Memory Built-In Self-Test (MBIST)

Digital System Design

Term Project
Fall 2011

Prepared by

Phan Dang Ngoc (2010315435)
Aruna Ranaweera (2011315578)

Contents

1. Introduction

With the advent of deep-submicron level VLSI technology, core-based system-on-chip (SOC) design is attracting an increasing attention. On an SOC, popular reusable cores include memories (such as ROM, SRAM, DRAM and flash memory), processors (such as CPU, DSP and microcontroller), input/output circuits, etc. Memory cores are obviously among the most universal ones-almost all system chips contain. A memory is considered embedded if some or all of its terminals are not directly connected to the I/O pins of the host chip.

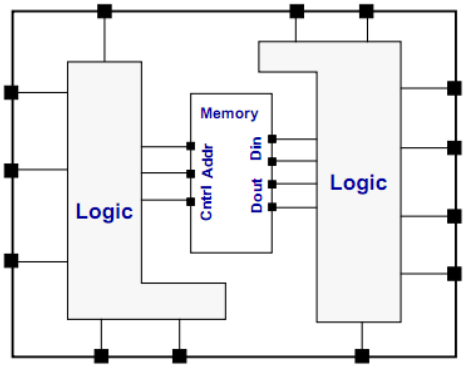


Figure 1.1: Representation of Embedded Memory

1.1 Semiconductor Memory Testing

Semiconductor memory testing research dates back to the early 1960s, with a history aligned with the growth of IC industry. Although test time and test coverage have always been major concerns, the industry basically enjoys mature techniques and tools for manufacturing test of memory products. The introduction of system chips did bring forth new problems for researchers. Both the number of embedded memory cores and area occupied by memories are rapidly increasing on system chips. The yield of on-chip memories thus determines chip yield. Go/no-go testing is no longer enough for embedded memories in the SOC era. In addition, memories have been widely used as the technology driver; that is, they are often designed with a density that is at the extremes of the process technology. Memory diagnosis is quickly becoming a critical issue, as far as manufacturing yield and time-to-market of SOC products are concerned. Effective memory diagnosis and *failure analysis* (FA) methodologies will help improve the yield of SOC products, especially with rapid evolution in new product development and advanced process technologies.

Testing embedded memory is more difficult than testing commodity memory. The first testing issue is accessibility. Accessing the Memory core from an external memory tester is costly in terms of pin/area overhead, performance penalty, and noise issues when the memory core is embedded in a CPU or ASIC and surrounded by logic blocks. Proper ***design for-testability*** (DFT) methodology must be provided for core isolation and tester access, and a

price has to be paid for the resulting hardware overhead, performance penalty, and noise and parasitic effects. Direct access and Scan access are two such tester accessing methods.

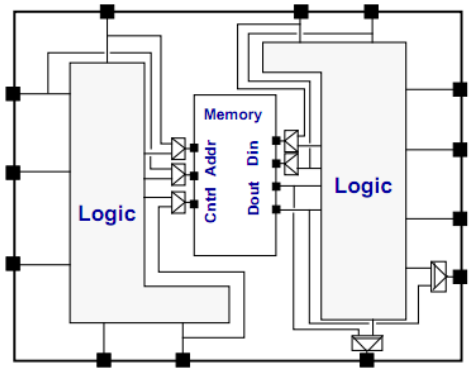


Figure 1.2: Direct access for Embedded Memory

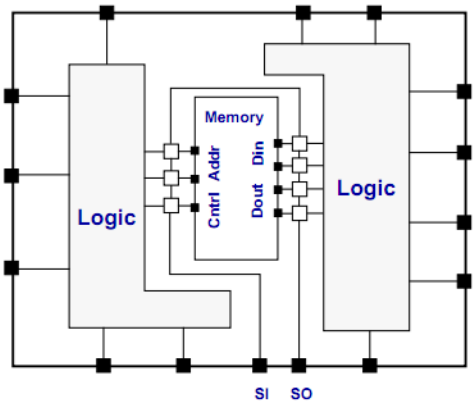


Figure 1.3: Scan access for Embedded Memory

Even if the accessing methods are manageable, memory testers for full qualification and testing of embedded memory will be much more expensive due to increased speed and I/O data width, and if we also consider engineering change the overall investment will be even higher. A promising solution to this dilemma is BIST. With memory BIST, the entire memory testing algorithm is implemented on-chip, and operates at the speed of the circuit.

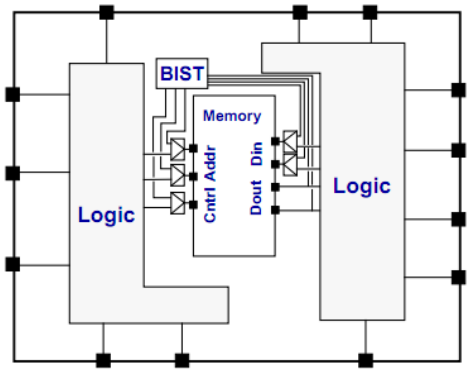


Figure 1.4: BIST Approach for Memory

Memory BIST is considered to be one of the most cost-effective and widely used solutions

for embedded memory testing due to the following reasons.

- No external test equipment
- Reduced development efforts
- Tests can run at circuit speed to yield a more realistic test time
- On-chip test pattern generation to provide higher controllability and observability.
- On-chip response analysis
- Test can be on-line or offline
- Easier burnin support
- It can support both Static and dynamic functional faults

1.2 BIST Architectures and Functions

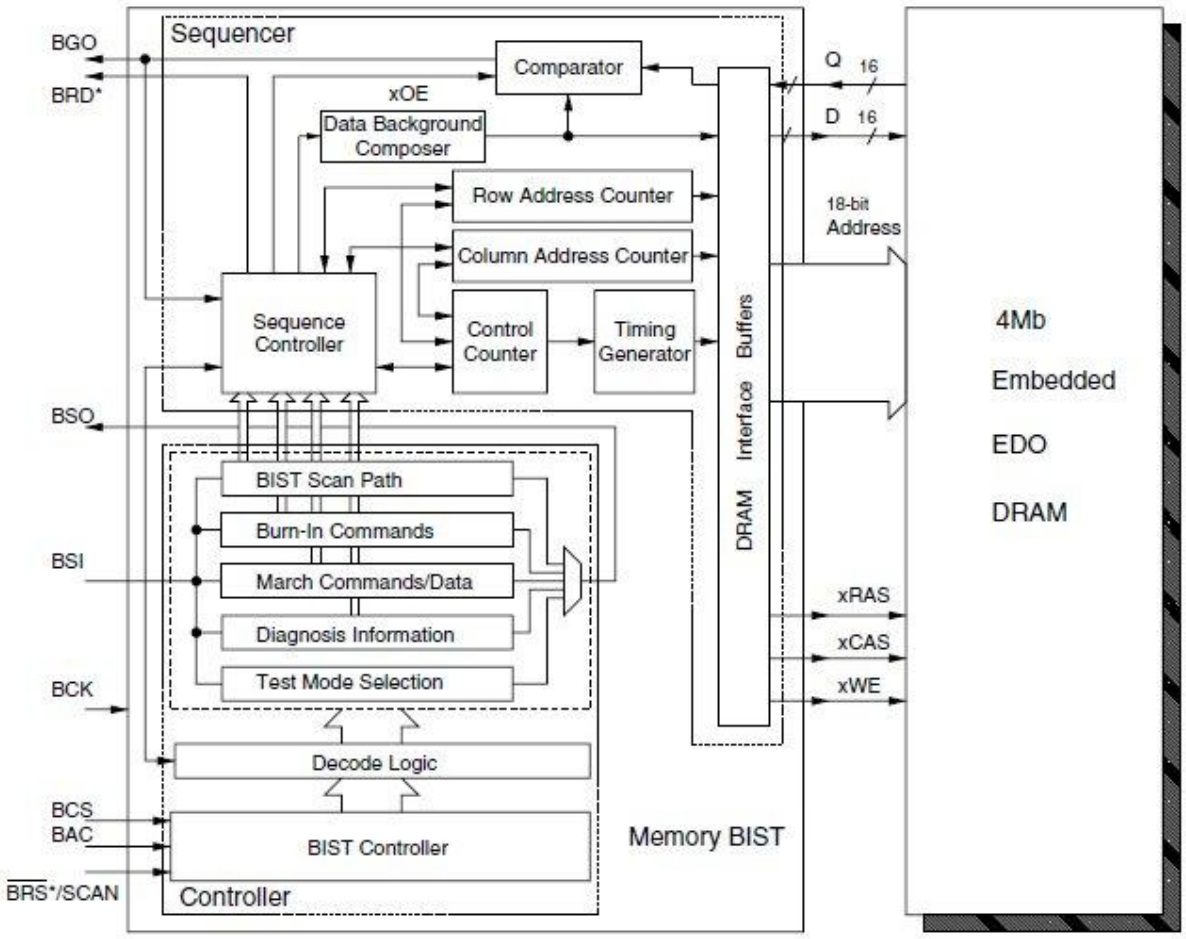


Figure 1.5: MBIST Architecture

Figure 1.5 shows the block diagram of the BIST design and the interface between the BIST logic and the EDRAM. The BIST logic is activated by the ***BIST activation control*** (BAC) input; that is, the EDRAM is in normal mode when BAC = 0, and it is in BIST mode when BAC=1. The BIST controller is a *finite-state machine* (FSM), whose state transition is controlled by the ***BIST control selection*** (BCS) input. The BIST controller also controls the

scan chains-test patterns and commands can be shifted in from the ***BIST scan-in*** (BSI) input and results can be shifted out from the ***BIST scan-out*** (BSO) output. As shown in Figure 1.5 it has multiple chains internally. The Decode Logic and Test Mode Selection modules determine the proper data register to scan in the test commands and subsequently activate the sequencer. The sequencer generates the timing sequence for the EDRAM, with the help of some built-in counters and the timing generator. The output data (Q) from the EDRAM will be compared with the original input data (D) generated by the sequence controller. The comparison is done by the comparator, which will report any discrepancy.

Apart from BAC, BCS, BSI, and BSO, the BIST logic has three additional I/O signals. The ***BIST ready flag*** (BRD) is used to indicate when the BIST sequence is finished, so the ***Go/No-Go indicator signal*** (BGO) can be sampled to check whether the EDRAM is functioning correctly or not. The BRS/SCAN signal is used as both the reset and scan test control. All registers in the BIST controller FSM are scanned, and before we use the BIST logic to test the EDRAM the logic itself is scan tested. Finally, we need a ***BIST clock*** (BCK) input.

Note that BCK and BAC have to be dedicated; that is, these two input pins cannot be shared (*e.g.*, using multiplexers) by others, but BRD is optional and may be removed if pin count is a concern (in that case, we can encode BGO to signal the completion of the BIST sequence and show the test result). The ***reset*** (BRS) also is optional, as a short synchronizing sequence for the BIST controller (an FSM) can be used as the reset sequence. However, the SCAN pin is still required in that case. Apart from BCK and BAC, all other I/O signals of the BIST logic can share pins with signals outside the DRAM core (*i.e.*, multiplexed pins can be used to reduce pin overhead).

1.3 Memory Functional Fault Model

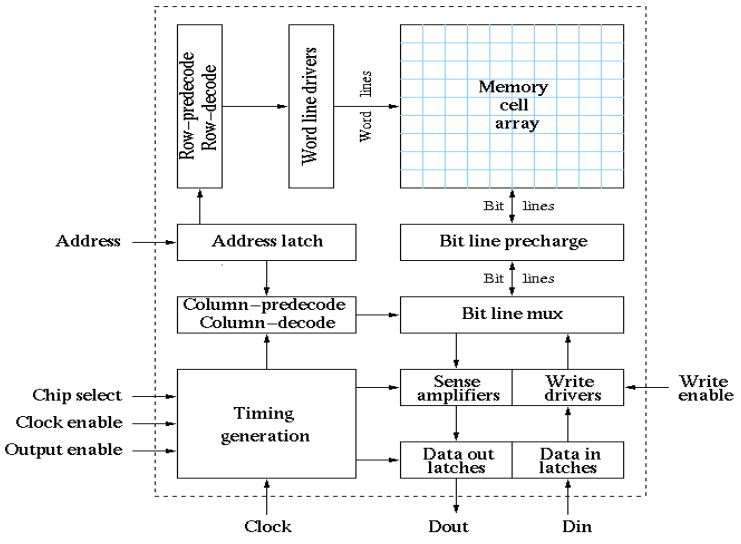


Figure 1.6: Memory Cell

Faults may occur in the address decoder, read/write circuitry, or memory cell array. They

may be static fault or dynamic faults. The faults occur in memory cell array can be single cell fault or multi cell faults.

The following notation has been used in this report to denote a fault.

$\uparrow$: Denotes a rising transition of a cell (due to a write operation).
$\downarrow$: Denotes a falling transition of a cell.
$\updownarrow$: Denotes either a rising or a falling transition of a cell.
$\forall$: Denotes any operation of a cell
$< S/F >$: Denotes a fault in a cell, where S is the value or operation activating the fault, F is the faulty value of the cell, $S \in \{0,1,\uparrow,\downarrow,\updownarrow\}$ and $F \in \{0,1\}$
$< S_1,.....,S_{m-1},S_m / F >$: Denotes a fault involving m cells, where $S_1,... S_{m-1}$ are the conditions of the first m -1 cells, respectively, that are required to activate the fault in cell m (whose state) is Sm, F is the faulty value/state of cell m

Stuck at Faults (SAF)

A cell is stuck-at-1 or 0; $<\forall/1>$ denotes a stuck-at-1 and $<\forall/0>$ denotes a stuck-at-0.

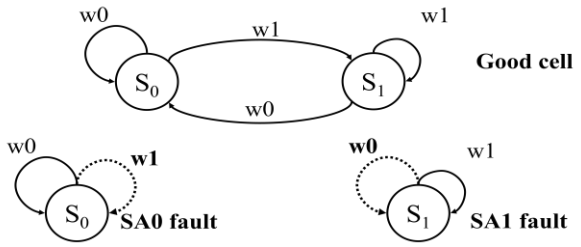


Figure 1.7: Stuck at Fault

Stuck-open fault (SOF)

A cell is not accessible due to, *e.g.*, a broken wordline or a permanent open switch.

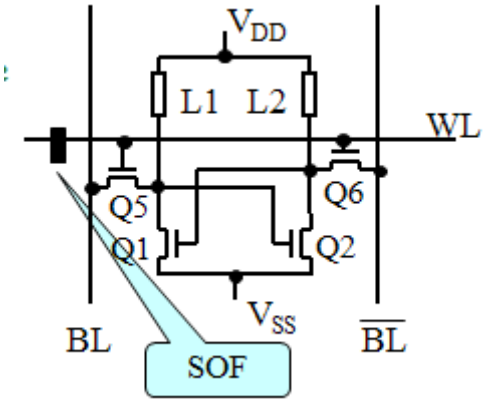


Figure 1.8: Stuck Open Fault

In this circuit when a read operation is applied to a cell, the differential sense amplifier has to

sense a voltage difference between BL and $\overline{BL}$. But in case of an SOF, BL and $\overline{BL}$ have the same (high) level.

Transition Fault (TF):

A cell fails to transit; it can be <↑ /0> or <↓ /1>.

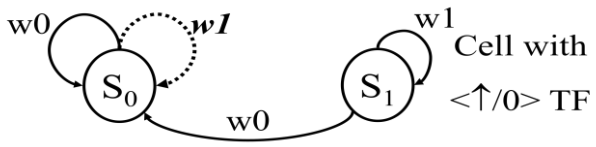


Figure 1.9: Transition Fault

Address Decoder Faults (AF):

Address decoder faults (AFs) can be categorized as follows according to their functional behavior

a. no cell can be accessed by a certain address
b. multiple cells are accessed simultaneously by a certain address
c. a certain cell is not accessible by any address
d. a certain cell is accessible by multiple addresses.

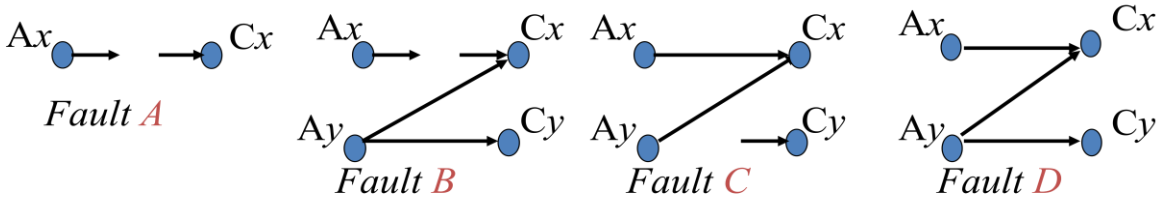


Figure 1.10: Address Decoder Faults

Data Retention Faults (DRF):

A cell cannot retain its logic value.

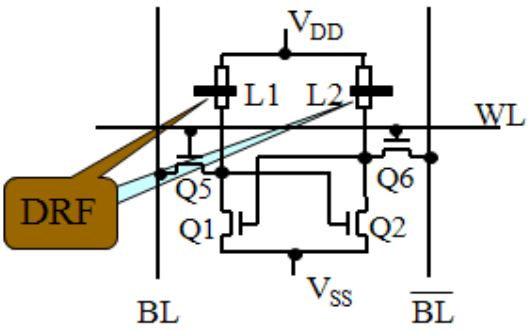


Figure 1.11: Data Retention Fault

Typically DRF is caused by a broken pull-up device which causes the leakage current of the node with a logic 1 not to be replenished. After a time delay '*Del*' (Typically: 100 ms ≤ *Del*

≤ 500 ms) the cell will flip. The DRF fault type has two subtypes, which may be present simultaneously in the same cell: $<1_T/0>$ and $<0_T/1>$. $<1_T/0>$ means that a logic 1 will become a logic 0 after a delay time *T*.

Coupling Fault (CF):

A coupling fault (CF) between two cells occurs when the logic value of a cell is influenced by the content of, or operation on, another cell. There are three types of coupling faults.

State Coupling Fault (CFst)
– Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state.
Inversion Coupling Fault (CFin)
– Transition in coupling cell complements (inverts) coupled cell.
Idempotent Coupling Fault (CFid)
– Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0.

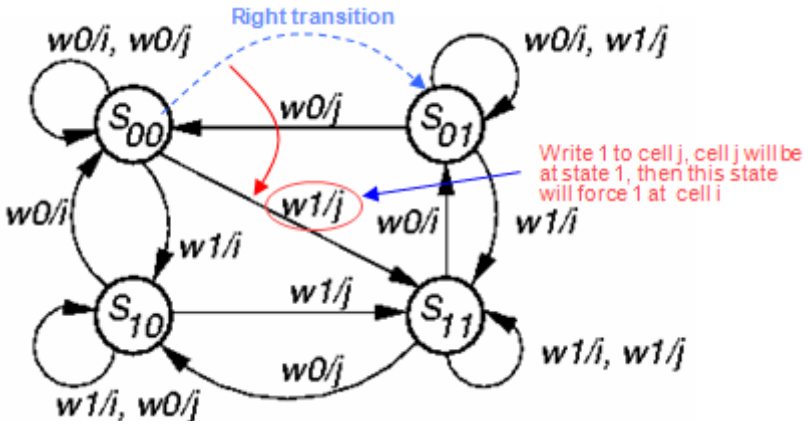


Figure 1.12: Coupling Fault

Pattern Sensitive Fault (PSF):

The PSF is a general (multi-cell) coupling fault, which causes the content of a memory cell, or the ability to change the content, to be influenced by certain patterns of other cells in the memory.

