbist_inst
corea_first_insertion_tessent_sib_3
...st_insertion_tessent_mbist_c1_controller
corea

source schematic_ijtag_network.tcl to see it in the future purpose when we use it.

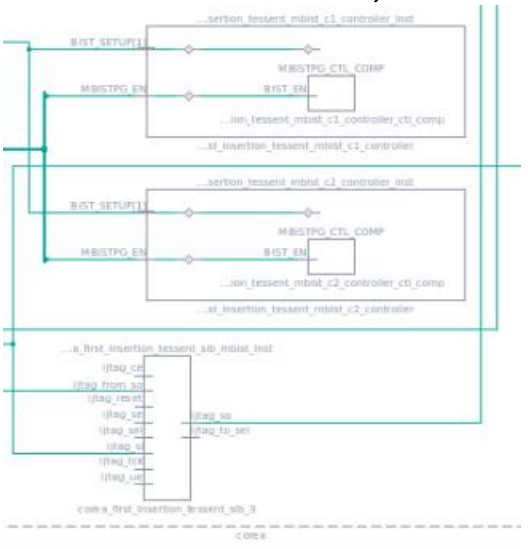
Hierarchical Schematic for COREA


...a_first_insertion_tessent_mbist_bap_inst
tdr_inst
...sertion_tessent_mbist_c1_controller_inst
MBISTPG_CTL_COMP
...sertion_tessent_mbist_c2_controller_inst
MBISTPG_CTL_COMP
...a_first_insertion_tessent_sib_mbist_inst
corea_first_insertion_tessent_sib_sti_inst
...irst_insertion_tessent_tdr_sri_ctrl_inst
corea_first_insertion_tessent_sib_sri_inst

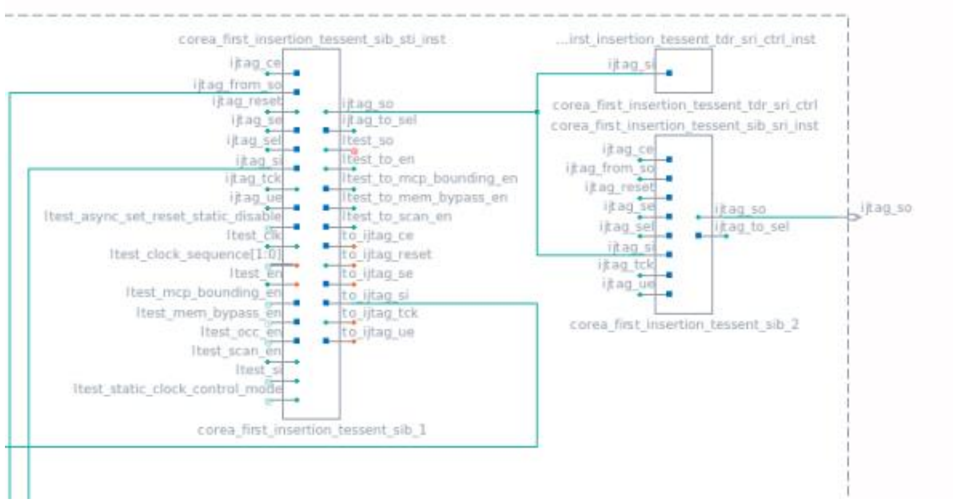
a. MBIST BAP



b. MBIST C1 and C2 Controllers, and MBIST SIB


...sertion_tessent_mbist_c1_controller_inst
BIST_SETUP[1]
MBISTPG_CTL_COMP
MBISTPG_EN
BIST_EN
...ion_tessent_mbist_c1_controller_ctl_comp
...st_insertion_tessent_mbist_c1_controller
...sertion_tessent_mbist_c2_controller_inst
BIST_SETUP[1]
MBISTPG_CTL_COMP
MBISTPG_EN
BIST_EN
...ion_tessent_mbist_c2_controller_ctl_comp
...st_insertion_tessent_mbist_c2_controller
...a_first_insertion_tessent_sib_mbist_inst
ijtag_ce
ijtag_from_so
ijtag_reset
ijtag_se
ijtag_sel
ijtag_si
ijtag_tck
ijtag_ue
ijtag_so
ijtag_to_sel
corea_first_insertion_tessent_sib_3
corea

c. SIB STI, TDR SRI and SIB - 2


corea_first_insertion_tessent_sib_sti_inst
ijtag_ce
ijtag_from_so
ijtag_reset
ijtag_se
ijtag_sel
ijtag_si
ijtag_tck
ijtag_ue
ltest_async_set_reset_static_disable
ltest_clk
ltest_clock_sequence[1:0]
ltest_en
ltest_mcp_bounding_en
ltest_mem_bypass_en
ltest_occ_en
ltest_scan_en
ltest_si
ltest_static_clock_control_mode
ijtag_so
ijtag_to_sel
ltest_so
ltest_to_en
ltest_to_mcp_bounding_en
ltest_to_mem_bypass_en
ltest_to_scan_en
to_ijtag_ce
to_ijtag_reset
to_ijtag_se
to_ijtag_si
to_ijtag_tck
to_ijtag_ue
corea_first_insertion_tessent_sib_1
...irst_insertion_tessent_tdr_sri_ctrl_inst
ijtag_si
corea_first_insertion_tessent_tdr_sri_ctrl
corea_first_insertion_tessent_sib_sri_inst
ijtag_ce
ijtag_from_so
ijtag_reset
ijtag_se
ijtag_sel
ijtag_si
ijtag_tck
ijtag_ue
ijtag_so
ijtag_to_sel
corea_first_insertion_tessent_sib_2
ijtag_so