DIGITAL INDUSTRIES SOFTWARE

Automation of shared bus memory test with Tessent MemoryBIST

Executive summary

New requirements in automotive, artificial intelligence (AI), and processor applications have resulted in increased use of memory-heavy IP. Memory-heavy IPs for these applications are optimized for high performance, and they will often have a single access point for testing the memories. Tessent MemoryBIST provides an out-of-the-box solution for using this single access point, or shared bus interface. This paper describes the fundamental concepts and advantages of using a shared bus architecture for testing and repairing memories within IPs cores. It also outlines key features of the Tessent MemoryBIST automation that is available from Siemens Digital Industries Software.

Author: Harshitha Kodali

siemens.com/software

SIEMENS

Contents

Why use a shared bus architecture

These days, designs contain a huge number of memory arrays embedded in the core, and these memories often consume a substantial portion of the total chip area. This increase in memory size and number implies extra hardware cost for the associated memory built-in self-test (MBIST) logic. In addition to the area of the MBIST logic, there may be additional costs due to increased routing. It may even negatively impact the chip's performance in the critical functional paths to and from memories.

A shared bus architecture provides a common access point for several memories, allowing users to optimize routing and core performance. In addition, a shared bus architecture provides flexibility to users to route design-for-test (DFT) signals along functional paths behind the shared bus interface. Tessent MemoryBIST instruments automatically connect to the DFT signals to apply MBIST patterns through the shared bus interface.

About the shared bus architecture

A shared bus architecture contains an interface that provides a common access point to several memories. The architecture also provides the freedom of adding memories inside a module while preserving a fixed footprint at the memory cluster boundary.

A shared bus architecture contains a memory cluster module that provides access to multiple memories using a common shared bus interface, as shown in Figure 1. The shared bus interface provides access to data, control, and clock ports required by memories. A shared bus cluster module could have one or many shared bus interfaces; Figure 1 shows a design with one interface.

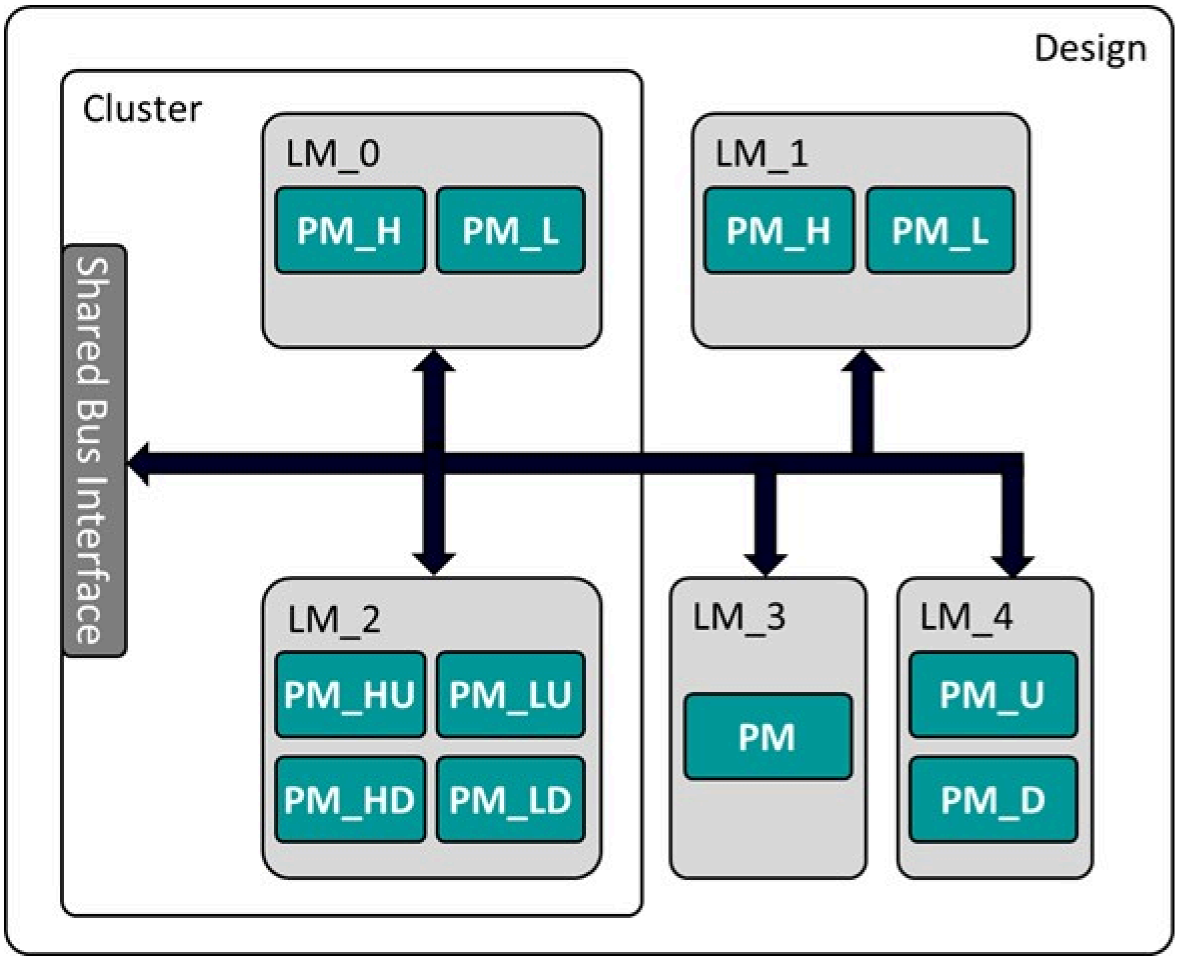


Figure 1. Shared bus cluster in a chip.

The memories accessed through the shared bus interface are called logical memories and are represented by **LM_0** through **LM_4** in Figure 1. These logical memories can be inside or outside a cluster module. A logical memory address space is composed of one or more physical memories, as shown in Figure 2.