Bridging Fault (BF):

A bridging fault (BF) occurs when there is a short between two cells

Recovery Fault (RF):

There are two types of recovery faults

a. Sense Amplifier Recovery Fault (SARF)
   Sense amp saturation after reading/writing long run of 0 or 1.
b. Write Recovery Fault (WRF)

Write followed by reading/writing at different location resulting in reading/writing at same location.

Disturb Fault (DF):

Victim cell forced to 0 or 1 if we (successively) read or write aggressor cell (may be the same cell)

Read Disturb Fault (RDF):
There is a read disturb fault (RDF) if the cell value will flip when being read (successively)

2. Test Algorithms

A test algorithm (or simply test) is a finite sequence of test elements. A test element contains a number of memory operations (access commands). In those memory operations Data pattern (background) and address sequence is specified for the Read and Write operations. A number of algorithms have been developed for memory testing maximizing fault coverage while minimizing the test time. Among them march test is one of the most widely used algorithm for memory testing.

2.1 March algorithm

A March test consists of a finite sequence of March elements, while a **March element** is a finite sequence of operations applied to every cell in the memory array before proceeding to the next cell. An **operation** can consist of writing a 0 into a cell (w0), writing a 1 into a cell (w1), reading an expected 0 from a cell (r0), and reading an expected 1 from a cell (r1). The following notation is used along with (w0, w1, r0, r1) when specifying march algorithm.

$\Uparrow$: adress sequence is in the ascending order

$\Downarrow$: address order is in the descending order

$\Updownarrow$: address sequence is eitther $\Uparrow$ or $\Downarrow$

Some widely used march test algorithms are summarized in Table 2.1

Algorithm Name	Memory Operations	Faults detected
Zero-One Algorithm (MSCAN)	$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(w1);\Updownarrow(r1)\}$	SAF
Modified Algorithmic Test Sequence (MATS)	1. $\{\Updownarrow(w0);\Updownarrow(r0,w1),\Updownarrow(r1)\}$ 2. $\{\Updownarrow(w1);\Updownarrow(r1,w0);\Updownarrow(r0)\}$	OR-type AF AND-type AF
MATS+	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$	AF, SAF
Marching 1/0	$\{\Uparrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Uparrow(w1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1)\}$	AF, SAF, TF
MATS++	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)\}$	AF, SAF, TF
March X	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	AF, SAF, TF, CFin
March Y	$\{\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)\}$	AF, SAF, TF, CFin
March C	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	AF, SAF, TF, all CF
March C-	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	AF, SAF, TF, all CF
Extended March C-	$\{\Updownarrow(w0);\Uparrow(ro,w1,r1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	AF, SAF, TF, all CF, SOF

Table 2.1

March Test algorithms for Dynamic Faults

Dynamic Faults:

Dynamic faults need the application of more than one operation to in order to sensitize. The following notation is used to indicate a dynamic fault on single cell.

<S/F/R>
where S = sequence of memory operations that cause the fault {w0,w1,r0, r1}
F = Value or behavior of the faulty cell {0,1, ↑↓}
R = Logical value appears at the output {0,1,-}

Dynamic Read Destructive Fault (dRDF):
A write or read operation followed immediately by a read operation performed on a cell changes the data in the cell, and returns an incorrect value on the output.

Dynamic Deceptive Read Destructive Fault (dDRDF):
A write or a read operation followed immediately by a read operation performed on a cell changes the data in the cell and returns a correct value on the output.

Dynamic Incorrect Read Fault (dIRF):
A read operation performed immediately after a read or after a write operation fails

Dynamic Transition Fault (dTF):
A transition write operation performed immediately after a read or a write operation fails.

Dynamic Write Destructive Fault (dWDF):
A non-transition write operation preformed immediately after a read or a write operation causes the cell to flip.

March Test Algorithms for single cell Dynamic Fault are listed in Table 2.2

FFM	FPs
dRDF	⟨0r0r0/↑/1⟩, ⟨1r1r1/↓/0⟩ ⟨0w0r0/↑/1⟩, ⟨0w1r1/↓/0⟩ ⟨1w0r0/↑/1⟩, ⟨1w1r1/↓/0⟩
dDRDF	⟨0r0r0/↑/0⟩, ⟨1r1r1/↓/1⟩ ⟨0w0r0/↑/0⟩, ⟨0w1r1/↓/1⟩ ⟨1w0r0/↑/0⟩, ⟨1w1r1/↓/1⟩
dIRF	⟨0r0r0/0/1⟩, ⟨1r1r1/1/0⟩ ⟨0w0r0/0/1⟩, ⟨0w1r1/1/0⟩ ⟨1w0r0/0/1⟩, ⟨1w1r1/1/0⟩
dTF	⟨0r0w1/↓/-⟩, ⟨1r1w0/↑/-⟩ ⟨0w0w1/↓/-⟩, ⟨1w1w0/↑/-⟩ ⟨1w0w1/↓/-⟩, ⟨0w1w0/↑/-⟩
dWDF	⟨0r0w0/↑/-⟩, ⟨1r1w1/↓/-⟩ ⟨0w0w0/↑/-⟩, ⟨1w1w1/↓/-⟩ ⟨1w0w0/↑/-⟩, ⟨0w1w1/↓/-⟩

Table 2.2

Two cell Dynamic Faults

Two-cell dynamic faults consist of FPs sensitized by applying more than one operation sequentially to two cells: the aggressor (a-cell) and the v-cell. The a-cell is the cell to which the sensitizing operation (or state) should be applied in order to sensitize the fault, while the v-cell is the cell where the fault appears. Then depending on how many operations are applied to the a-cell and to the v-cell, and on the order in which they applied, four types of S can be distinguished The following notation is used to indicate this type of faults.

S_{aa} : The two sequential operations are applied to the a-cell
S_{vv} : The two sequential operations are applied to the v-cell.
S_{av}: The first operation is applied to the a-cell, followed immediately with a second one to the v-cell
S_{va}: The first operation is applied to the v-cell, followed immediately with a second one to the a-cell

$< S_a..; S_v / F / R >$ where S_a: State of cell a
S_v : State of cell v
F: operation that sensitized fault
R: Value of the faulty cell after the operation

Dynamic disturb Coupling Fault (dCFds):
a write operation followed immediately by a read operation performed on the a-cell causes the v-cell to flip.

Dynamic Read Destructive Coupling Fault (dCFrd):
A write followed immediately by a read operation performed on the v-cell changes the data in the v-cell and returns an incorrect value on the output, if the a-cell is in a certain specific state.

Dynamic Deceptive Read Destructive Coupling Fault (dCFdrd):
A write followed immediately by a read operation performed on the v-cell changes the data on the v-cell and returns a correct value on the output if the a-cell is in a certain specific state.

Dynamic Incorrect Read Coupling Fault (dCFir):
A read operation performed immediately after a write operation on the v-cell returns an incorrect value on the output, while the v-cell remains in its correct state, if the a-cell is in a certain specific state. Table 2.3 summarize some dynamic faults

FFM	Fault primitives
dCFds	$\langle 0w0r0;0/\uparrow/-\rangle, \langle 0w1r1;0/\uparrow/-\rangle,$
	$\langle 1w0r0;0/\uparrow/-\rangle, \langle 1w1r1;0/\uparrow/-\rangle,$
	$\langle 0w0r0;1/\downarrow/-\rangle, \langle 0w1r1;1/\downarrow/-\rangle,$
	$\langle 1w0r0;1/\downarrow/-\rangle, \langle 1w1r1;1/\downarrow/-\rangle$
dCFrd	$\langle x;0w0r0/\uparrow/1\rangle, \langle x;0w1r1/\downarrow/0\rangle,$
	$\langle x;1w0r0/\uparrow/1\rangle, \langle x;1w1r1/\downarrow/0\rangle$
dCFdrd	$\langle x;0w0r0/\uparrow/0\rangle, \langle x;0w1r1/\downarrow/1\rangle,$
	$\langle x;1w0r0/\uparrow/0\rangle, \langle x;1w1r1/\downarrow/1\rangle$
dCFir	$\langle x;0w0r0/0/1\rangle, \langle x;0w1r1/1/0\rangle,$
	$\langle x;1w0r0/0/1\rangle, \langle x;1w1r1/1/0\rangle$

Table 2.3

March RAW algorithm is used to detect above faults

$$\{\Updownarrow(w0);\Uparrow(r0,w0,r0,r0,w1,r1);\Uparrow(r1,w1,r1,r1,w0,r0);\Downarrow(r0,w0,r0,r0,w1,r1);\Downarrow(r1,w1,r1,r1,w0,r0);\Updownarrow(r0)\}$$

M_0 M_1 M_2 M_3 M_4 M_5

Dynamic Fault Coverage by March Algorithm

Single cell dynamic fault coverage is summarized in Table 2.4

FFM	FPs	March RAW	
		S	D
dRDF	<0r0r0/↑/1> <1r1r1/↓/0> <0w0r0/↑/1> <0w1r1/↓/0> <1w0r0/↑/1> <1w1r1/↓/0>	M1, M3 M2, M4 M1, M3 M1, M3 M2, M3 M2, M4	M1, M3 M2, M4 M1, M3 M1-M2, M3-M4 M2-M3, M4-M5 M2, M4
dDRDF	<0r0r0/↑/0> <1r1r1/↓/1> <0w0r0/↑/0> <0w1r1/↓/1> <1w0r0/↑/0> <1w1r1/↓/1>	M1, M3 M1, M3 M2, M4 M2, M4	M1, M3 M1-M2, M3-M4 M2-M3, M4-M5 M2, M4
dIRF	<0r0r0/0/1>	M1, M3	M1, M3
	<1r1r1/1/0> <0w0r0/0/1> <0w1r1/1/0> <1w0r0/0/1> <1w1r1/1/0>	M2, M4 M1, M3 M1, M3 M2, M4 M2, M4	M2, M4 M1, M3 M1, M3 M2, M4 M2, M4
dTF	<0r0w1/↓/-> <1r1w0/↑/-> <0w0w1/↓/-> <1w1w0/↑/-> <1w0w1/↓/-> <0w1w0/↑/->	M1, M3 M2, M4	M1, M3 M2, M4
dWDF	<0r0w0/↑/-> <1r1w1/↓/-> <0w0w0/↑/-> <1w1w1/↓/-> <1w0w0/↑/-> <0w1w1/↓/->	M1, M3 M2, M4	M1, M3 M2, M4

Table 2.4

❖ Fault coverage:

Test	Test length	dRDF	dDRDF	dIRF	dTF	dWDF	Total FC
March RAW	26n	6/6	4/6	6/6	2/6	2/6	20/30

Two cell dynamic fault coverage is summarized in Table 2.5

The fault coverage is given in follow table. In this table, a distinction (a) the v-cell has a higher address than the a-cell (i.e., v > a), an address than the a-cell (v < a) is used.

FFM	FPs	March RAW	
		v > a	v < a
dCFds	⟨0w0r0;0/↑/-⟩	M1/M1	M3/M3
	⟨0w1r1;0/↑/-⟩	M1/M1	M3/M3
	⟨1w1r1;0/↑/-⟩	M4/M5	M2/M3
	⟨1w0r0;0/↑/-⟩	M4/M5	M2/M3
	⟨0w0r0;1/↓/-⟩	M3/M4	M1/M2
	⟨0w1r1;1/↓/-⟩	M3/M4	M1/M2
	⟨1w1r1;1/↓/-⟩	M2/M2	M4/M4
	⟨1w0r0;1/↓/-⟩	M2/M2	M4/M4
dCFrd	⟨0; 0w0r0/↑/1⟩	M3/M3	M1/M1
	⟨0; 0w1r1/↓/0⟩	M3/M4	M1/M2
	⟨0; 1w1r1/↓/0⟩	M2/M2	M4/M4
	⟨0; 1w0r0/↑/1⟩	M2/M3	M4/M5
	⟨1; 0w0r0/↑/1⟩	M1/M1	M3/M3
	⟨1; 0w1r1/↓/0⟩	M1/M2	M3/M4
	⟨1; 1w1r1/↓/0⟩	M4/M4	M2/M2
	⟨1; 1w0r0/↑/1⟩	M4/M5	M2/M3
dCFdrd	⟨0; 0w0r0/↑/0⟩	M3/M3	M1/M1
	⟨0; 0w1r1/↓/1⟩	M3/M4	M1/M2
	⟨0; 1w1r1/↓/1⟩	M2/M2	M4/M4
	⟨0; 1w0r0/↑/0⟩	M2/M3	M4/M5
	⟨1; 0w0r0/↑/0⟩	M1/M1	M3/M3
	⟨1; 0w1r1/↓/1⟩	M1/M2	M3/M4
	⟨1; 1w1r1/↓/1⟩	M4/M4	M2/M2
	⟨1; 1w0r0/↑/0⟩	M4/M5	M2/M3
dCFir	⟨0; 0w0r0/0/1⟩	M3/M3	M1/M1
	⟨0; 0w1r1/1/0⟩	M3/M4	M1/M2
	⟨0; 1w1r1/1/0⟩	M2/M2	M4/M4
	⟨0; 1w0r0/0/1⟩	M2/M3	M4/M5
	⟨1; 0w0r0/0/1⟩	M1/M1	M3/M3
	⟨1; 0w1r1/1/0⟩	M1/M2	M3/M4
	⟨1; 1w1r1/1/0⟩	M4/M4	M2/M2
	⟨1; 1w0r0/0/1⟩	M4/M5	M2/M3

Table 2.5

- Fault coverage:

Test	Test length	dCFds	dCFrd	dCFdrd	dCFir	Total FC	FC (%)
March RAW	26n	16/16	16/16	16/16	16/16	64/64	100

Because March RAW algorithm can be used to detect all dynamic faults with higher coverage than other algorithms we choose this algorithm to implement the MBIST.

3. Design Methodology from front-end to back-end

The design flow of our design from front-end to back end is shown in the figure 3.1