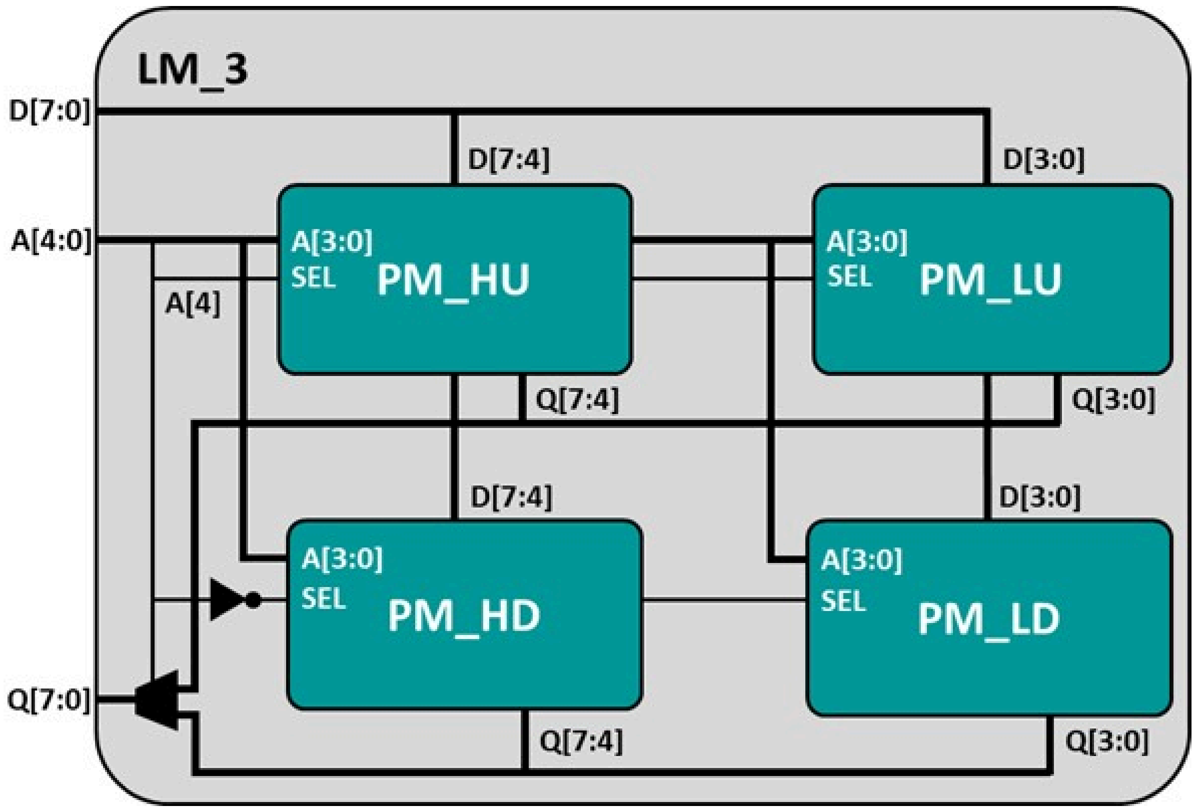


Figure 2. Composition of a logical memory address space.

A single logical memory can be accessed at any time by the shared bus interface. Each logical memory is enabled by specifying its corresponding selection code on the array select port of the interface. Once enabled, the corresponding logical memory can be accessed externally through the clock, data, and control ports of the interface.

A logical memory may be composed of one or more physical memories. The process of selecting and instantiating physical memories to compose each logical memory space is called physical RAM integration. Figure 2 shows the composition of logical memory **LM_3**. The logical memory receives and drives 8 bits of the data channel to/

from the shared bus interface. **LM_3** implements a horizontal and vertical stacking configuration, where two physical memories are active at a given time. The 4-bit data ports (Q[7:4] and Q[3:0]) of each horizontal pair of memories combine to form the lower or upper address range of the 8-bit data path of **LM_3**, depending on the address value, A[4].

The hierarchical nature of the logical and physical memories has several advantages:

- Early verification of the shared bus interface can be performed at the logical memory level using a behavioral model without physical memories
- The user can decide on the implementation of the physical memories without affecting the overall shared bus access, as the changes are localized within the logical level
- The user has the flexibility to select RAM and stacking configurations based on the design requirements

Memory BIST shared bus hardware

The embedded test hardware generated for the shared bus includes an MBIST controller and memory interfaces. In addition, extra modules, such as virtual memories and glue logic, are also generated and connected between the MBIST interfaces and the memory cluster, as shown in Figure 3.

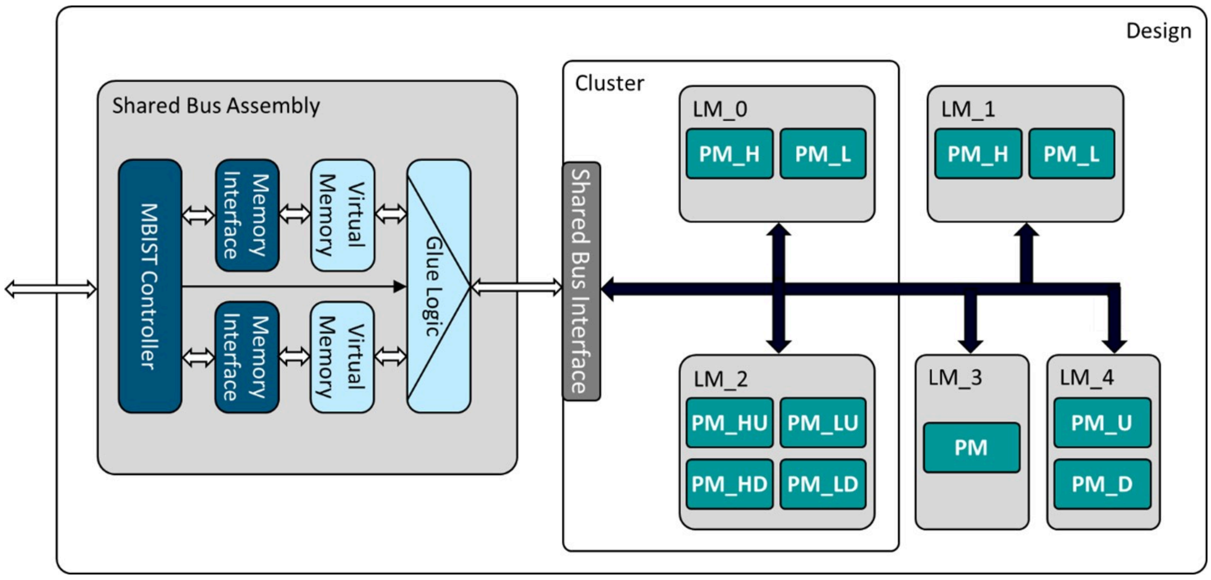


Figure 3. Memory BIST shared bus hardware.

One dedicated MBIST controller is assigned per shared bus memory cluster module. The memory interface provides the interface between the controller design object and the memory. The MBIST controller and interface logic are instantiated at the same level as the shared bus memory cluster.

The virtual memories correspond to the logical or physical memories inside the shared bus memory cluster module. One virtual memory is generated for each logical or physical memory. The glue logic handles the control and access logic between the shared bus interface ports, the MBIST controller, and the virtual memories. The virtual memories, and glue logic provides access to all the logical or physical memories. The virtual access enables the MBIST controller to run the memory algorithms and perform standard operations on all logical or physical memories.

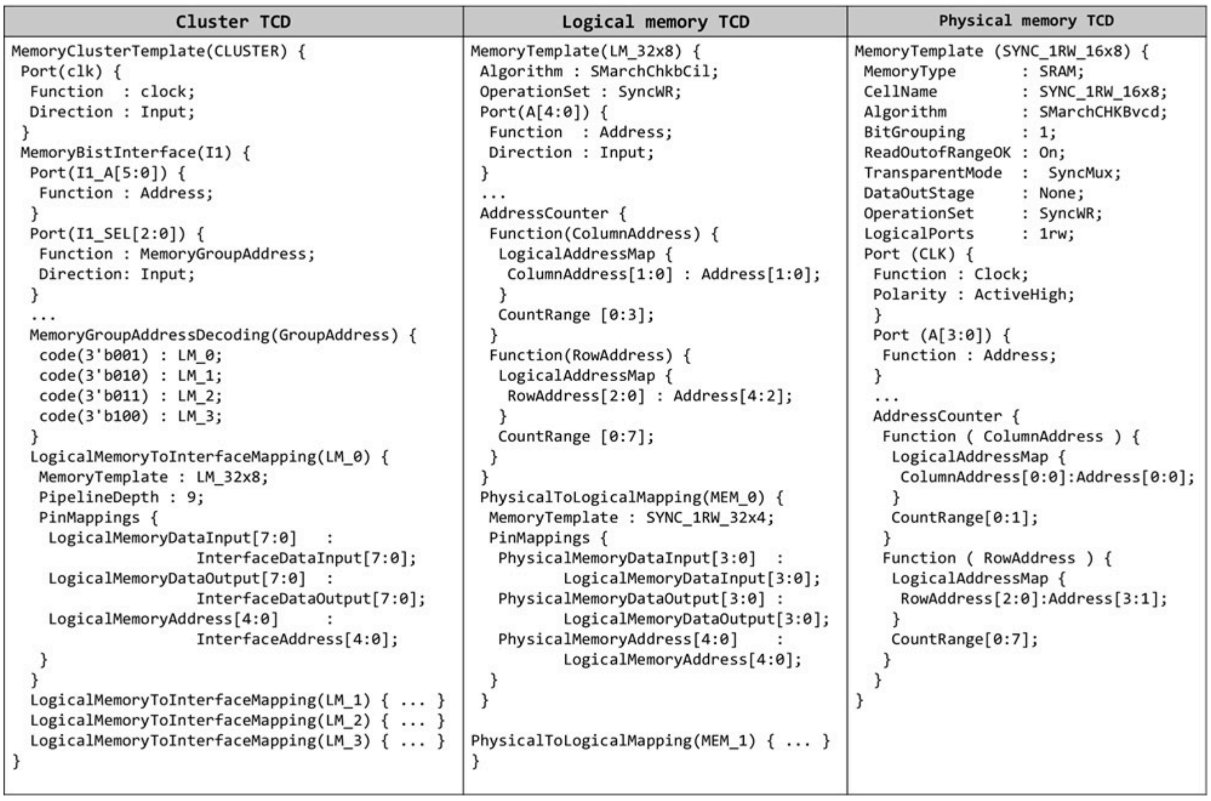
The MBIST controller, memory interface modules, virtual memories, and glue logic are all grouped inside a wrapped shared bus assembly module. This wrapping of the BIST logic enables cross-boundary area optimization during synthesis and reduces the ungrouped logic in the design after synthesis. This methodology enablesimproved logic optimization and significant area reduction.

Shared bus DFT flow automation

Tessent MemoryBIST provides an automated solution that addresses multiple DFT insertion flow requirements. The tool supports mapping and validating memory library files, memory repair, complex memory configurations, and multiple ways to optimize area.

Prerequisites to the DFT flow

MBIST DFT insertion can be done on an RTL or a gate-level netlist. In addition to the MBIST libraries, a Tessent cell library or ATPG library is required if the design has standard cells to be mapped. The memories attached to the shared bus interface must be described to Tessent MemoryBIST in the form of a Tessent Core Description (TCD). DFT insertion for a shared bus design requires memory cluster TCDs, logical memory TCDs, and physical memory TCDs, as shown in Figure 4.



Cluster TCD

```
MemoryClusterTemplate(CLUSTER) {
 Port(clk) {
  Function  : clock;
  Direction : Input;
 }
 MemoryBistInterface(I1) {
  Port(I1_A[5:0]) {
   Function : Address;
  }
  Port(I1_SEL[2:0]) {
   Function : MemoryGroupAddress;
   Direction: Input;
  }
  ...
  MemoryGroupAddressDecoding(GroupAddress) {
   code(3'b001) : LM_0;
   code(3'b010) : LM_1;
   code(3'b011) : LM_2;
   code(3'b100) : LM_3;
  }
  LogicalMemoryToInterfaceMapping(LM_0) {
   MemoryTemplate : LM_32x8;
   PipelineDepth : 9;
   PinMappings {
    LogicalMemoryDataInput[7:0]   :
                    InterfaceDataInput[7:0];
    LogicalMemoryDataOutput[7:0]  :
                    InterfaceDataOutput[7:0];
    LogicalMemoryAddress[4:0]     :
                    InterfaceAddress[4:0];
   }
  }
  LogicalMemoryToInterfaceMapping(LM_1) { ... }
  LogicalMemoryToInterfaceMapping(LM_2) { ... }
  LogicalMemoryToInterfaceMapping(LM_3) { ... }
}
```

Logical memory TCD

```
MemoryTemplate(LM_32x8) {
 Algorithm : SMarchChkbCil;
 OperationSet : SyncWR;
 Port(A[4:0]) {
  Function  : Address;
  Direction : Input;
 }
 ...
 AddressCounter {
  Function(ColumnAddress) {
   LogicalAddressMap {
    ColumnAddress[1:0] : Address[1:0];
   }
   CountRange [0:3];
  }
  Function(RowAddress) {
   LogicalAddressMap {
    RowAddress[2:0] : Address[4:2];
   }
   CountRange [0:7];
  }
 }
 PhysicalToLogicalMapping(MEM_0) {
  MemoryTemplate : SYNC_1RW_32x4;
  PinMappings {
   PhysicalMemoryDataInput[3:0]  :
          LogicalMemoryDataInput[3:0];
   PhysicalMemoryDataOutput[3:0] :
          LogicalMemoryDataOutput[3:0];
   PhysicalMemoryAddress[4:0]    :
          LogicalMemoryAddress[4:0];
  }
 }

PhysicalToLogicalMapping(MEM_1) { ... }
}
```

Physical memory TCD

```
MemoryTemplate (SYNC_1RW_16x8) {
 MemoryType        : SRAM;
 CellName          : SYNC_1RW_16x8;
 Algorithm         : SMarchCHKBvcd;
 BitGrouping       : 1;
 ReadOutofRangeOK : On;
 TransparentMode   :  SyncMux;
 DataOutStage      : None;
 OperationSet      : SyncWR;
 LogicalPorts      : 1rw;
 Port (CLK) {
  Function : Clock;
  Polarity : ActiveHigh;
  }
  Port (A[3:0]) {
   Function : Address;
  }
  ...
  AddressCounter {
   Function ( ColumnAddress ) {
    LogicalAddressMap {
     ColumnAddress[0:0]:Address[0:0];
    }
    CountRange[0:1];
   }
   Function ( RowAddress ) {
    LogicalAddressMap {
     RowAddress[2:0]:Address[3:1];
    }
    CountRange[0:7];
   }
  }
}
```

Figure 4. Shared bus cluster, logical memory, and physical memory TCD example.