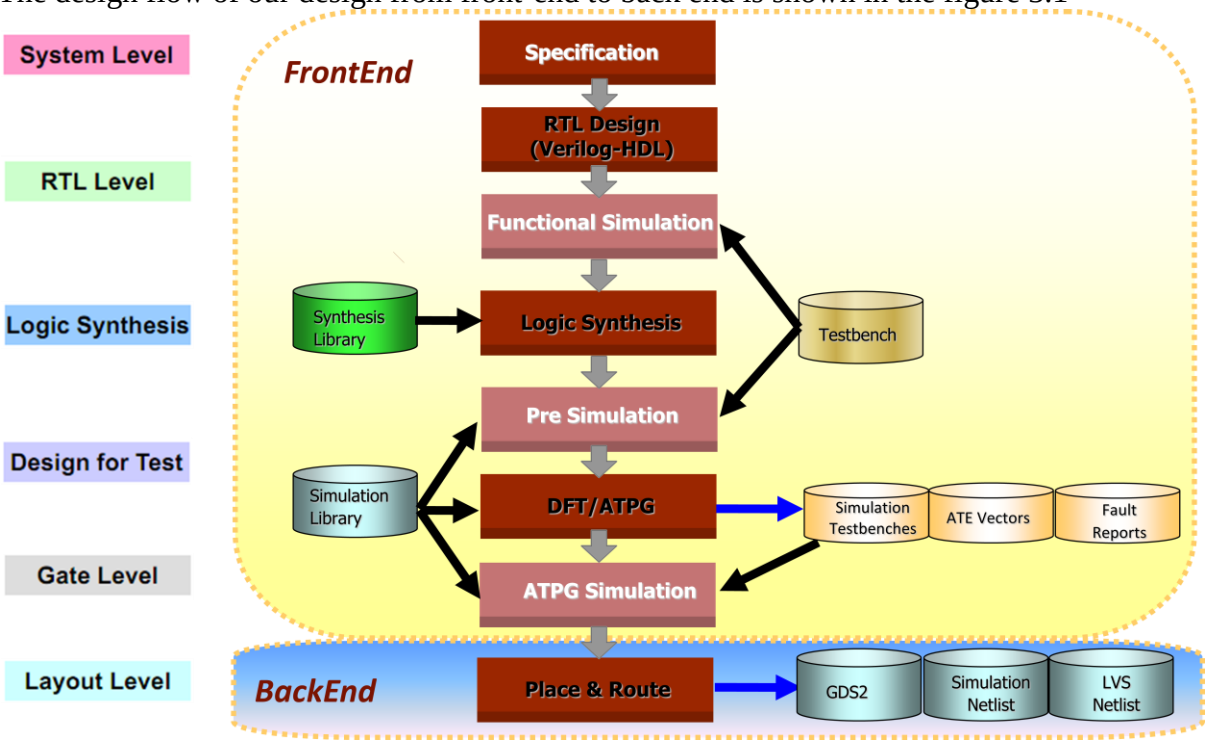


Figure 3.1

3.1 Top Level Design

The top level Design of MBIST is shown in Figure 3.2

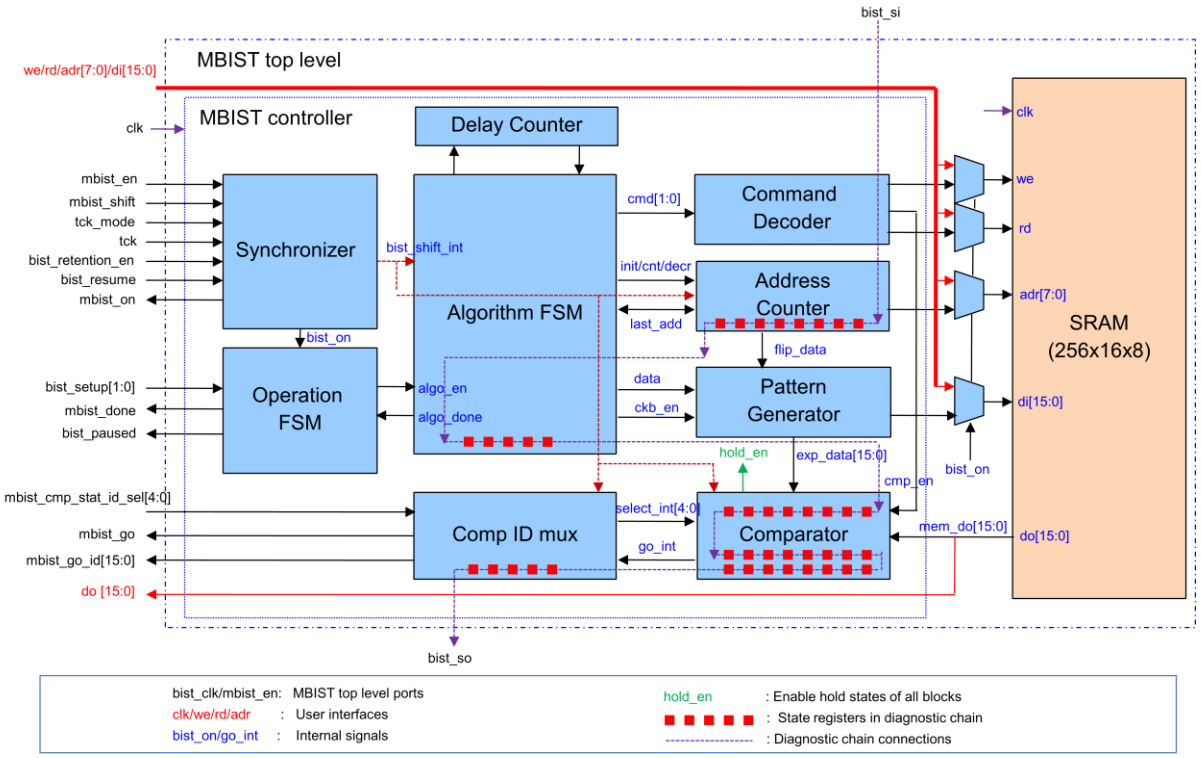


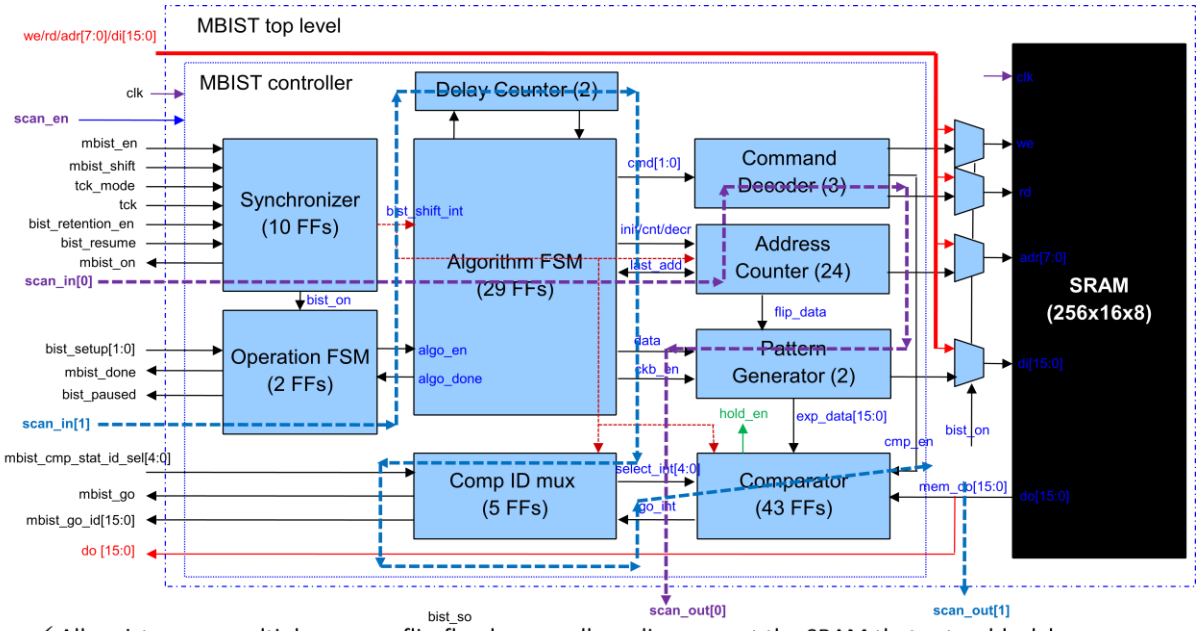
Figure 3.2

The function of each block are as follows:

Block	Function
Synchronizer	This module synchronizeres input signals from top-level
Operation FSM	Controls the overall operations of the BIST
Algorithm FSM	Implements all defined test algorithms (MARCH RAW) to detect failures in memory
Command Decoder	Receive the command from Algorithm and decode it to acess the memory (nop, read, write..), and the compare enable signal
Address Counter	This module counts through the Address space of the memory beging tested
Pattern Generator	Generates the data and expected data
Comp ID mux	Select the comparator, the output fail/pass
Comparator	Comparator the output data from memory and expected data patterns
Delay Counter	Create delay between the phases of algorithm

SRAM (256x16x8)	Synthesizable 1P SRAM modeling

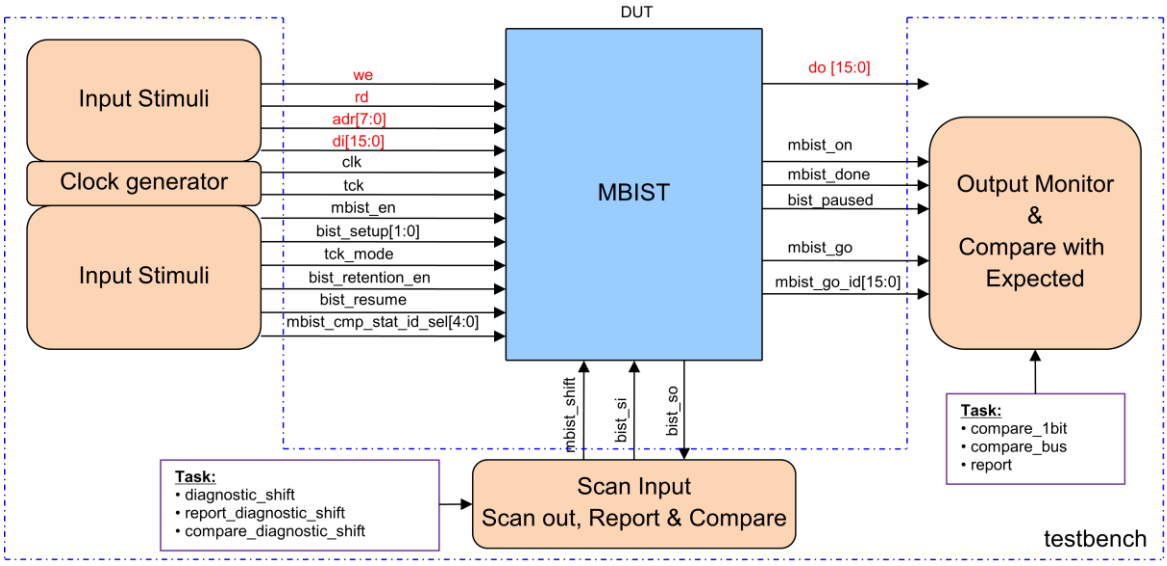
The scan in the design also are implemented for DFT, because the PDK doesn't support the scan cell, so all registers are multiplexer scan flip-flop by manually coding, except the SRAM block that set as black box.



✓ All registers are multiplexer scan flip-flop by manully coding, except the SRAM that set as black box
✓ Two scan chains: **chain0** with **68** FFs, **chain1** with **54** FFs

3.2 Testbench for Functional simulation

The testbench includes stimulus and test vectors as bellows:



The function of testbenh are:

- ✓ Outputs are compared with expected and report as failed if any missing between them
- ✓ The diagnostic chain are shift-out and compare with expected about: algorithm step, address, failing bit positions
- ✓ The input depend on each vectors and each operating modes.
- ✓ The faults in memory is modeled by assertion the memory data to stuck-at 0 or 1
- ✓ Used **+define+PRE_SIM** for pre-simulation

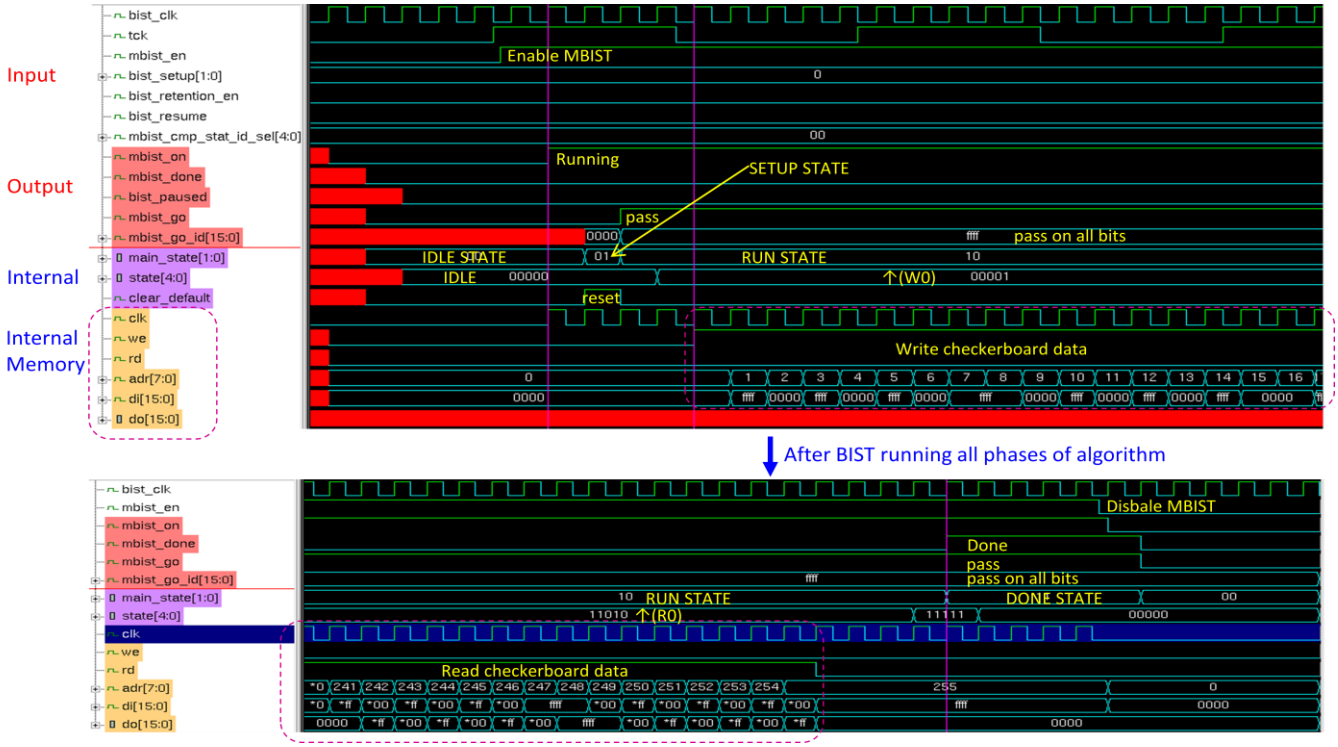
There are 6 vectors to test MBIST with different modes as follows:

- **vec_1**
  - ✓ **Non Stop-on-Fail mode, test result passed with no fault assertion**
- **vec_2**
  - ✓ **Nop Stop-on-Fail mode, test result failed with fault assertion as stuck-at data 0/1**
- **vec_3**
  - ✓ **Stop-on-Fail mode, test result failed at first fault**
  - ✓ **Scan out the diagnostic chain to identify the source of physical failure in the memory (algorithm step, address, failing bit position, and so forth)**
- **vec_4**
  - ✓ **Stop-on-Fail mode, failed at N-th errors**
  - ✓ **Scan in the number of Errors**
  - ✓ **Run MBIST, test results failed at N-th errors**
  - ✓ **Scan out the diagnostic chain to identify the source of physical failure in the memory**
- **vec_5**
  - ✓ **Retention test mode, test results passed with 100 cycles retention in memory**
- **vec_6**
  - ✓ **Retention test mode, test results failed after data retention, scan out the diagnostic chain**

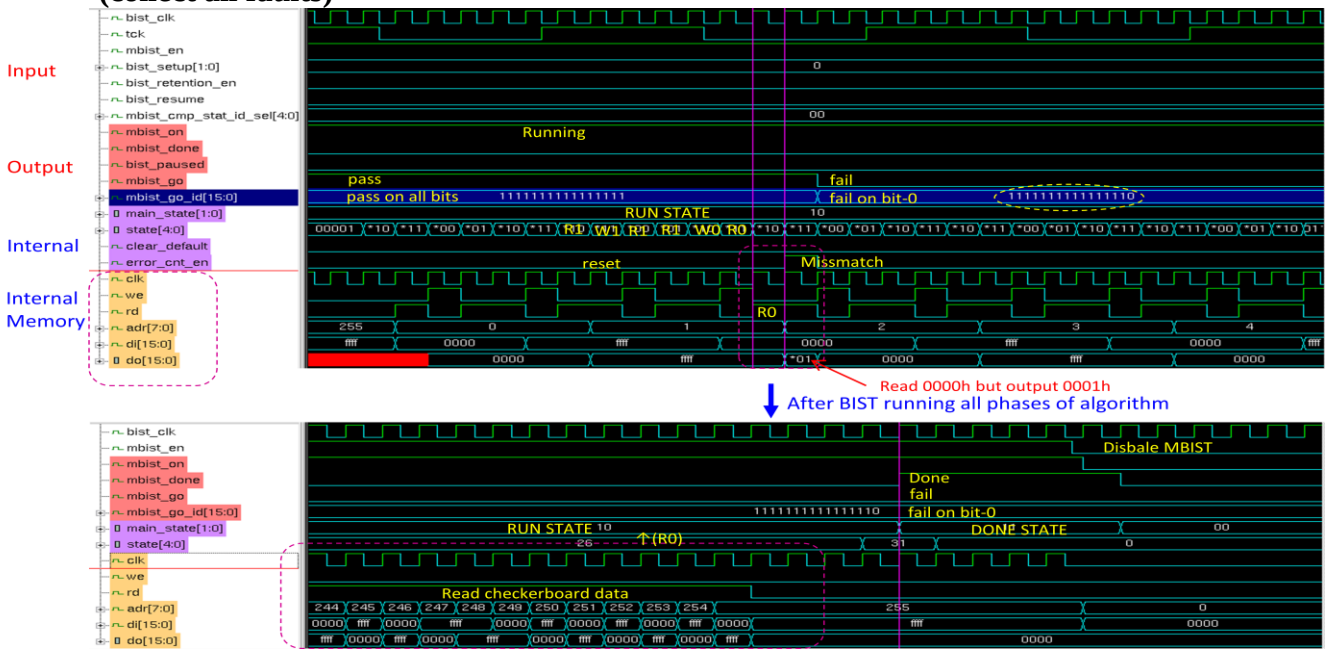
4. Funtional Simulation

This section shows the results of functional simulation step, with 6 vectors as section 3.

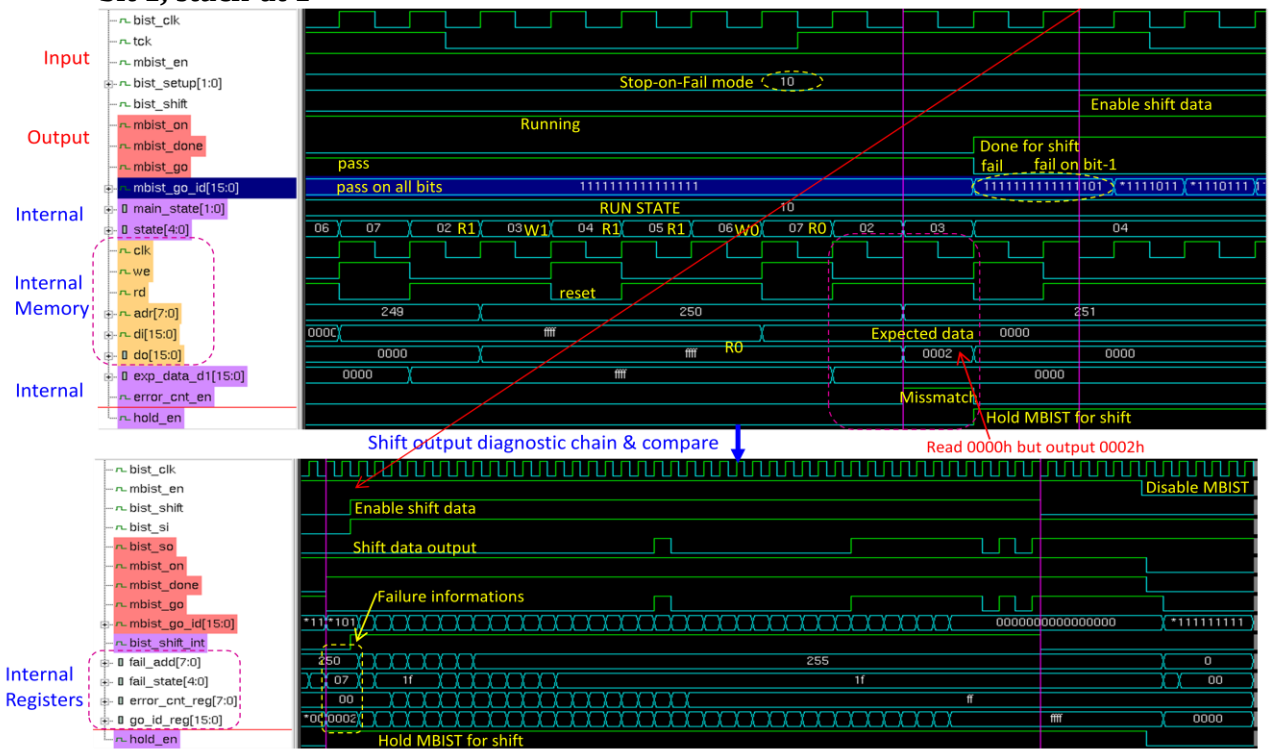
- Vector 1: **Non Stop-on-Fail mode with no faults assertion**



- Vector 2: **Non Stop-on-Fail mode with faults assertion at word 1, bit 0, stuck-at 1 (collect all faults)**



- Vector 3: **Stop-on-Fail mode, MBIST stop at first faults assertion at word 250, bit 1, stuck-at 1**