The cluster TCD is associated with each shared bus memory cluster module. It describes the shared bus interface ports, access codes to enable logical memories, and port mappings between the logical memories and the shared bus interface.

The logical memory TCD is associated with each logical memory in the cluster module. It describes the logical memory ports, access codes to enable physical memories (if any), and the port mappings between logical and physical memories. The end-user modifies the logical TCD to incorporate the physical memory to logical memory mapping after RAM integration.

The physical memory TCD is associated with each physical memory and is usually generated by a memory compiler. The physical memory TCD describes the characteristics of the memory important for the test.

Automation to map and validate memory library files

Tessent MemoryBIST provides an automated approach called shared bus learning to map the physical memory composition of each logical memory and validate the cluster and logical memory library files. It is recommended to run shared bus learning on a single, stand-alone cluster before DFT insertion. An overview of a shared bus learning flow is shown in Figure 5. Note that shared bus learning currently supports automation for 1RW and 1R1W memories.

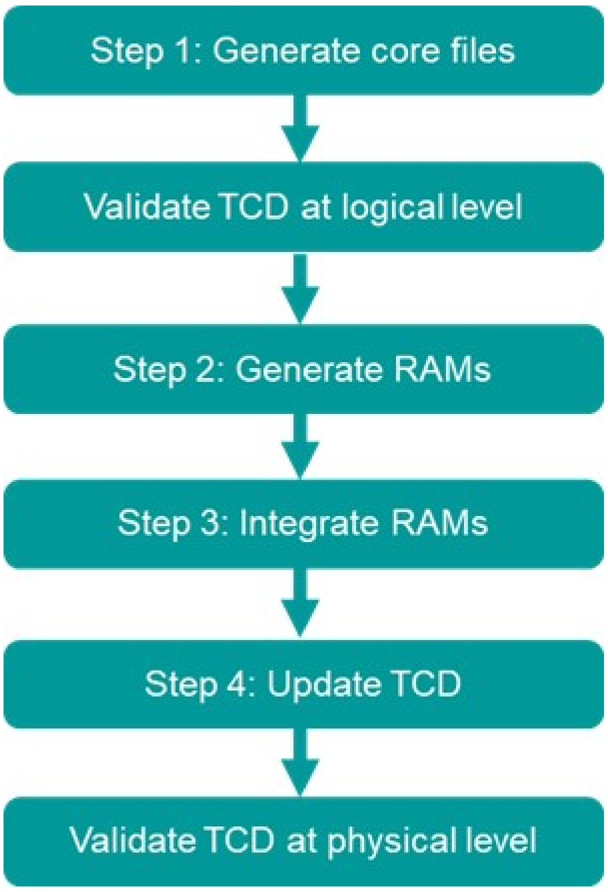

Step 1: Generate core files
Validate TCD at logical level
Step 2: Generate RAMs
Step 3: Integrate RAMs
Step 4: Update TCD
Validate TCD at physical level

Figure 5: Overview of shared bus learning flow.

Physical-to-Logical (P2L) mapping automation

The logical memory address space can be implemented by one or more physical memories during RAM integration. After adding physical memories into the cluster RTL, the logical memory TCD must indicate how the physical memories will be activated during MBIST. The logical memory TCD must reflect the connectivity of the physical memories to the logical memory ports. Step 4, highlighted in Figure 6, shows where the P2L mapping automation fits into the overall DFT preparation flow.

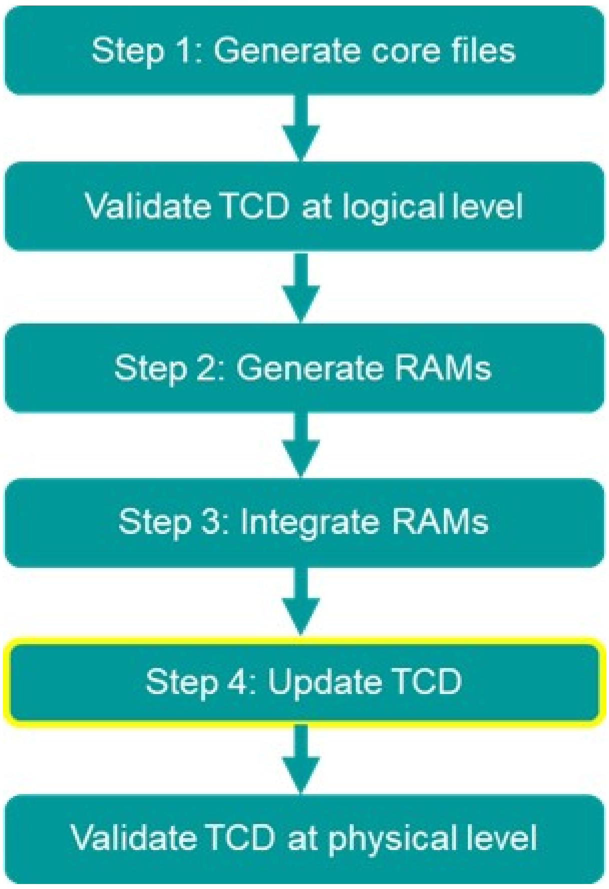

Step 1: Generate core files
Validate TCD at logical level
Step 2: Generate RAMs
Step 3: Integrate RAMs
Step 4: Update TCD
Validate TCD at physical level

Figure 6. Shared bus learning P2L mapping automation in DFT preparation flow.

The Tessent Shell command **set_memory_cluster_library_generation_options** automates the creation of the P2L wrappers. First, it derives the mappings of the physical memories within a logical memory from the memory cluster RTL. The command then populates the extracted information into the logical memory TCD and writes the updated files.