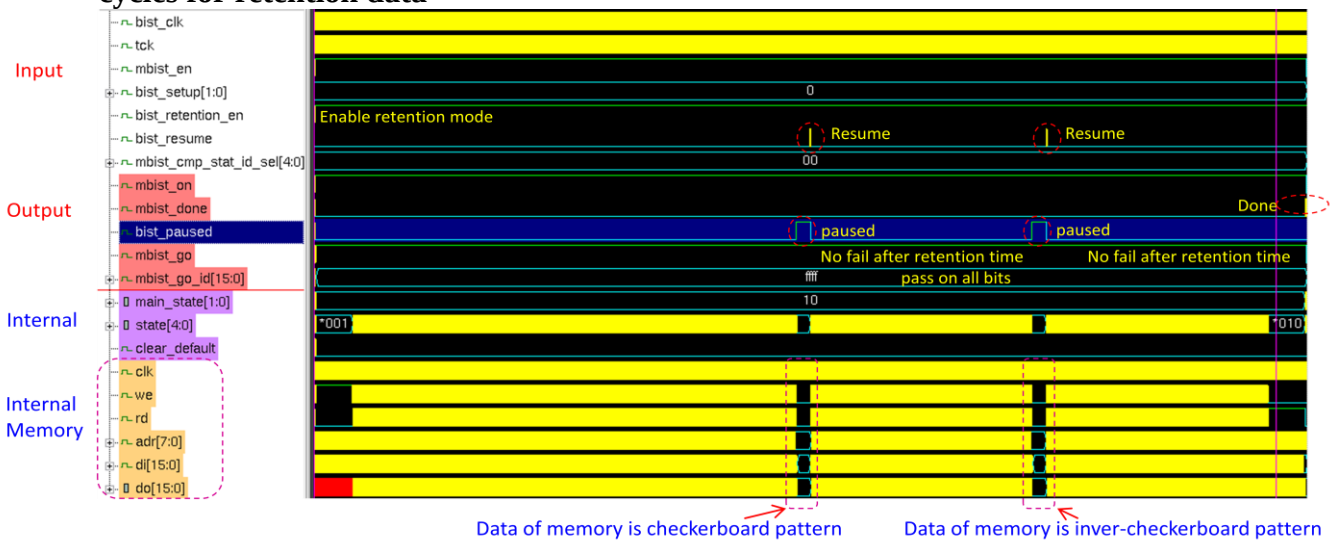


- Vector 4: **Stop-on-Fail mode, MBIST stop at 4th faults assertion at word 0, bit 15, stuck-at 0**

Log file report failure information as follows:

```
================= Register-chain output ====================
 Address[7:0]    : 00000000
 Sequence[5:0]   : 10011
 Bitmap[15:0]    : 1000000000000000
 ID_Sel[4:0]     : 00000
============================================================
```

- Vector 5: **Retention test mode, No fault assertion, paused memory with 100 cycles for retention data**

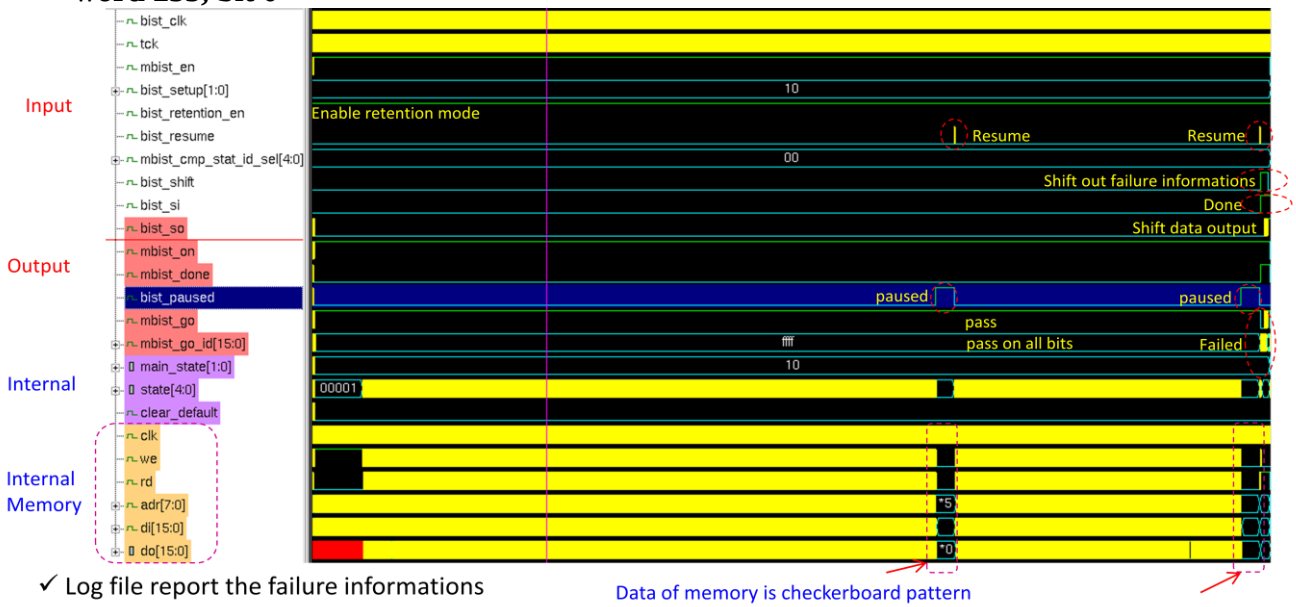


✓ When paused, MBIST don't access memory, and after resume, the MBIST will read data from memory to check the data retention

✓ The data retention time from the *bist_paused* go high to asserted *bist_resume* to high

✓ Log file for simulation results

- Vector 6: **Retention test mode, memory failed after data retention, fault assertion word 255, bit 0**



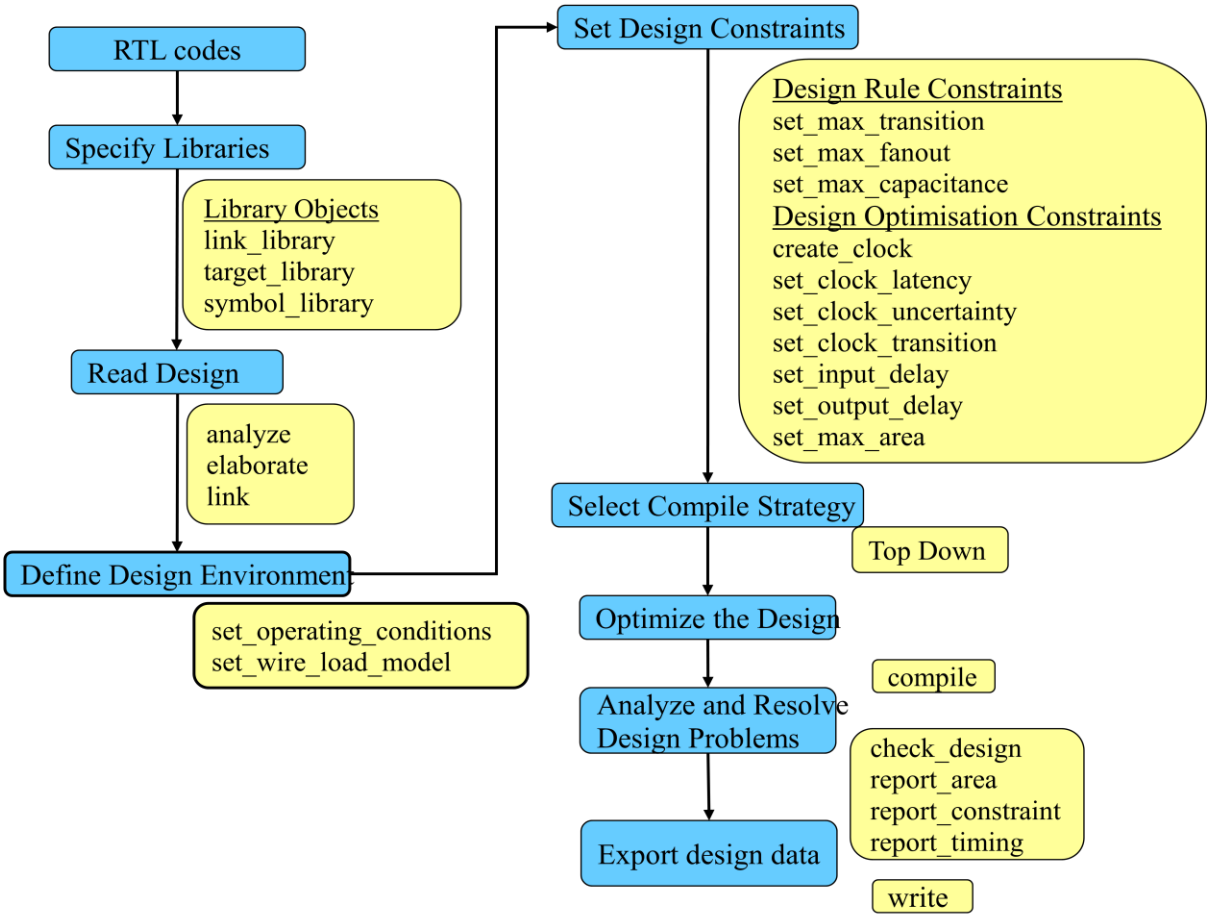
✓ Log file report the failure informations

Log file report failure information as follows:

```
================= Register-chain output ====================
 Address[7:0]    : 11111111
 Sequence[5:0]   : 10100
 Bitmap[15:0]    : 0000000000000001
 ID_Sel[4:0]     : 00000
============================================================
```

5. Logic Synthesis

The design flow of our design for logic synthesis is shown below figure:



5.1 Synthesis Results

After preparing the command file (TCL language) for design compiler, the results of logic synthesis as bellows:

- **Top Level Design**

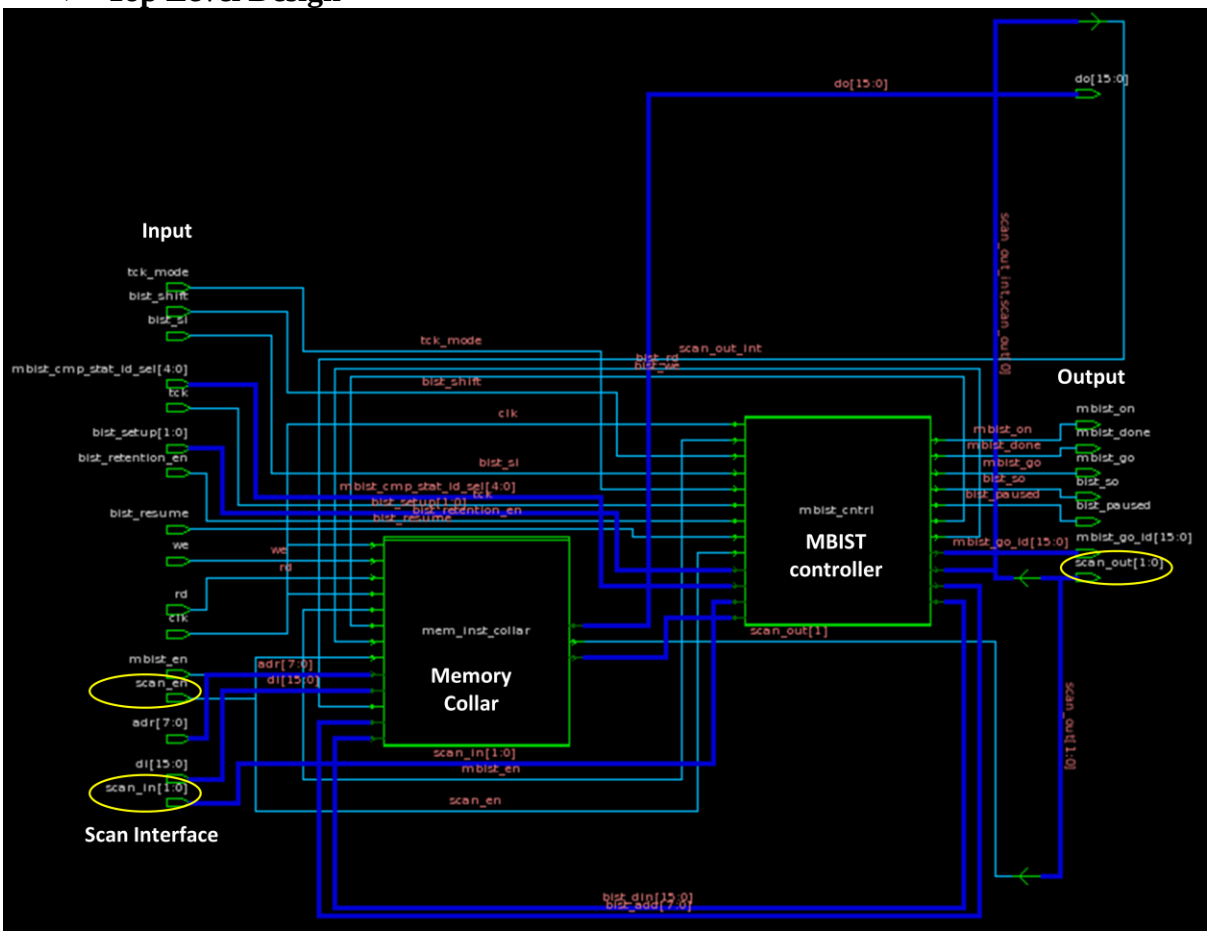


Figure 5.1

- **MBIST Controller**

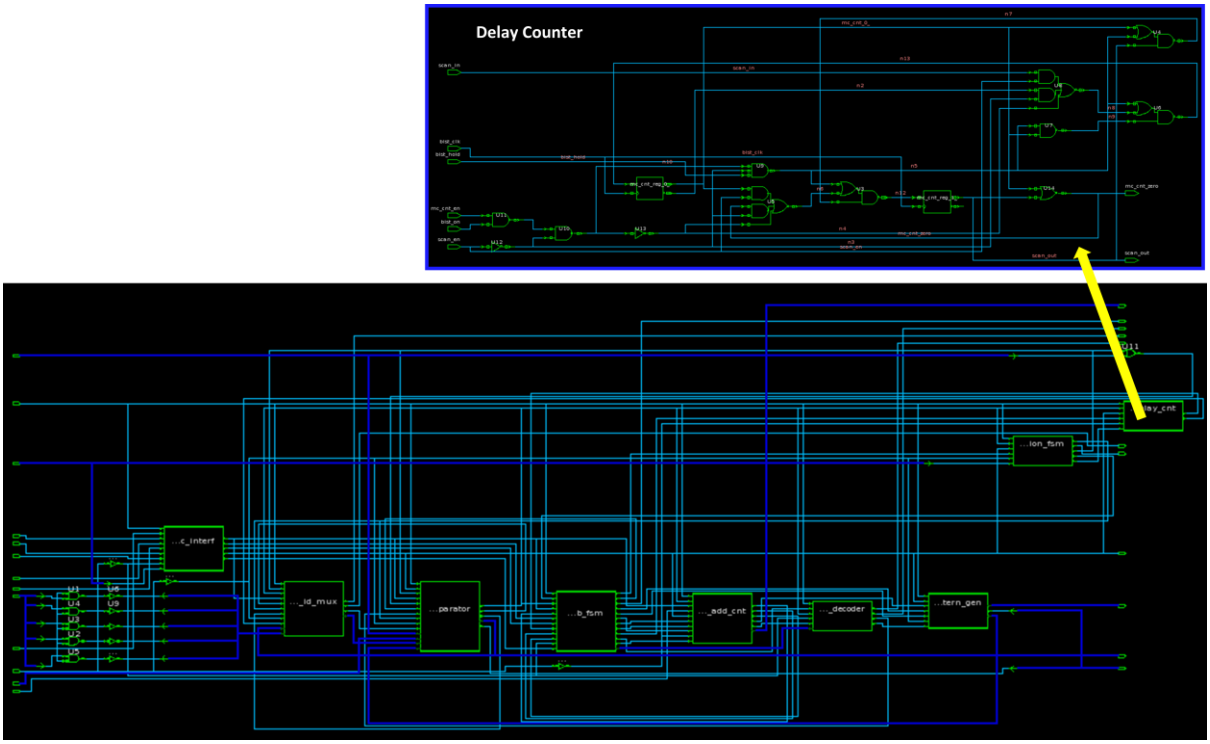


Figure 5.2

❖ **Synchronizer**

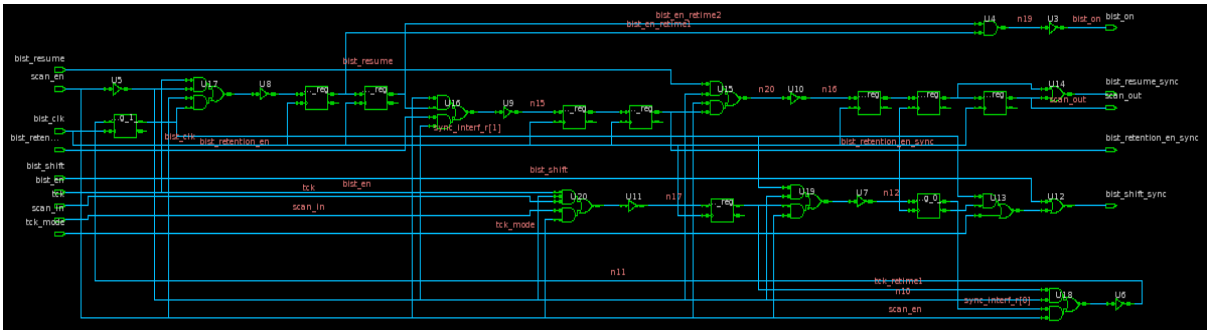


Figure 5.3

❖ **Comparator**

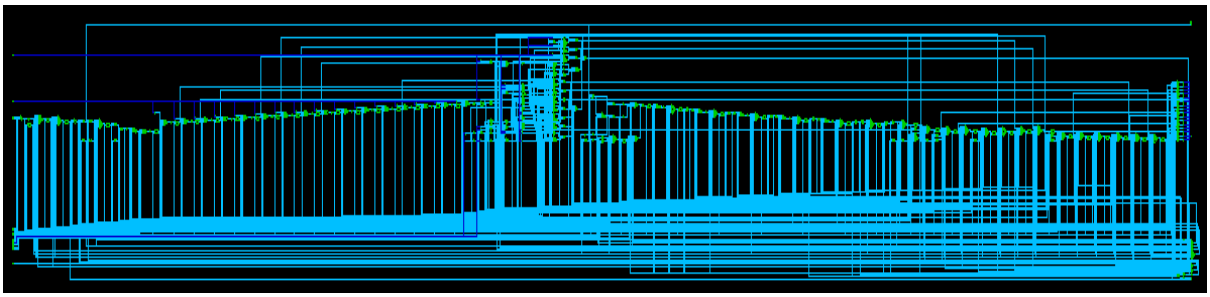


Figure 5.4

❖ **Address Counter**

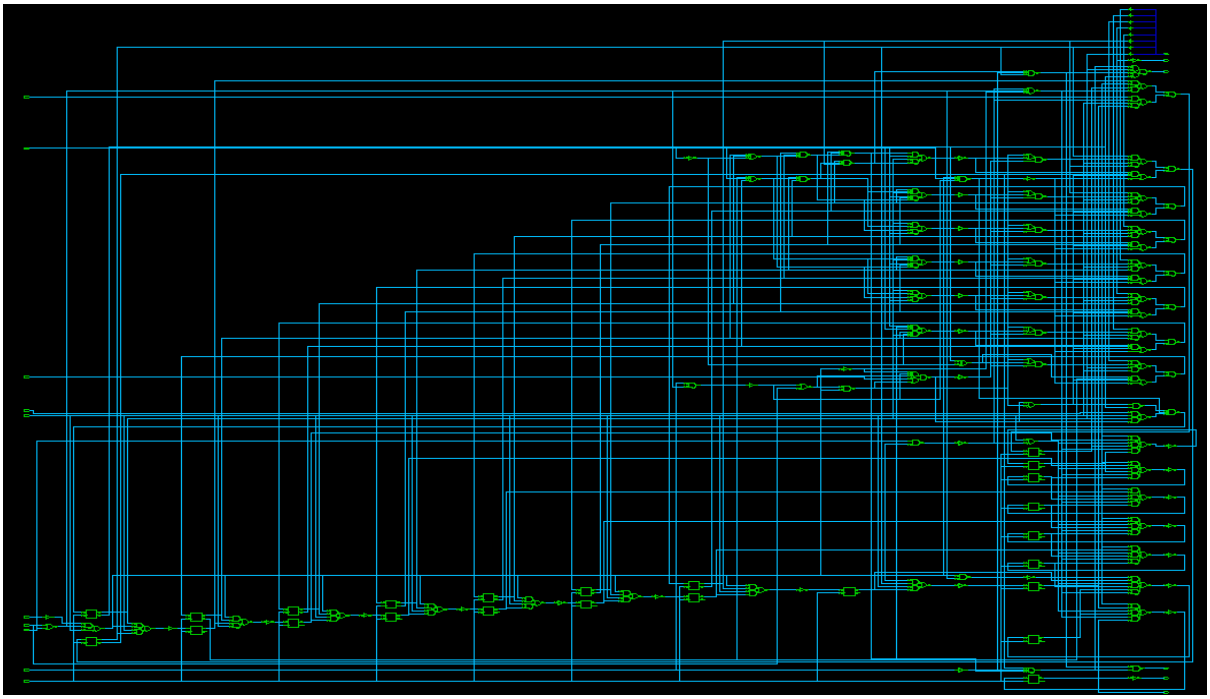


Figure 5.5

❖ Algorithm FSM

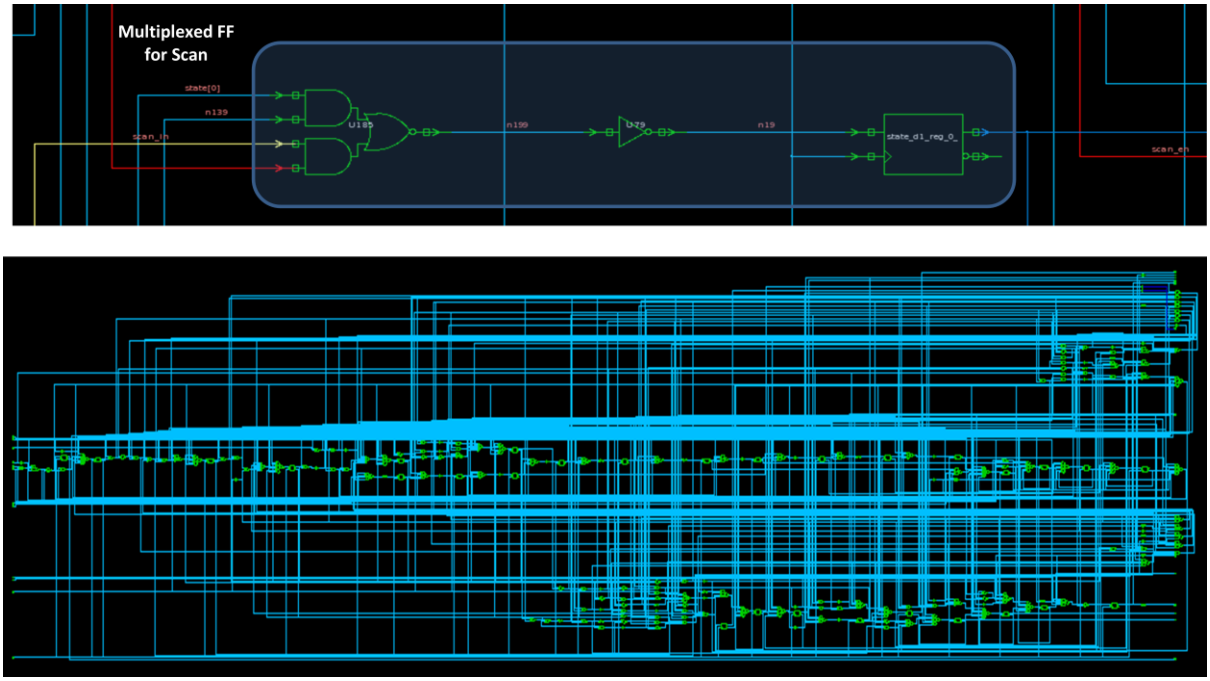


Figure 5.6

5.2 Synthesis Report

❖ Area report

```
****************************************
Report : area
Design : ram_256x16x8_mbist_top
Version: D-2010.03
Date   : Mon Dec 19 15:56:25 2011
****************************************

Library(s) Used:

    anam (File: /Tools/Library/ANAM_0.25um/CNU_SYN_LIB/ci025a_02.db)

Number of ports:              83
Number of nets:              126
Number of cells:               2
Number of references:          2

Combinational area:      18635.399752
Noncombinational area:   18206.200808
Net Interconnect area:     undefined  (No wire load specified)

Total cell area:         36841.600560
Total area:                undefined
1
```

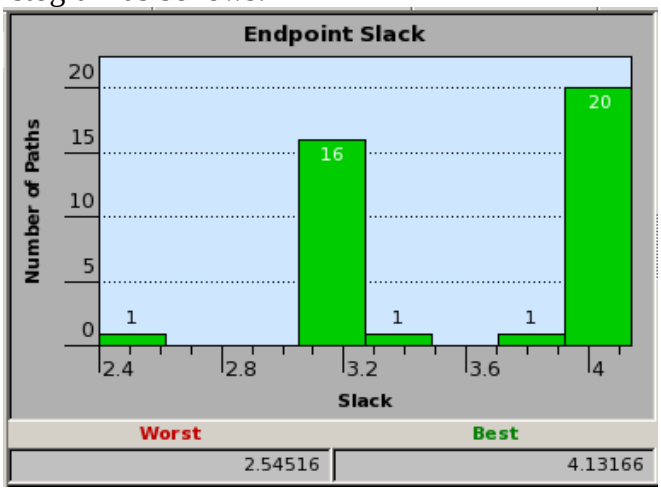
❖ Clock Report

```
****************************************
Report : clocks
Design : ram_256x16x8_mbist_top
Version: D-2010.03
Date   : Mon Dec 19 15:56:28 2011
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
------------------------------------------------------------------------------
clk             10.00   {0 5}               d         {clk}
------------------------------------------------------------------------------
1
```

❖ Timing report (max delay)

```
****************************************
Report : timing
        -path full
        -delay max
        -nworst 50
        -max_paths 50
        -sort_by slack
Design : ram_256x16x8_mbist_top
Version: D-2010.03
Date   : Mon Dec 19 15:56:25 2011
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: TYPICAL   Library: anam
Wire Load Model Mode: enclosed

  Startpoint: mbist_cntrl/mbist_operation_fsm/main_state_reg[1]/CP
              (internal path startpoint clocked by clk)
  Endpoint: mbist_go (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                                    Incr       Path
  -----------------------------------------------------------------------------
  clock clk (rise edge)                                    0.00       0.00
  clock network delay (ideal)                              0.30       0.30
  input external delay                                     0.00       0.30 r
  mbist_cntrl/mbist_operation_fsm/main_state_reg[1]/CP (ADFNTNB)
                                                           0.00 #     0.30 r
  mbist_cntrl/mbist_operation_fsm/main_state_reg[1]/Q (ADFNTNB)
                                                           0.63       0.93 r
  mbist_cntrl/mbist_operation_fsm/U18/ZN (ANR02D1)         0.65       1.58 f
  mbist_cntrl/mbist_operation_fsm/setup (ram_256x16x8_mbist_operation_fsm)
                                                           0.00       1.58 f
  mbist_cntrl/mbist_comparator/clear (ram_256x16x8_mbist_comparator)
                                                           0.00       1.58 f
  mbist_cntrl/mbist_comparator/U170/ZN (ANR02D1)           0.08       1.66 r
  mbist_cntrl/mbist_comparator/go_id[12] (ram_256x16x8_mbist_comparator)
                                                           0.00       1.66 r
  mbist_cntrl/mbist_cmp_stat_id_mux/cmp_stat_id[13] (ram_256x16x8_mbist_cmp_stat_id_mux)
                                                           0.00       1.66 r
  mbist_cntrl/mbist_cmp_stat_id_mux/U44/ZN (AAO01D1)       0.15       1.81 f
  mbist_cntrl/mbist_cmp_stat_id_mux/U43/ZN (AAO04D1)       0.11       1.91 r
  mbist_cntrl/mbist_cmp_stat_id_mux/U42/ZN (AAO05D1)       0.22       2.13 f
  mbist_cntrl/mbist_cmp_stat_id_mux/U41/ZN (AOA06D1)       0.12       2.25 r
  mbist_cntrl/mbist_cmp_stat_id_mux/cmp_stat (ram_256x16x8_mbist_cmp_stat_id_mux)
                                                           0.00       2.25 r
  mbist_cntrl/mbist_go (ram_256x16x8_mbist_cntrl)          0.00       2.25 r
  mbist_go (out)                                           0.00       2.25 r
  data arrival time                                                   2.25

  clock clk (rise edge)                                   10.00      10.00
  clock network delay (ideal)                              0.30      10.30
  clock uncertainty                                       -0.50       9.80
  output external delay                                   -5.00       4.80
  data required time                                                  4.80
  -----------------------------------------------------------------------------
  data required time                                                  4.80
  data arrival time                                                  -2.25
  -----------------------------------------------------------------------------
  slack (MET)                                                         2.55
```

Report timing slack histogram as bellows:



❖ Power Report

```
****************************************
Report : power
        -analysis_effort low
Design : ram_256x16x8_mbist_top
Version: D-2010.03
Date   : Mon Dec 19 15:56:28 2011
****************************************


Library(s) Used:

    anam (File: /Tools/Library/ANAM_0.25um/CNU_SYN_LIB/ci025a_02.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: TYPICAL   Library: anam
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------


Global Operating Voltage = 2.5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = Unitless


  Cell Internal Power  =   0.0000 uW    (0%)
  Net Switching Power  =   8.1186 mW  (100%)
                         ---------
Total Dynamic Power    =   8.1186 mW  (100%)

Cell Leakage Power     =   0.0000

1
```

6. Pre-layout simulation

This section shows the results of pre-simulation , with 6 vectors as section 3. The gate level netlist from Logic synthesis are used with backannotation

❖ Simulation log files

```
# vcs involke
vcs -R +compsdf +define+PRE_SIM+SDF+vec_$1 -f netlist.inc -v /Tools/Library/ANAM_0.25um/CNU_SIM_LIB/Verilog/ci025a_02.v
 -l log/ram_256x16x8_mbist_$1\.log +neg_tchk +maxdelays +v2k
Top Level Modules:
       testbench
TimeScale is 10 ps / 1 ps

   ***   $sdf_annotate() version 1.2R
   ***    SDF file: "ram_256x16x8_mbist_top.sdf"
   ***    Annotation scope: testbench.DUT
   ***    No MTM selection argument specified
   ***    No SCALE FACTORS argument specified
   ***    No SCALE TYPE argument specified
   ***    MTM selection defaulted to "TOOL_CONTROL":
               (+maxdelays compiled, MAXIMUM delays selected)
   ***    SCALE FACTORS defaulted to "1.0:1.0:1.0":
   ***    SCALE TYPE defaulted to: "FROM_MTM"
   ***    Turnoff delay: "FROM_FILE"
   ***    Approximation (mipd) policy: "MAXIMUM"

   ***    SDF annotation begin: Wed Nov 30 23:20:50 2011


SDF Info: +pulse_r/100, +pulse_e/100 in effect

          Total errors: 0
          Total warnings: 0
   ***    SDF annotation completed: Wed Nov 30 23:20:51 2011

Doing SDF annotation ...... Done
=======================================
 Testing BIST with Non Stop-on-fail
 No faults assertion
 Waiting for BIST run...
=======================================
=======================================
 SUMMARY REPORT
 Test Result    : PASS
=======================================
$finish called from file "../data/testbench/vec_1.v", line 51.
$finish at simulation time            67030000
          V C S   S i m u l a t i o n   R e p o r t
```

❖ **Vec_1: Non Stop-on-Fail mode with no faults assertion**

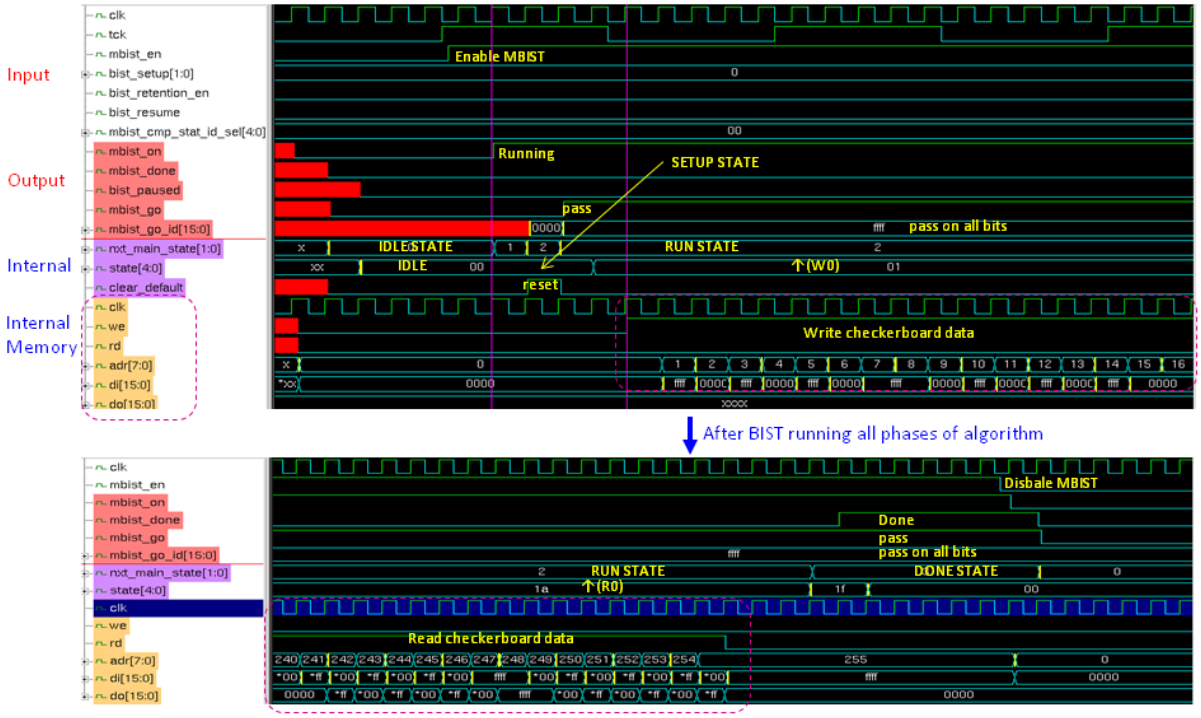


Figure 6.1

❖ **Vec_2: Non Stop-on-Fail mode with faults assertion at word 1, bit 0, stuck-at 1 (collect all faults)**

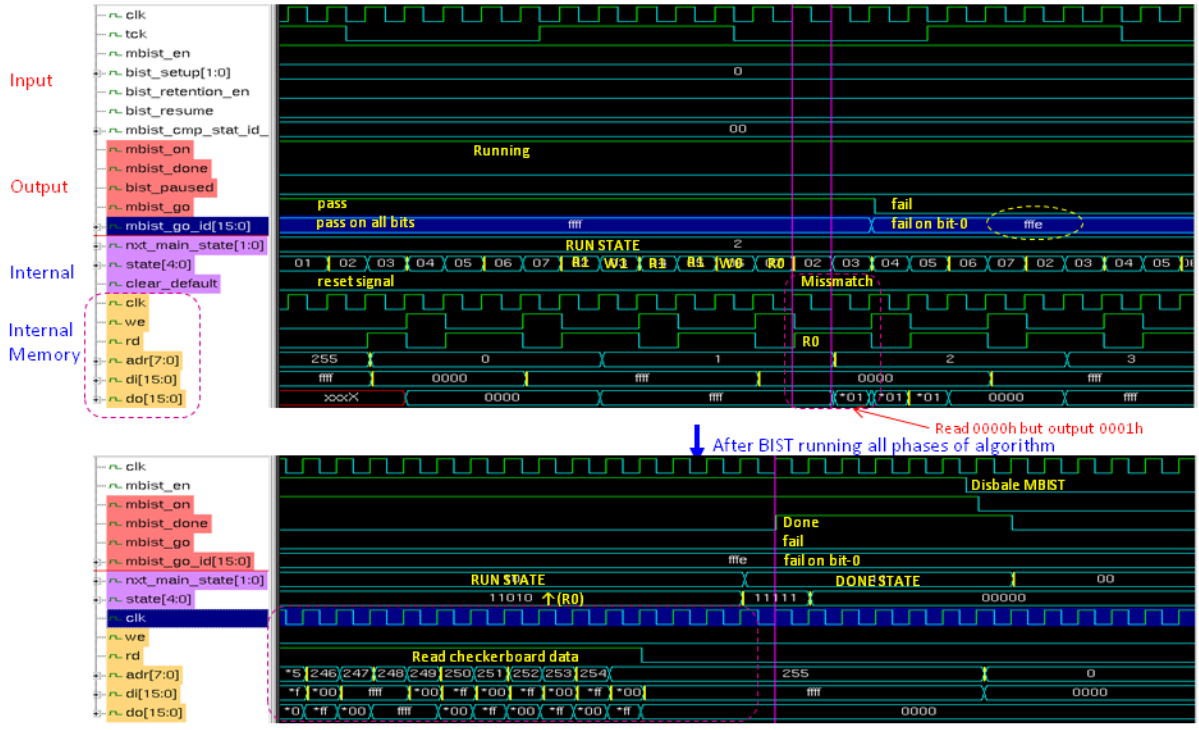


Figure 6.2

❖ **Vec_3: Stop-on-Fail mode, MBIST stop at first faults assertion at word 250, bit 1, stuck-at 1**