Library validation

The shared bus learning feature also includes a library validation capability. This library validation verifies the memory cluster TCD and the logical memory TCD through a design tracing-based approach. By using the **set_memory_cluster_validation_options** command, it validates that:

- No memory is omitted from MBIST testing
- Port mappings specified in the cluster and logical memory TCD are consistent with the design
- Pipeline stages that surround the logical memory are consistent with the cluster TCD

The typical usage for **set_memory_cluster_validation_options** is to validate the final memory cluster TCD and logical memory TCD after the RAM integration process. The command also provides an option to perform validation at the logical level. In this usage, validation of the memory cluster TCD is performed before implementing the physical memories into the design. The yellow highlights in Figure 7 show two places in the DFT preparation flow where validation can be done. The validation is a recommended step before proceeding with DFT insertion.

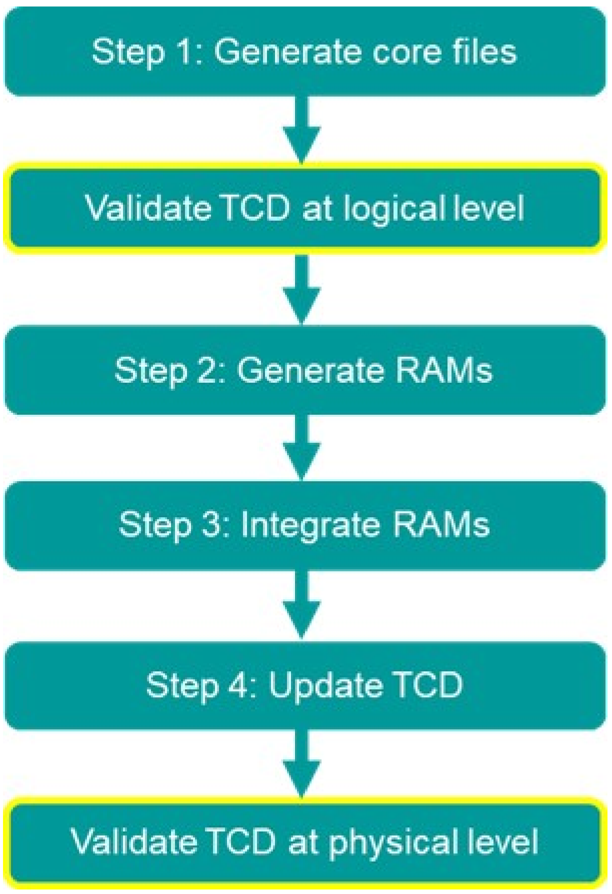

Step 1: Generate core files
Validate TCD at logical level
Step 2: Generate RAMs
Step 3: Integrate RAMs
Step 4: Update TCD
Validate TCD at physical level

Figure 7: Shared bus learning library validation in DFT preparation flow.

Shared bus DFT insertion flow

Tessent Shell is an integrated platform built on an IJTAG (IEEE 1687) infrastructure that enables users to perform all the tasks required for DFT. Figure 8 shows the steps in the DFT insertion flow.

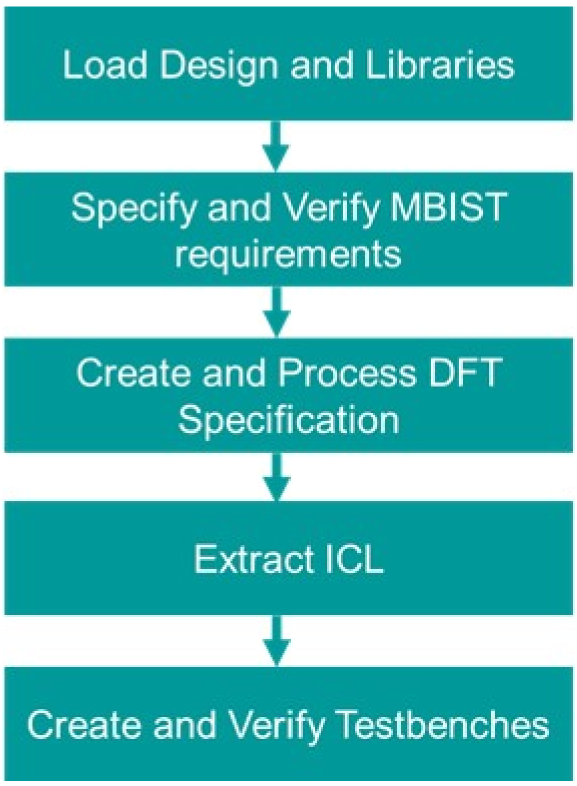

Load Design and Libraries
Specify and Verify MBIST requirements
Create and Process DFT Specification
Extract ICL
Create and Verify Testbenches

Figure 8: Shared bus DFT insertion flow.

The first step is to load the RTL/gate-level design and memory TCD libraries required for MBIST. Recall, a Tessent cell library or ATPG library is required if the design has standard cells to be mapped.

The second step is to specify and verify MBIST requirements. Here, the user can define clocks, load DEF, and load UPF/CPF files, if available. Users can also make global changes that apply to all the MBIST controllers and memories. Tessent MemoryBIST can test the shared bus cluster at the logical or physical memory access level. When testing at the physical memory access level, the memory interface and virtual memory (shown in Figure 3) is created for each physical memory. The memory access level option can be set during this time. After specifying the requirements, the tool then analyzes the design and performs design rule checks (DRC) specific to the current context.

The third step is to create and process the **DftSpecification** information. First, TMB creates a **DftSpecification** which describes how the memories are tested. Tessent MemoryBIST considers various standard parameters and options set by the user when creating this specification. After creation, the **DftSpecification** can be modified as desired. Finally, validate the modified specification and generate and insert the DFT hardware into the design by processing the final **DftSpecification**.

The fourth step is the extraction of instrument connectivity language (ICL) that describes the connectivity of the instruments present on the IJTAG network. Tessent MemoryBIST also extracts the timing constraints for synthesis and static timing analysis (STA).

The fifth step is to create and verify the testbenches. By default, Tessent MemoryBIST generates signoff or simulation Verilog testbench patterns. However, it can also generate manufacturing test patterns in various formats that can be applied on the automatic test equipment (ATE).

Repairable memories in a memory cluster module

Tessent MemoryBIST also supports repairable memories with Row/Word-only, Column/IO-only, and Row/Column repair types in a shared bus cluster. Tessent MemoryBIST inserts the required built-in repair analysis (BIRA) and built-in self-repair (BISR) logic for such repairable memories. The BIRA circuitry determines if a memory is repairable, and then calculates the repair information based on its redundancy scheme. The BIRA logic for a shared bus memory cluster is placed within the MBIST controller. The BISR register holds the repair information. BISR registers associated with repairable memories are connected to form scan chains. BISR chains are automatically inserted if memories instantiated in the design have spare resources described in their memory TCD. The BISR logic is placed inside the shared bus memory cluster near the memory to be repaired, as shown in Figure 9.

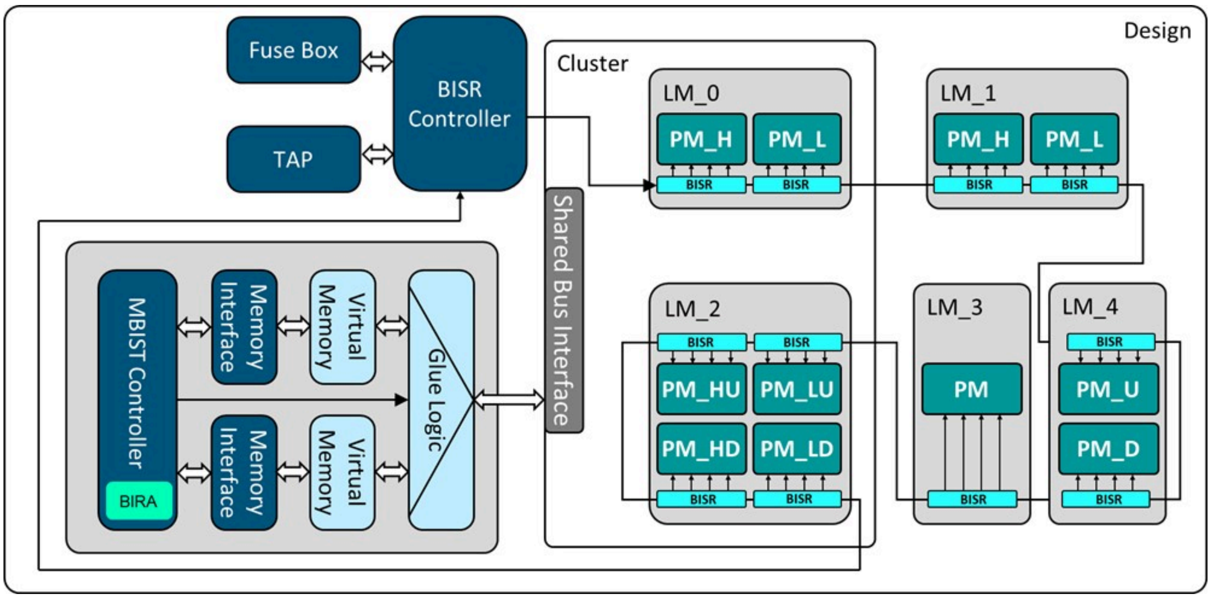


Figure 9. Design overview of a memory cluster module with BISR.

The BISR controller hardware is automatically created at the top level of the chip. The BISR controller implements functions that:

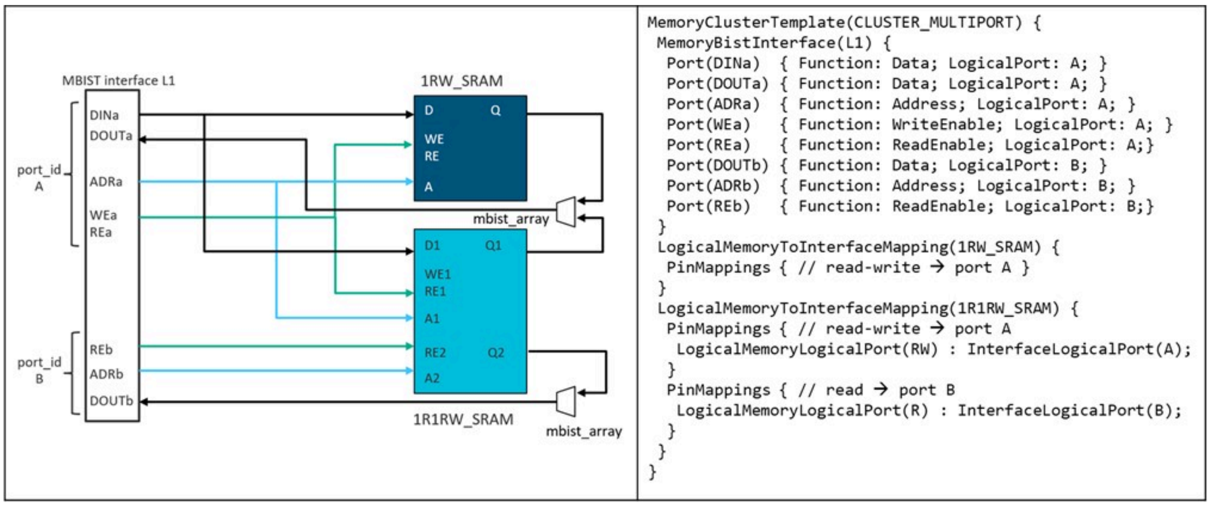
- Compress the repair information and write the result into the fuse box
- Decompress the fuse box contents and shift the contents into the chip BISR chain
- Initialize or observe BISR chain content
- Read and program fuses

The BISR controller is accessed using the TAP, and the BISR chain control ports are automatically connected to the BISR controller. The BISR controller is also connected to the fuse box. In addition, the BISR controller provides external access to the BISR chain and the fuse box via the TAP. Therefore, all the BISR controller functions mentioned above can be implemented via the TAP.

The BISR controller can asynchronously reset the BISR chain when the controller's functional repair enable pin is held active. The BISR controller can also decompress the fuse box content and shift the contents into the chip BISR chain when a low-to-high transition is applied on its functional repair enable input pin.

Complex memory configurations support: multi-port and pseudo-vertical stacking

Tessent MemoryBIST supports testing of multi-port memories, i.e., xRyWzRW within the shared bus memory cluster. Tessent allows performing operations on inactive ports of multi-port memories in parallel with the algorithm's read/write operations on the active port to detect inter-port faults. It provides flexibility to users to control concurrent operations through an operation set. The MBIST DFT insertion flow for multi-port memories is the same as single port memories. Figure 10 shows an example cluster TCD with 1R1RW logical memory with the required properties for each logical port mapping.



```
MemoryClusterTemplate(CLUSTER_MULTIPORT) {
 MemoryBistInterface(L1) {
  Port(DINa)  { Function: Data; LogicalPort: A; }
  Port(DOUTa) { Function: Data; LogicalPort: A; }
  Port(ADRa)  { Function: Address; LogicalPort: A; }
  Port(WEa)   { Function: WriteEnable; LogicalPort: A; }
  Port(REa)   { Function: ReadEnable; LogicalPort: A;}
  Port(DOUTb) { Function: Data; LogicalPort: B; }
  Port(ADRb)  { Function: Address; LogicalPort: B; }
  Port(REb)   { Function: ReadEnable; LogicalPort: B;}
 }
 LogicalMemoryToInterfaceMapping(1RW_SRAM) {
  PinMappings { // read-write → port A }
 }
 LogicalMemoryToInterfaceMapping(1R1RW_SRAM) {
  PinMappings { // read-write → port A
   LogicalMemoryLogicalPort(RW) : InterfaceLogicalPort(A);
  }
  PinMappings { // read → port B
   LogicalMemoryLogicalPort(R) : InterfaceLogicalPort(B);
  }
 }
}
```

Figure 10: Shared bus memory cluster and TCD with multi-port logical memory.