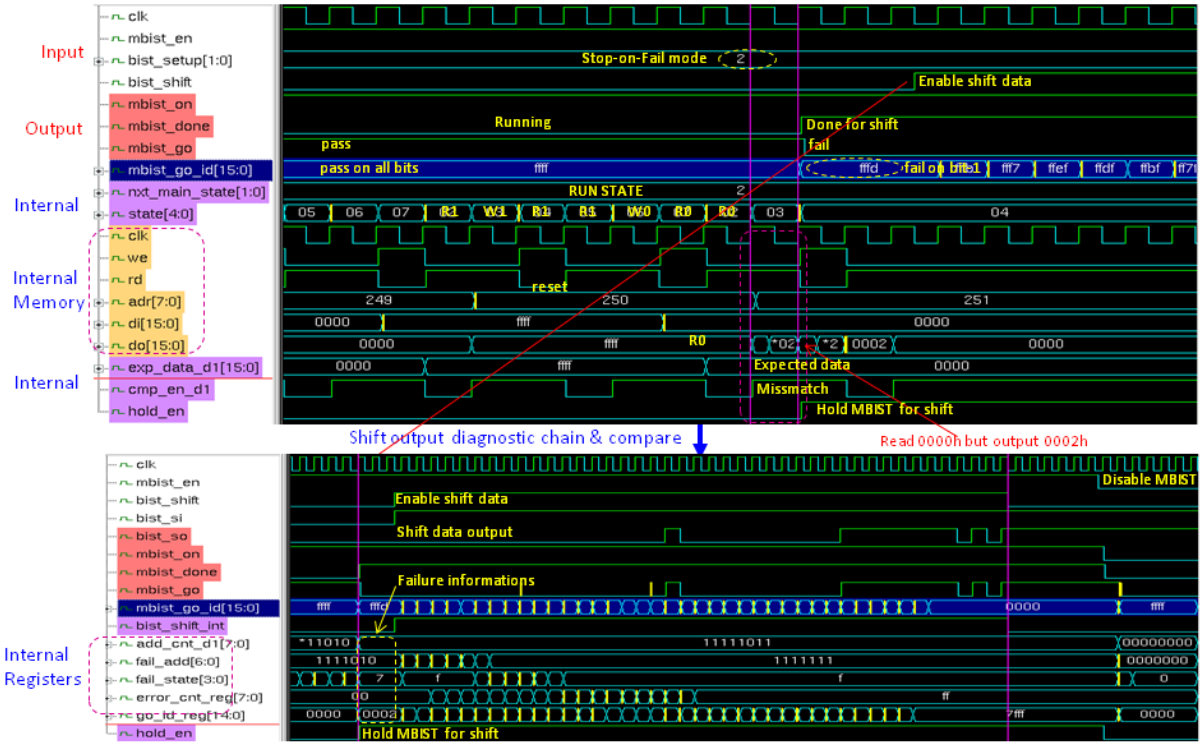


Figure 6.3

❖ Vec_4: Stop-on-Fail mode, MBIST stop at 4th faults assertion at word 0, bit 15, stuck-at 1

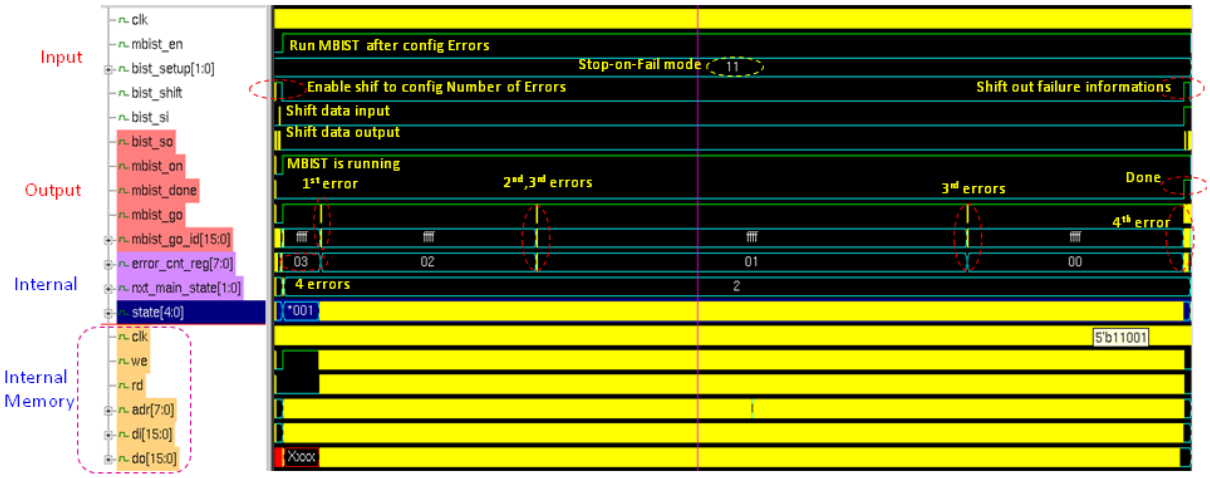


Figure 6.4

Log file report the failure information

```
==========================================
SUMMARY REPORT
Test Result     : PASS
==========================================
```

```
================= Register-chain output ====================
Address[7:0]    : 00000000
Sequence[5:0]   : 11001
Bitmap[15:0]    : 1000000000000000
ID_Sel[4:0]     : 00000
============================================================
```

❖ **Vec_5: Retention test mode, No fault assertion, paused memory with 100 cycles for retention data**

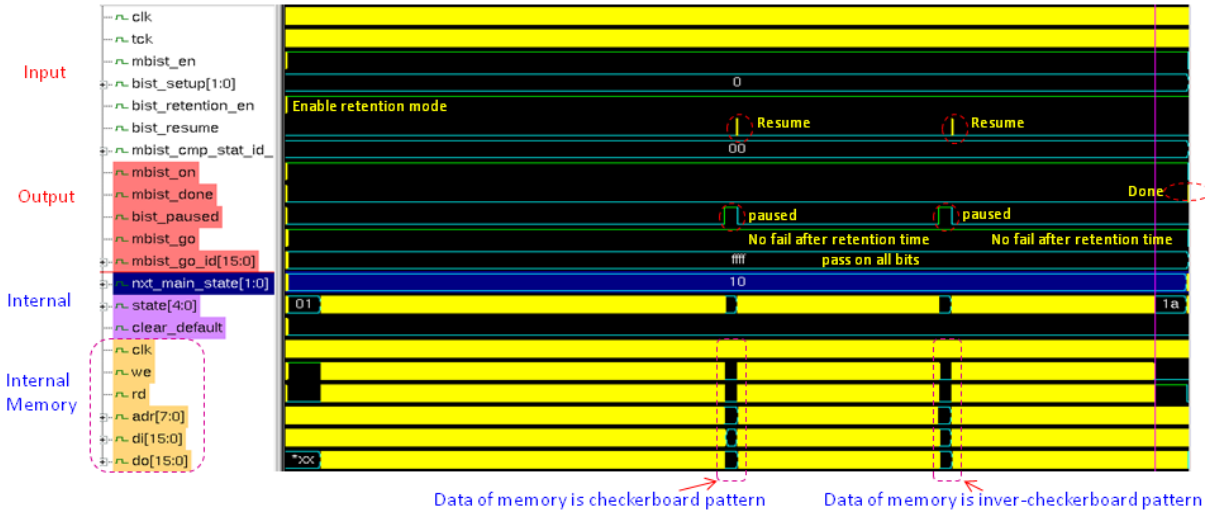


Figure 6.5

- **When paused, MBIST don't access memory, and after resume, the MBIST will read data from memory to check the ability data retention**
- **The data retention time from the *bist_paused* go high to asserted *bist_resume* to high**

- **Log file for simulation results**

```
Doing SDF annotation ...... Done
=========================================
 Testing BIST with Retention mode
 No faults assertion
 Waiting for BIST run...
=========================================
=========================================
 SUMMARY REPORT
 Test Result    : PASS
=========================================
```

❖ **Vec_6: Retention test mode, memory failed after data retention, fault assertion word 255, bit 0**

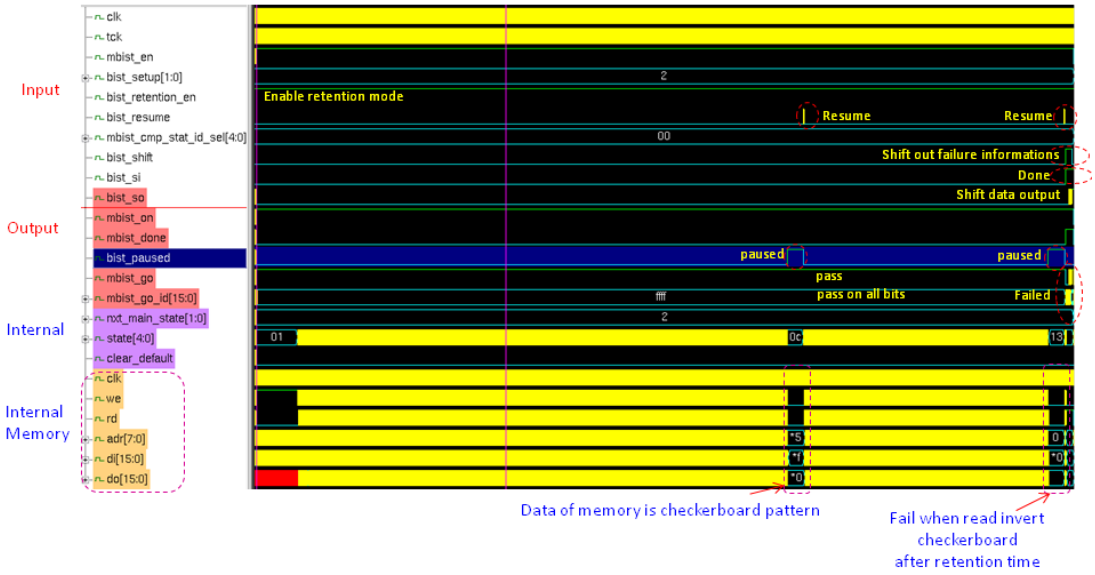


Figure 6.6

Log file report the failure information

```
================== Register-chain output ====================
 Address[7:0]    : 11111111
 Sequence[5:0]   : 10100
 Bitmap[15:0]    : 0000000000000001
 ID_Sel[4:0]     : 00000
=============================================================
===========================================
 SUMMARY REPORT
 Test Result     : PASS
===========================================
```

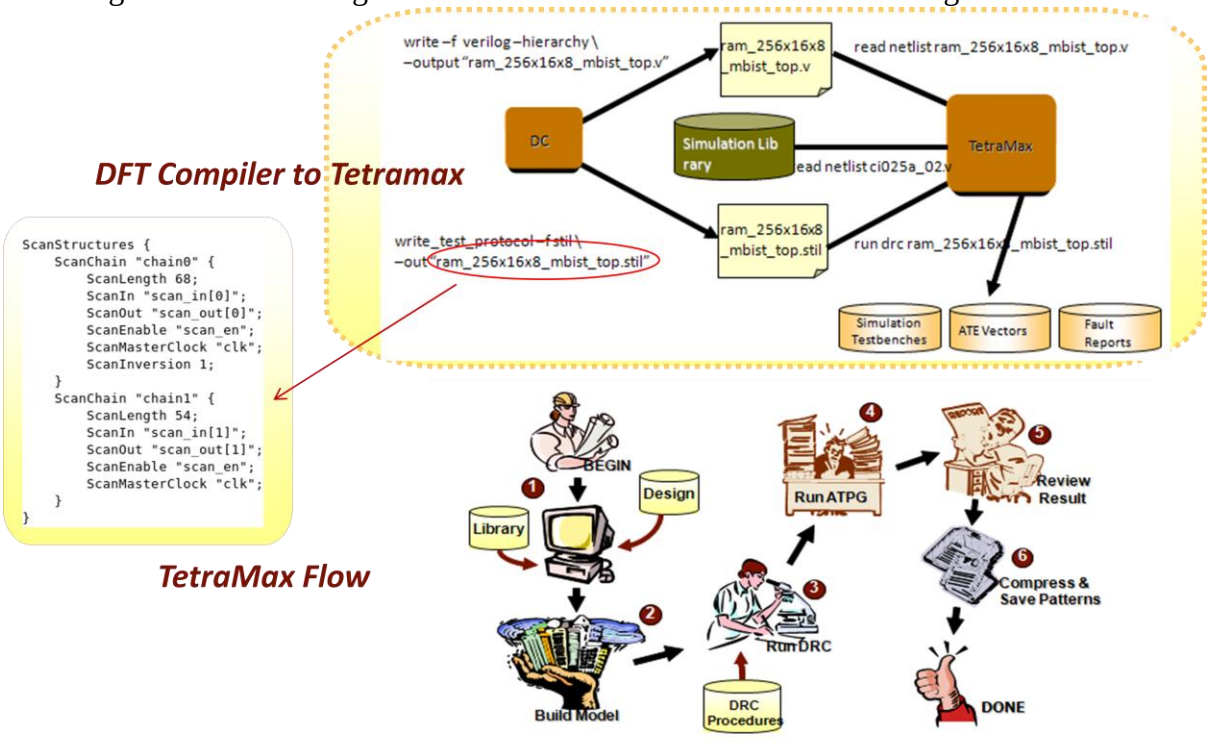
Summary of Simulation Results

Vector	Functional Simulation	Pre-layout Simulation
vec_1	PASSED	PASSED
vec_2	PASSED	PASSED
vec_3	PASSED	PASSED
vec_4	PASSED	PASSED
vec_5	PASSED	PASSED
vec_6	PASSED	PASSED

Table 6.1

7. ATPG with Tetramax

The design flow of our design for ATPG with Tetramax is shown below figure:



The output gate-level netlist from logic synthesis are used for ATPG, also the STIL file for scan protocol was created for input of Tetramax, after get all files, we build the command file for run the Tetramax

- **Command and log files**

Command file

```
// Setting to log message
   set message log ./log/atpg.log -replace
   set message display
   set message -transcript_comments
   set message -level standard

// Read model into TetraMax
   read netlist /Tools/Library/ANAM_0.25um/CNU_SIM
   read netlist /home/Sun29/2010315435/MBIST/Outpu

   report modules -summary
   report modules -error

// Specify RAM/ROM block - St non-DFT as black box
   set rules B12 warning
   set build -black_box ram_256x16x8

// Option to run build model
   set build -NONet_connections_change_netlist
   set build -Merge Tied_gates_with_pin_loss
   set build -DElete_unused_gates

// Run bulid ATPG design Model
   run build_model ram_256x16x8_mbist_top

// Read STIL - Design Rules Checking

   add scan enable 1 scan_en

   set drc -trace
   run drc /home/Sun29/2010315435/MBIST/Output/ram
   report violations -all >> ./log/rules.rpt

// Specify ATPG settings
   add faults -all             //add faults list
   set patterns internal       //select the patter
   set faults -atpg_effectiveness -fault_coverage
   set atpg -abort_limit 10
   set atpg -coverage 100
   set faults -summary verbose
   set atpg -capture_cycles 2  //Fast-scan ATPG
   set atpg -full_seq_atpg     //Full-scan ATPG

   run atpg -auto_compression

   report_summaries > ./log//summaries.rpt
```

Log file

```
set drc -trace
run drc /home/Sun29/2010315435/MBIST/Output/ram_256x16x8_mbist_top.stil
 ------------------------------------------------------------------------
 Begin scan design rules checking...
 ------------------------------------------------------------------------
 Begin reading test protocol file /home/Sun29/2010315435/MBIST/Output/ram_256x16x8_mbist_top.stil...
 End parsing STIL file /home/Sun29/2010315435/MBIST/Output/ram_256x16x8_mbist_top.stil with 0 errors.
 Test protocol file reading completed, CPU time=0.00 sec.
 ------------------------------------------------------------------------
 Begin simulating test protocol procedures...
 Test protocol simulation completed, CPU time=0.01 sec.
 ------------------------------------------------------------------------
 Begin scan chain operation checking...
 Begin sensitization trace for chain = chain0  sci = scan_in[0]  sco = scan_out[0].
 Chain chain0 successfully traced with 68 scan_cells.
 Begin sensitization trace for chain = chain1  sci = scan_in[1]  sco = scan_out[1].
 Chain chain1 successfully traced with 54 scan_cells.
 DRC Summary Report
 ------------------------------------------------------------------------
 No violations occurred during DRC process.
 Design rules checking was successful, total CPU time=0.01 sec.
 ------------------------------------------------------------------------
 Begin Full-Sequential ATPG for 1302 uncollapsed faults ...
  --- abort limit : 10 seconds, NO BACKTRACK LIMIT
 ------------------------------------------------------------
 #patterns  #faults        #ATPG faults  test       process
 stored     detect/active  red/au/abort  coverage   CPU time
 ---------  -------------  ------------  --------  ----------
 67              0      0       0/656/0     79.22%         1.53
 End writing file 'ram_256x16x8_mbist_top_atpg_0.v' with 10 patterns, File_size = 24234, CPU_time = 0.0 sec.
 Verilog testbench top module name = ram_256x16x8_mbist_top_bist_testbench
 End writing file 'ram_256x16x8_mbist_top_atpg_1.v' with 10 patterns, File_size = 24252, CPU_time = 0.0 sec.
 Verilog testbench top module name = ram_256x16x8_mbist_top_bist_testbench
 End writing file 'ram_256x16x8_mbist_top_atpg_2.v' with 10 patterns, File_size = 24252, CPU_time = 0.0 sec.
write faults faultlist.lst -all -replace
 Write faults completed: 6272 faults were written into file "faultlist.lst".
```

The log file indicates that don't have any violatios occurred during DRC process, and the test pattern was created with 6272 faults.

❖ Tetramax reports

The reports for failt covertage and faults list as bellows:

TEST > report_summaries

```
    Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT       4964
  detected_by_simulation         DS     (3279)
  detected_by_implication        DI     (1685)
Possibly detected                PT          0
Undetectable                     UD          6
  undetectable-redundant         UR        (6)
ATPG untestable                  AU       1302
  atpg_untestable-not_detected   AN     (1302)
Not detected                     ND          0
-----------------------------------------------
total faults                              6272
test coverage                           79.22%
fault coverage                          79.15%
ATPG effectiveness                     100.00%
-----------------------------------------------
          Pattern Summary Report
-----------------------------------------------
#internal patterns                          67
    #basic_scan patterns                    67
-----------------------------------------------
```

Write faults

```
sa0   DI   mbist_cntrl/mbist_sync_interf/tck_retime1_reg/CP
sa1   DS   tck
sa1   --   mbist_cntrl/mbist_sync_interf/U20/A3
sa1   DS   mbist_cntrl/mbist_sync_interf/U20/A1
sa1   DS   mbist_cntrl/mbist_sync_interf/U20/A2
sa1   DS   mbist_cntrl/mbist_sync_interf/U20/B2
sa0   DS   mbist_cntrl/mbist_sync_interf/U20/A1
sa0   --   mbist_cntrl/mbist_sync_interf/U20/A2
sa0   --   mbist_cntrl/mbist_sync_interf/U20/A3
sa0   --   tck
sa1   DI   mbist_cntrl/mbist_sync_interf/tck_retime1_reg/Q
sa0   DI   mbist_cntrl/mbist_sync_interf/tck_retime1_reg/Q
sa0   DI   mbist_cntrl/mbist_sync_interf/U11/ZN
sa1   --   mbist_cntrl/mbist_sync_interf/U11/I
sa1   --   mbist_cntrl/mbist_sync_interf/U20/ZN
sa0   --   mbist_cntrl/mbist_sync_interf/tck_retime1_reg/D
sa1   DI   mbist_cntrl/mbist_sync_interf/U11/ZN
sa0   --   mbist_cntrl/mbist_sync_interf/U11/I
sa0   --   mbist_cntrl/mbist_sync_interf/U20/ZN
sa1   --   mbist_cntrl/mbist_sync_interf/tck_retime1_reg/D
sa1   DI   mbist_cntrl/mbist_sync_interf/U20/B1
sa0   DI   mbist_cntrl/mbist_sync_interf/U20/B1
```

The fault coverage about 79.22%, the fault coverage is low, because we set the SRAM as black box and the DFF are manually coding.