Tessent MemoryBIST also supports pseudo-vertical stacking of physical memory in the shared bus cluster. The pseudo-vertical stacking configuration is always implemented using a single physical memory. For example, Figure 11 shows a logical memory with a logical address space of 64x8.

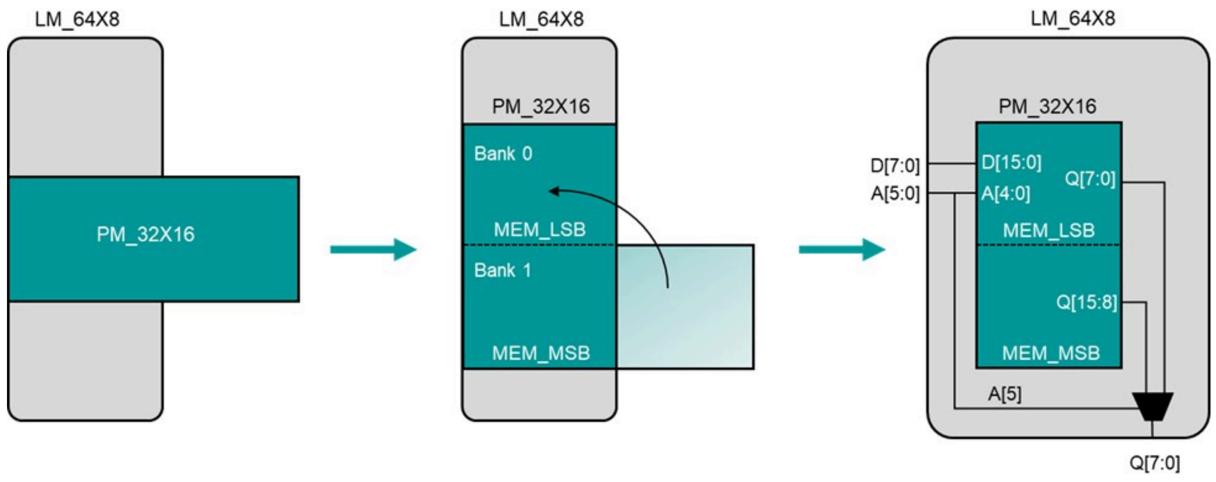
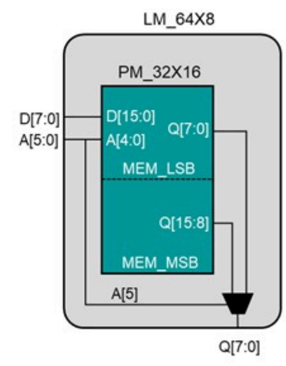


Figure 11. Logical memory with pseudo-stacked physical memory in a shared bus.

The logical memory shown in Figure 11 is implemented with a physical memory of dimension 32x16. The physical memory has half the number of addresses but is double the data width of the logical address space. The physical memory is split virtually, so each 8 bits of the physical memory's data bus will represent one-half of the logical memory. One can think of the physical memory being split into a "left half" corresponding to bits [15:8] and a "right half" corresponding to bits [7:0]. The logical memory is then composed of each virtual "half" of the physical memory stacked vertically. Hence the term pseudo vertical stacking.

The logical memory's address A[5] splits the physical memory data path into two parts by selecting the upper or lower portion, or bank, of the physical memory data path bits. The remaining logical memory address A[4:0] bits control the address inputs of the physical memory.

From the perspective of the logical memory TCD, pseudo-vertical stacking of physical memory is implemented by specifying multiple **PhysicalToLogicalMapping** wrappers, where each references the same physical memory instance. Figure 12 shows an example of the **PhysicalToLogicalMapping** wrappers in the logical memory TCD.



```
PhysicalToLogicalMapping(MEM_LSB) {
    MemoryTemplate : SYNC_1RW_32x16_SR;
    MemoryInstanceName : SYNC_1RW_32x16_inst;
    PinMappings {
      PhysicalMemoryDataInput[7:0]   : LogicalMemoryDataInput[7:0];
      PhysicalMemoryDataOutput[7:0]  : LogicalMemoryDataOutput[7:0];
      PhysicalMemoryAddress[4:0]     : LogicalMemoryAddress[4:0];
    }
  }
  PhysicalToLogicalMapping(MEM_MSB) {
    MemoryTemplate : SYNC_1RW_32x16_SR;
    MemoryInstanceName : SYNC_1RW_32x16_inst;
    PinMappings {
      PhysicalMemoryDataInput[15:8]  : LogicalMemoryDataInput[7:0];
      PhysicalMemoryDataOutput[15:8] : LogicalMemoryDataOutput[7:0];
      PhysicalMemoryAddress[4:0]     : LogicalMemoryAddress[4:0];
    }
  }
```

Figure 12. Physical-to-logical mapping in logical memory TCD.

Tessent MemoryBIST also supports physical memories with unused bits in the MSBs of each memory partition defined in the corresponding **PhysicalToLogicalMapping** wrapper. If the unused bits are not in the MSB of each memory partition, then the physical memory access level is not supported, and memories have to be tested at the logical memory access level.

Area optimization in shared bus DFT

One of the area optimization capabilities of Tessent MemoryBIST in shared bus designs is interface reuse. If identical memories in the design are not tested concurrently, then the memory interface and virtual memory are reused among the identical memories. As a result, it reduces the MBIST DFT area significantly. Figure 13 shows an example where **PM_HU** and **PM_HD** of **LM_2** (serially tested identical memories) share the memory interface and virtual memory.