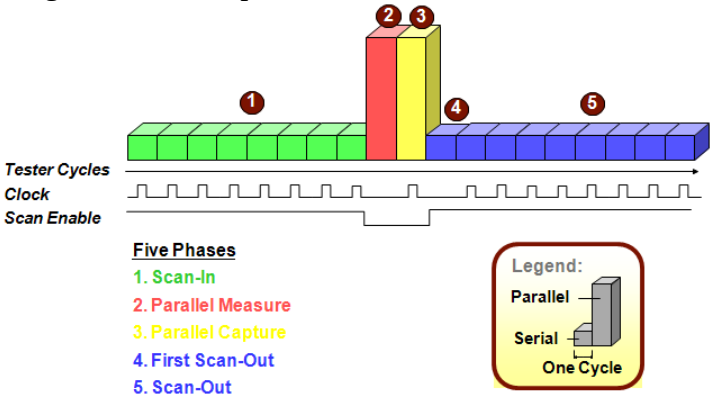
The patterns were generated as follows for ATPG simulation with VCS tools.

```
pattern = 0; // 200
LOAD0 = 68'b0011001100110011001100110011001100110011001100110011001100110011001100110011;
LOAD1 = 68'b0000000000000000001100110011001100110011001100110011001100110011001100110011;
UNLMSK[0] = 68'b0000000000000000000000000000000000000000000000000000000000000000000000000000;
UNLMSK[1] = 68'b0000000000000000000000000000000000000000000000000000000000000000000000000000;
#0 ->load_unload;
#6900; // 7100
ALLPIS = 44'b00000111111111101101011001010011011000001XX;
XPCT = 39'bXXXXXXXXXXXXXXXX000001100110011001100000;
MASK = 39'b000000000000000011111111111111111111111;
#0 ->capture;
#200; // 7300
```

8. ATPG Simulation

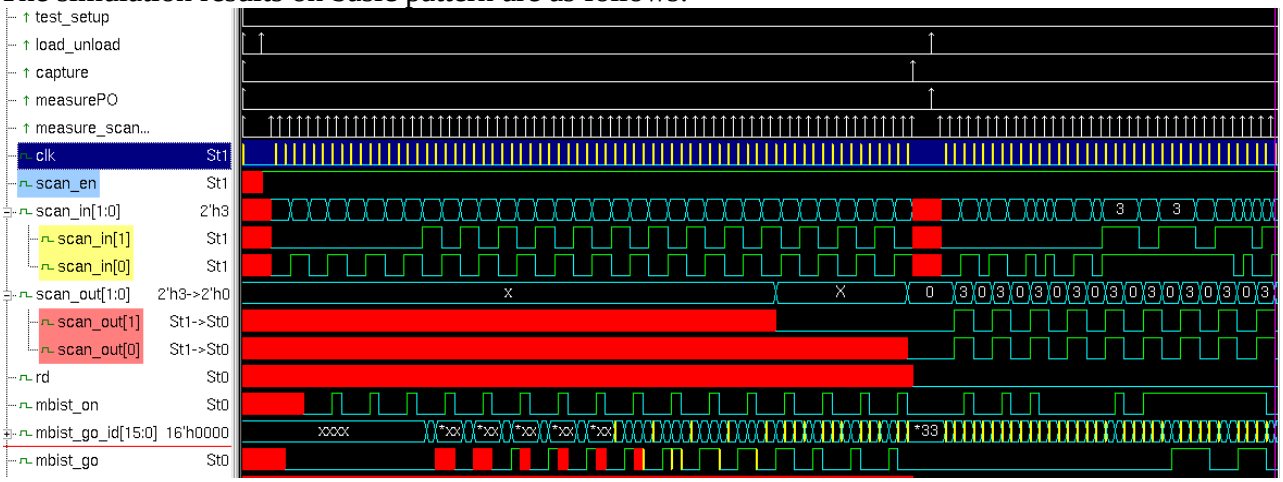
After save the pattern files from TetraMax, we simulation with gate-level netlist for verification.

The basic ATPG testing contains five phases are as follows:

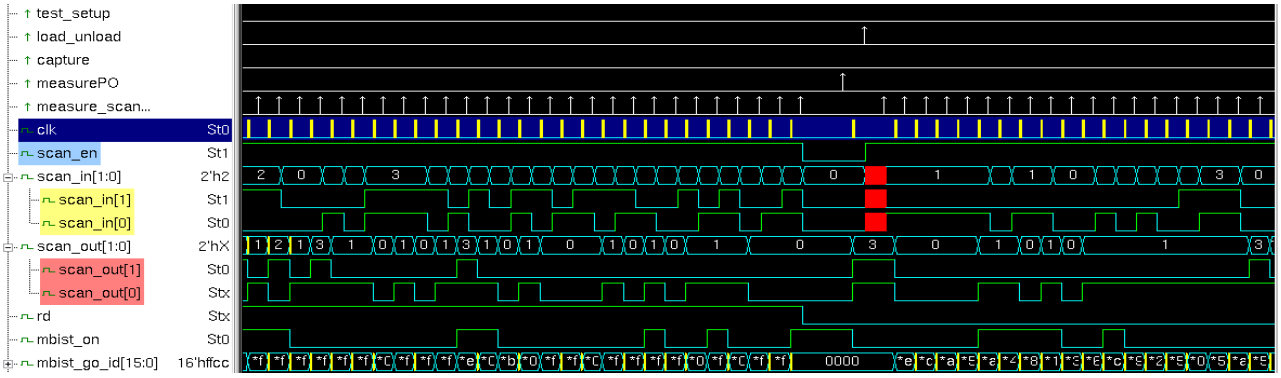


- First is scan-in the data into internal scan DFF
- Next phase, measure the primary output parallel
- Next phase, parallel capture the data into combination logic
- Next phase, fisrt scan-out with non-clocking
- Final phase, scan out the data with each clock cycle for each scan DFF.

The simulation results on basic pattern are as follows:



The simulation results on complex pattern with capture are as follows:



And the simulation log files for 67 patterns are PASSED

```
//            0.00 ns : Begin test_setup
//          200.00 ns : Begin patterns, first pattern = 0
//          200.00 ns : ...begin scan load for pattern 0
//        36100.00 ns : ...begin scan load for pattern 5
//        79000.00 ns : Simulation of 10 patterns completed with 0 errors

$finish called from file "../Tetramax/ram_256x16x8_mbist_top_atpg_0.v", line 657.
```

```
//            0.00 ns : Begin test_setup
//          200.00 ns : Begin patterns, first pattern = 60
//          200.00 ns : ...begin scan load for pattern 60
//        36200.00 ns : ...begin scan load for pattern 65
//        57500.00 ns : Simulation of 7 patterns completed with 0 errors

$finish called from file "../Tetramax/ram_256x16x8_mbist_top_atpg_6.v", line 609.
```

```
[2010315435@Sun38 Atpg_sim]$ grep errors log/*
log/ram_256x16x8_mbist_atpg_0.log://        79000.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_1.log://        79100.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_2.log://        79100.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_3.log://        79100.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_4.log://        79100.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_5.log://        79100.00 ns : Simulation of 10 patterns completed with 0 errors
log/ram_256x16x8_mbist_atpg_6.log://        57500.00 ns : Simulation of 7 patterns completed with 0 errors
```

Show that the gate-level netlist passed for back-end phase.

9. Place and Route

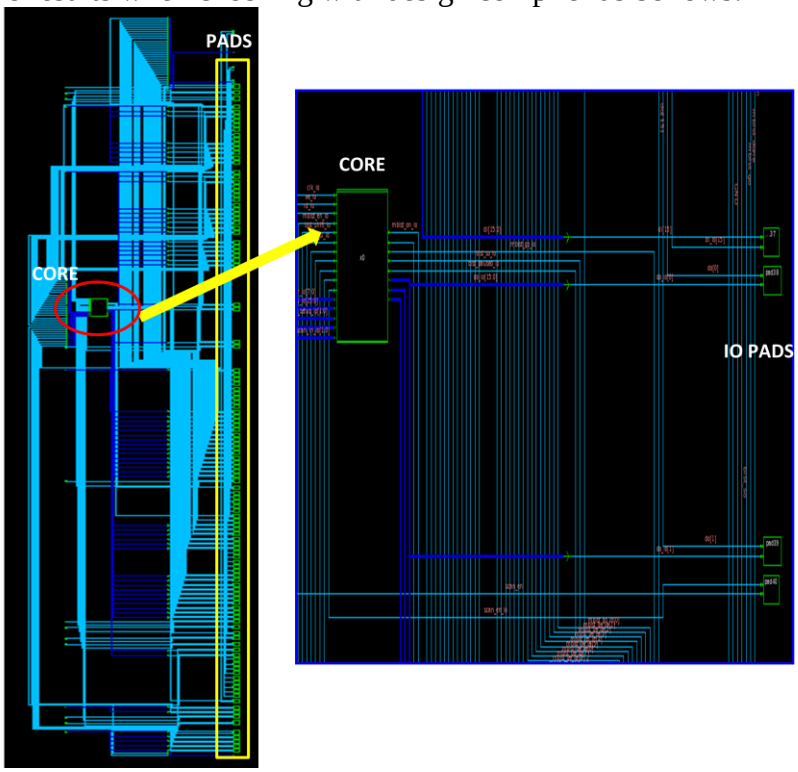
After the ATPG simulation finished, the gate-level neslit are used for P&R with Astro, However, the Gate level netlist don't the IO PAD, so we need to add the IO Pad for GLN, the P&R includes some step as follows:

❖ **<u>Step 1</u>: Adding IO PAD to the gate-level netlist & check with design compiler**

This step add the IO PAD to the gate level netlist, after finished, we check syntax and show the results in design compiler.

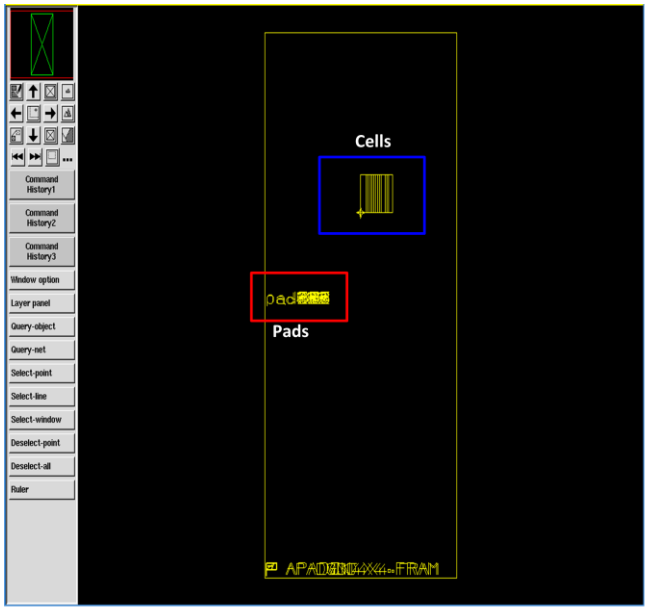
```
// left side
APADCK4X4 pad1 (.O(clk_io), .pad(clk));
APADIN4X4 pad2 (.O(we_io), .pad(we));
APADIN4X4 pad3 (.O(rd_io), .pad(rd));
APADIN4X4 pad4 (.O(adr_io[0]), .pad(adr[0]));
APADIN4X4 pad5 (.O(adr_io[1]), .pad(adr[1]));
APADIN4X4 pad6 (.O(adr_io[2]), .pad(adr[2]));

APADVDD4X4 pad7 (.vdd(VDD), .pad());
APADVDD4X4 pad8 (.vdd(VDD), .pad());
APADGND4X4 pad9 (.gnd(GND), .pad());
APADGND4X4 pad10 (.gnd(GND), .pad());
APADGND4X4 pad11 (.gnd(GND), .pad());

APADIN4X4 pad12 (.O(adr_io[3]), .pad(adr[3]));
APADIN4X4 pad13 (.O(adr_io[4]), .pad(adr[4]));
APADIN4X4 pad14 (.O(adr_io[5]), .pad(adr[5]));
APADIN4X4 pad15 (.O(adr_io[6]), .pad(adr[6]));
APADIN4X4 pad16 (.O(adr_io[7]), .pad(adr[7]));
APADIN4X4 pad17 (.O(di_io[0]), .pad(di[0]));
APADIN4X4 pad18 (.O(di_io[1]), .pad(di[1]));
APADIN4X4 pad19 (.O(di_io[2]), .pad(di[2]));
```

The results when checking with design compiler as bellows:

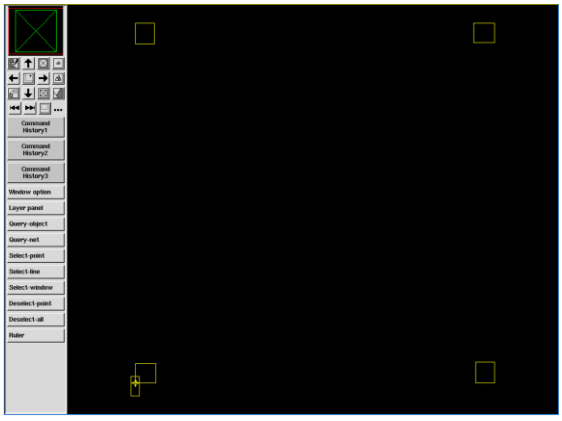


❖ **<u>Step 2</u>: Load Design, SDC, and TDF**

SDC file is the output of logic synthesis are loaded into Astro to set the design timing for the Astro, also the TDF are loaded to specify the pad location, the results as bellows:



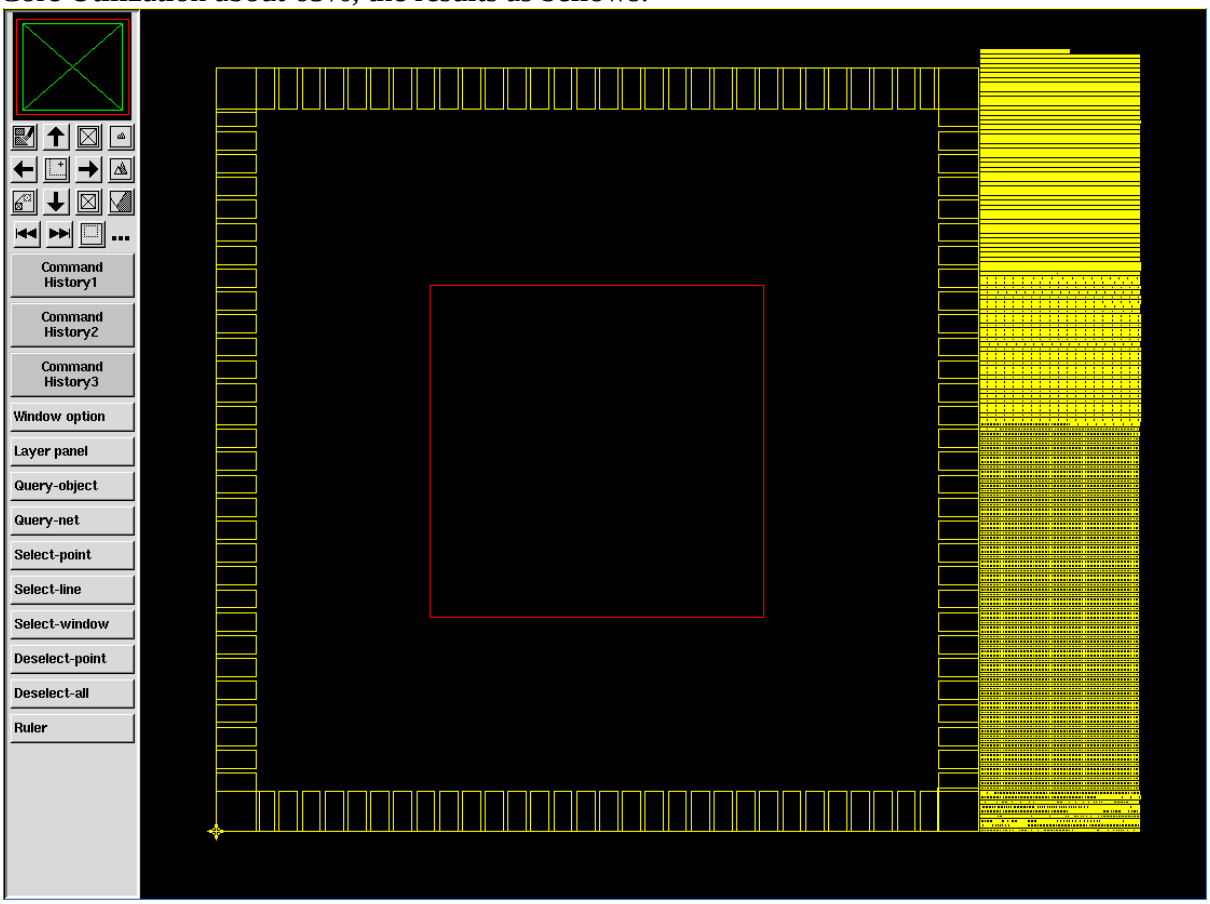
Load Design



Load SDC+TDF

❖ **<u>Step 3</u>: Floor Planning**

We use the floor-planning command of Astro and set the Width and Height with the Core Utilization about 65%, the results as bellows:



The Planner summary:

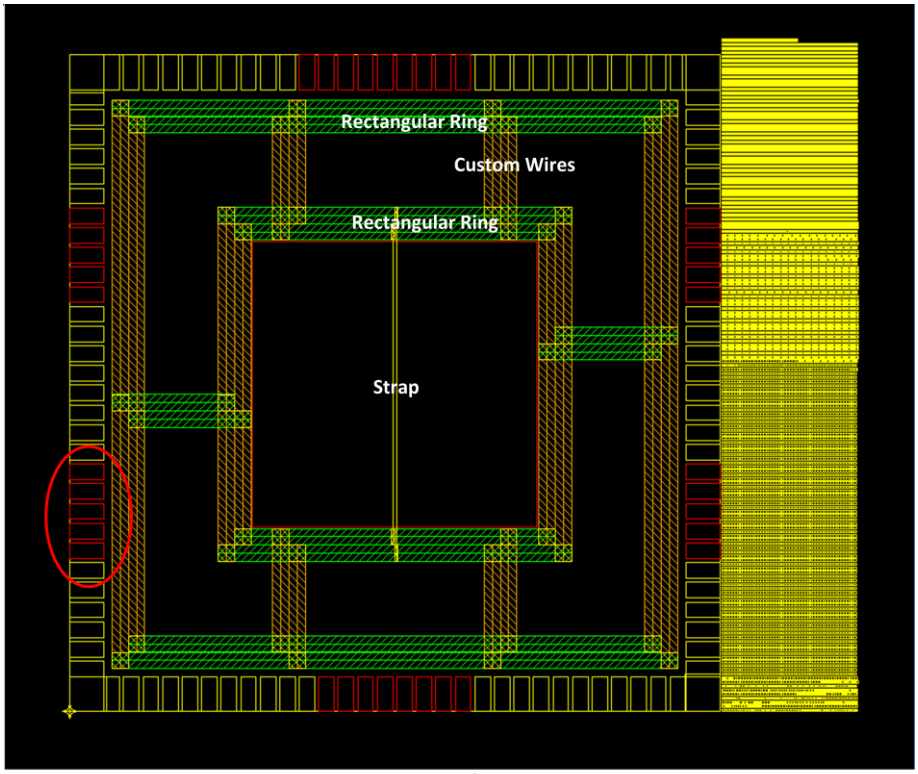
```
Planner Summary:
This floorplan is created by using tile name (unit).
        Row Direction = HORIZONTAL
        Control Parameter =  Width & Height
        Core Utilization = 0.653
        Number Of Rows = 118
        Core Width = 1749.84
        Core Height = 1750.76
        Aspect Ratio = 0.993
        Double Back ON
        Flip First Row = YES
        Start From First Row = YES
Planner run through successfully.
```

❖ **<u>Step 4</u>: Connect Ports to P/G, Create Rectangular Ring, Custom Wires**

In this step, we connect Ports to Power and Ground global nets, the results of Connect Ports to P/G as bellows:

```
On cell "ram_256x16x8_mbist_top_pad":
Connected 18417 ports to net (VDD) through pattern vdd!.
Connected 0 ports to the child nets.
#t
#t
#t
#t
On cell "ram_256x16x8_mbist_top_pad":
Connected 18417 ports to net (GND) through pattern gnd!.
Connected 0 ports to the child nets.
#t
```

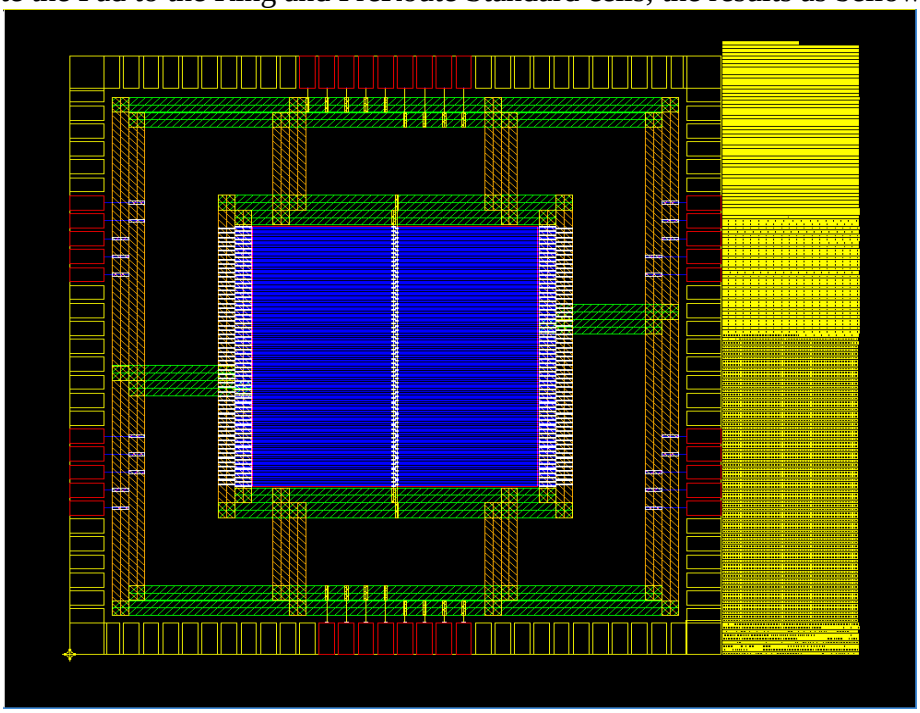
After finished the Port connections, we create the rectangular Ring, and the Strap as bellow results:



Create Rectangular Ring, Straps

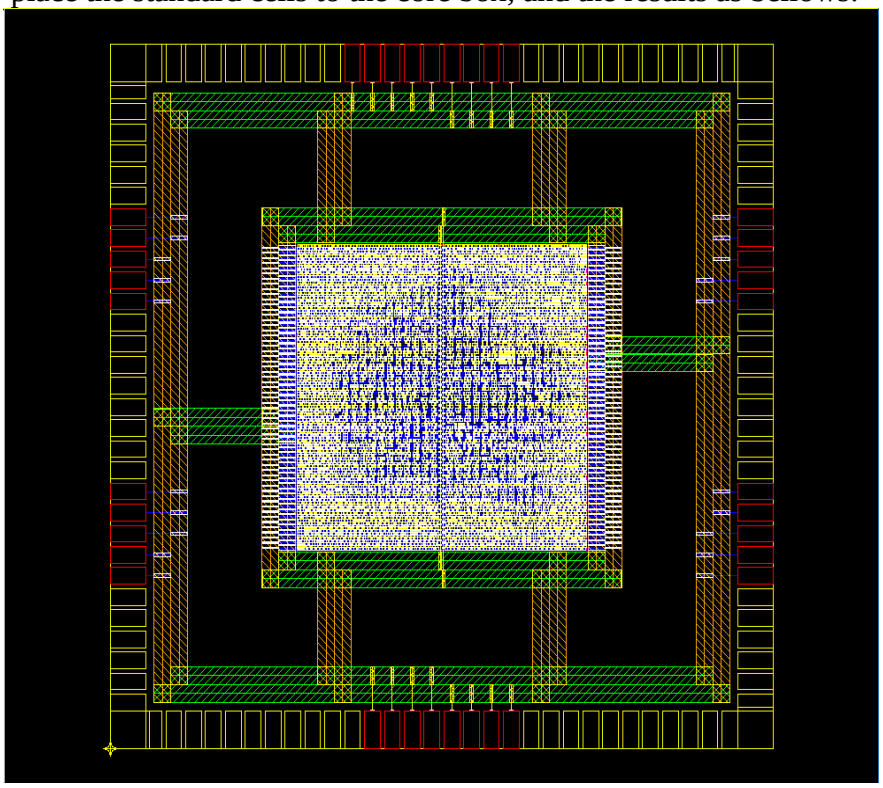
❖ **<u>Step 5</u>: PreRoute Marco/Pad and PreRoute Standard Cells**

PreRoute the Pad to the Ring and PreRoute Standard cells, the results as bellows:



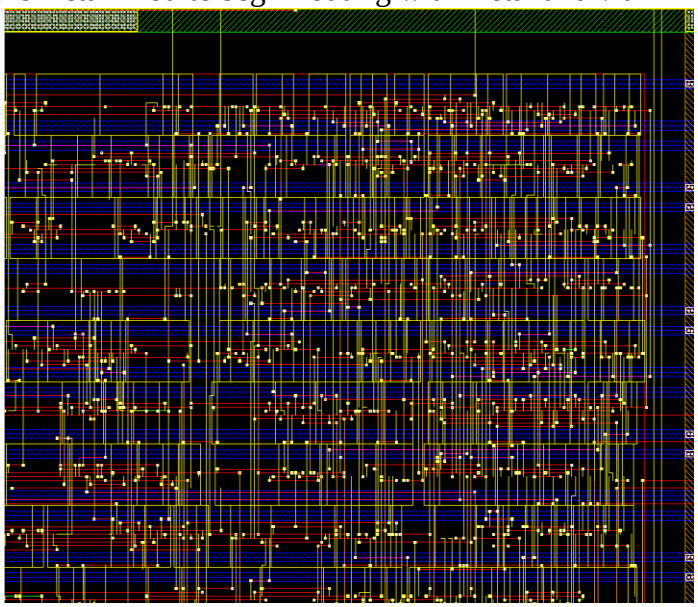
❖ **<u>Step 6</u>: Place the Standard Cells**

This step place the standard cells to the core box, and the results as bellows:



❖ **<u>Step 7</u>: Global Route/Detail Route**

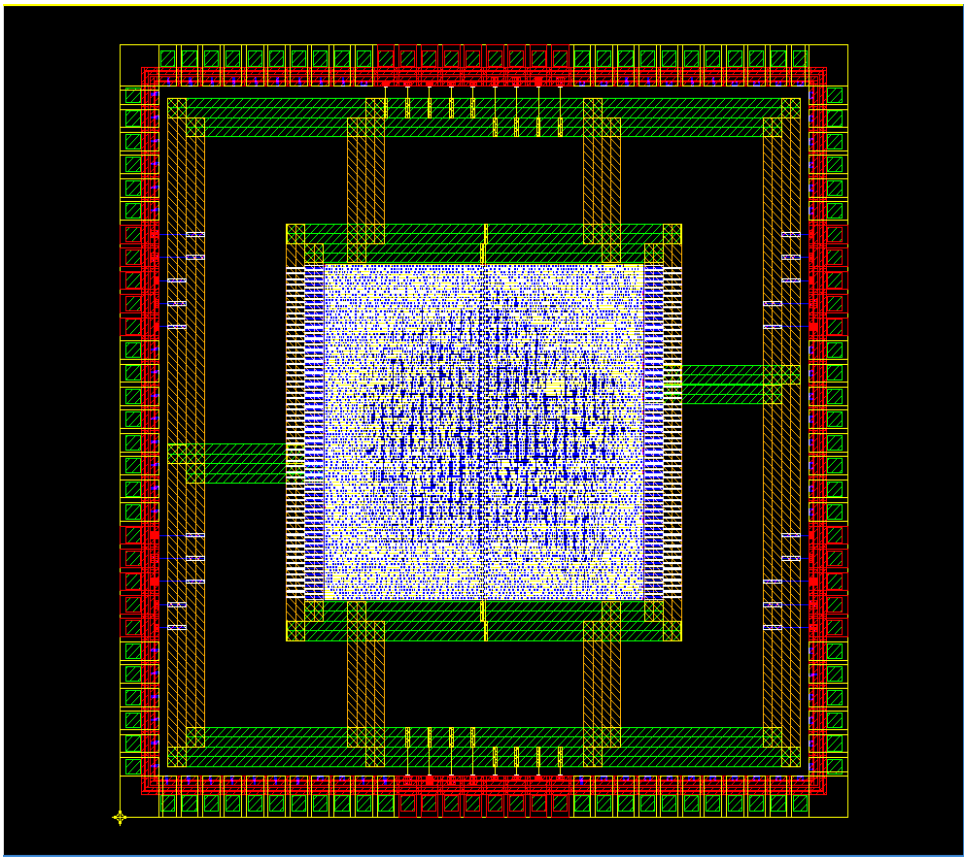
This step contains Global Route and Detail Route, the Global Route create the track for routing and the Detail Roulte begin routing with metal and via



The report DRC of Detail route is:

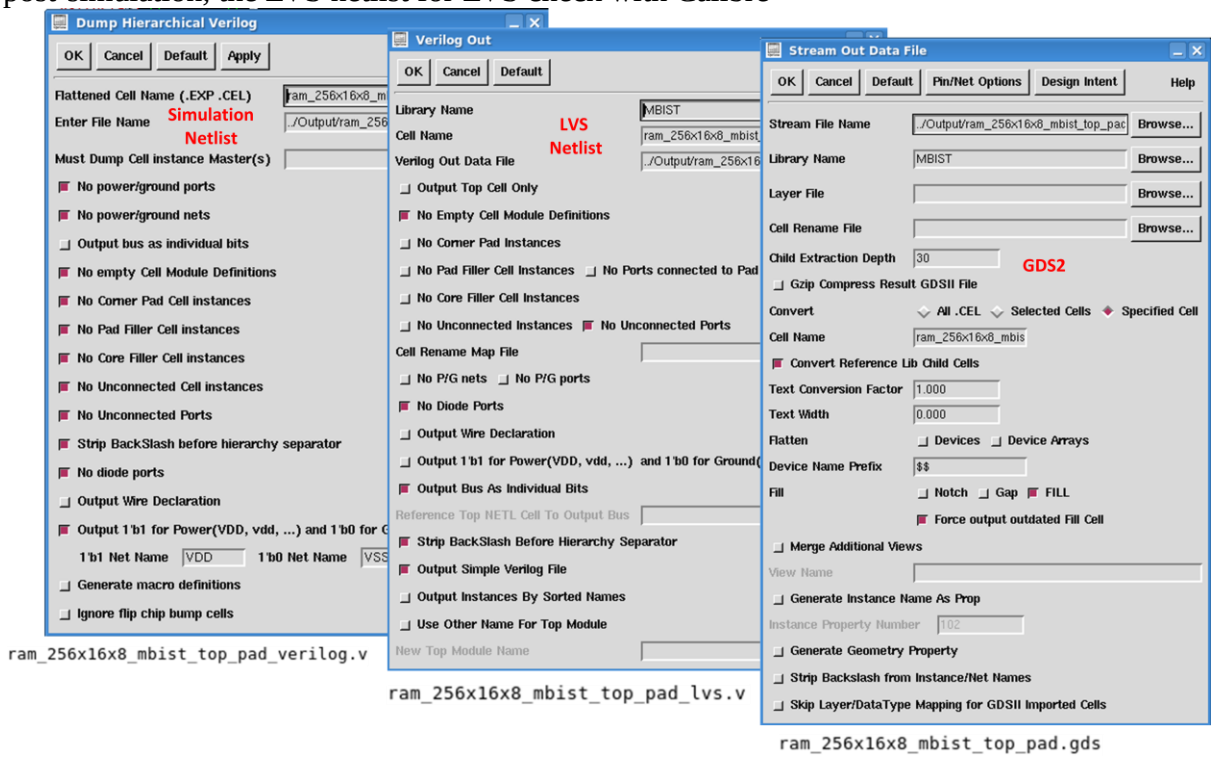
```
DRC-SUMMARY:

        @@@@@@@ TOTAL VIOLATIONS =       0 (0)
```

So we get the final layout design as bellows:



❖ **<u>Final step</u>: Export to GDS2, Simulation netlist, LVS netlist (for LVS check with Calibre)**

After finished the Routing, we export the data to GDS2, the simulation netlist for post-simulation, the LVS netlist for LVS check with Calibre



10. Datasheet

Features

ram_256x16x8_mbist_top is BIST for the **ram256x16x8** single-port **Random Access Memory (SRAM).** It supports the following features:

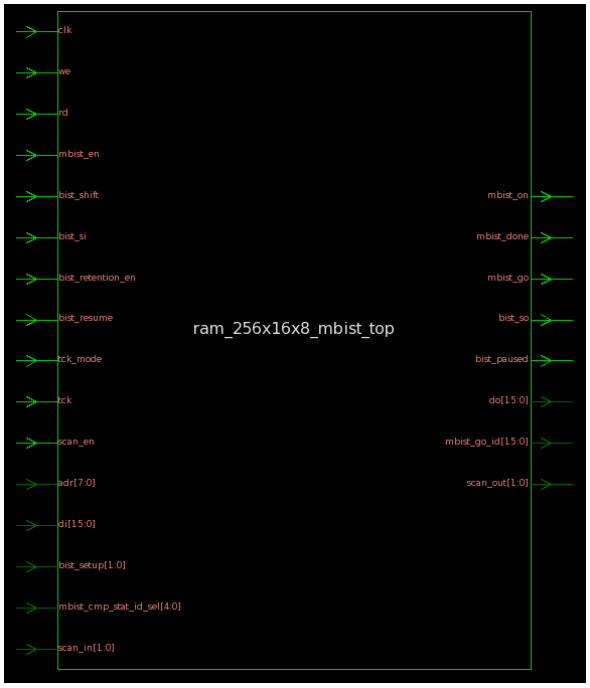
- **Non Stop-on-Fail mode**: to execute the test algorithms. The BIST will collect all the failure in SCAM
- **Stop-on-Fail Mode**: to execute the test algorithms. The BIST execution will stop immediately upon detecting failure in SRAM.
- **Retention Test Mode**: to pause BIST Algorithms at pre-defined test sequences to test the ability of memory cell to store data over time without any access.

Pin Description

Pad Name	Direction	Active	Description
clk	Input	N/A	System clock.
Memory interface pin			
we	Input	High	User write enable
rd	Input	High	User read enable
adr:0]	Input	N/A	User address
di[15:0]	Input	N/A	User data input
do[15:0]	Output	N/A	User data output
Scan interfaces			
scan_en	Input	High	Scan enable
scan_in[1:0]	Input	N/A	Scan input
scan_out[1:0]	Output	N/A	Scan output
BIST Interfaces			
mbist_en	Input	High	Enable BIST to execute
mbist_on	Output	High	Indicates that BIST are running
mbist_done	Output	High	Indicates that BIST has completed all test phases
mbist_go	Output	N/A	Indicates that BIST has completed all test phases without any failure when mbist_go is high (all comparators found no errors), when it is low, indicates that have the failure in memory
mbist_go_id[15:0]	Output	N/A	Its will high when corresponding comparator found no errors, otherwise indicates the position comparator errors
bist_shift	Input	High	Enables diagnostic data to be shifted in/out
bist_si	Input	N/A	Shift-in data into BIST internal scan chain.
bist_so	Output	N/A	Shift-out data.

bist_retention_en	Input	High	Enables Retention Test Mode. In this mode, BIST will pause at the defined test sequences for retention test.
bist_resume	Input	High	Resumes BIST execution when BIST is paused for retention test. This signal works only in the Retention Test Mode.
bist_setup[1:0]	Input	N/A	Configuration mode for BIST
tck_mode	Input	N/A	Select clock for diagnostic shift
tck	Input	N/A	Test clock
mbist_cmp_stat_id_sel[4:0]	Input	N/A	Select the output for comparartors

Interfaces



Appications

- Embedded Memory Built-in Selft Test

11. References

1. VLSI Test Principles and Architectures: Design for Testability, Laung-Terng Wang, Cheng-Wen Wu, Xiaoqing Wen. Morgan Kaufmann publications

2. CMOS VLSI Design, a Circuit and Systems Perspective, Neil H.E. Weste David Harris, Pearson Education, Inc.

3. RAM Fault Models & Test Algorithms, Cheng-wen Wu, National Tsing Hua University

4. Memory BIST, Ajay V. Tadiparti

5. Synopsis Simulation Manual