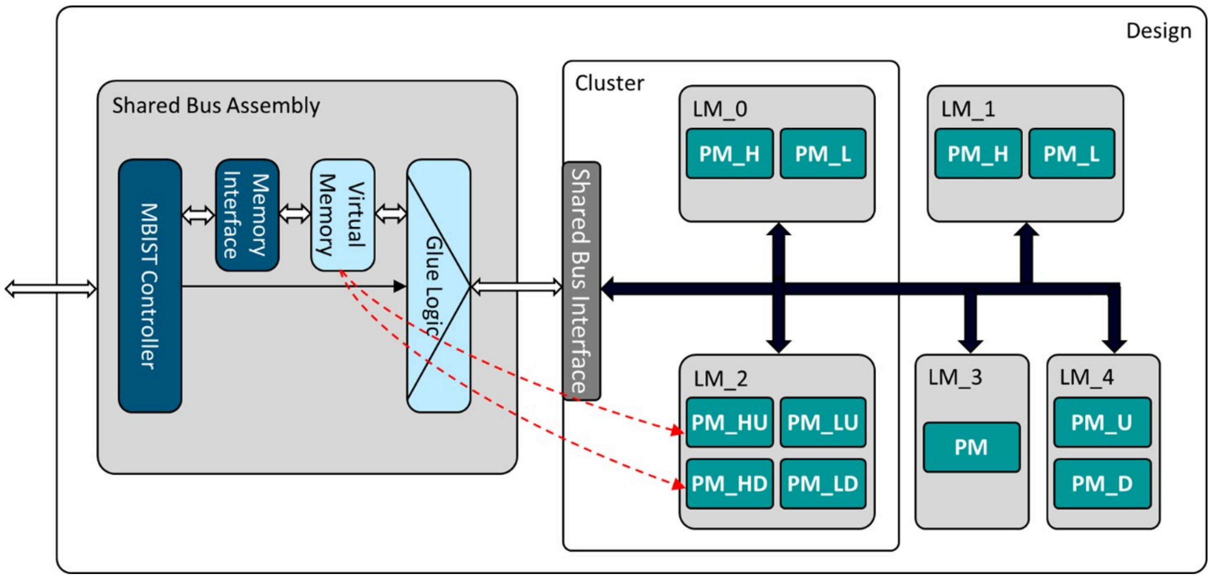


Figure 13: Area optimization from sharing memory interface and virtual memory.

Repair sharing can be implemented on memories with Row/Word-only, Column/IO-only, and Row/Column repair types. Memories of different dimensions can share the same BISR/BIRA hardware. Repair sharing applies the same repair information to all memories within a repair group. The designer also has control over the level of sharing or grouping of memories on a BISR/BIRA circuit to maintain a proper balance between potential yield impact and improvements in area and power-up time.

Figure 14 shows **LM_2** and **LM_4** implementing repair sharing, where the repair solution is shared among all of the physical memories within the logical memory.

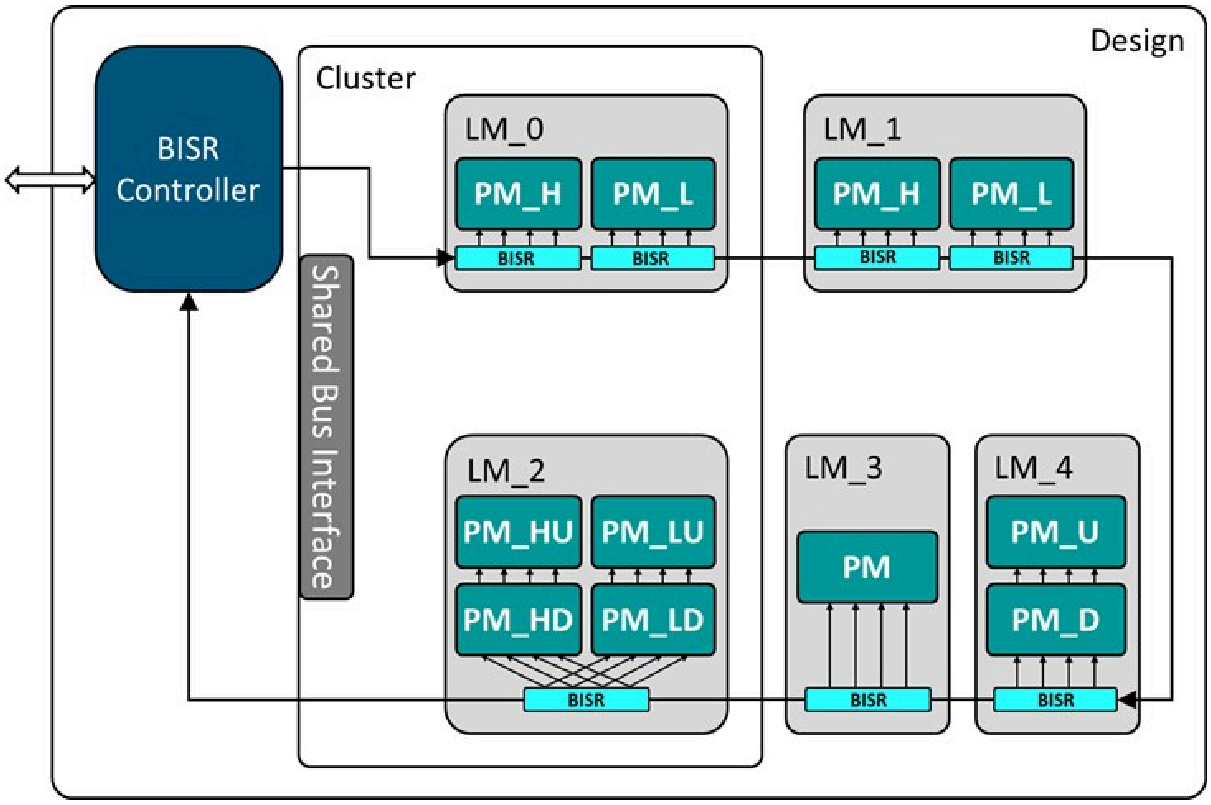


Figure 14: Repair sharing.

Summary: Tessent MemoryBIST automates shared bus memory test

Implementation of the most challenging DFT tasks is greatly simplified by the production-proven and widely-adopted automation available in Tessent products. Further, as the memory content of today’s chips continues to grow, Tessent MemoryBIST significantly helps manage this complexity.

The ever-increasing demand for testing complex memory configurations is efficiently addressed by Tessent MemoryBIST by automating support for:

- Memory library mapping and validation
- DFT insertion flow
- Complex memory configurations support
- DFT architecture for optimized area

For more information on Tessent MemoryBIST and shared bus support, refer to the Tessent MemoryBIST User's Manual or contact your support representative.

Siemens Digital Industries Software

Offices

Headquarters
Granite Park One 5800 Granite Parkway
Suite 600
Plano, TX 75024 USA
+1 972 987 3000

Americas
Granite Park One 5800 Granite Parkway
Suite 600
Plano, TX 75024 USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

For additional numbers, click here.

About us

Siemens Digital Industries Software helps organizations of all sizes digitally transform using software, hardware and services from the Siemens Xcelerator business platform. Siemens' software and the comprehensive digital twin enable companies to optimize their design, engineering and manufacturing processes to turn today's ideas into the sustainable products of the future. From chips to entire systems, from product to process, across all industries. Siemens Digital Industries Software – Accelerating transformation.

About the author

Harshitha Kodali

Harshitha Kodali is a product engineer for the Tessent group at Siemens Digital Industries Software.

10/2023 84519