SYNOPSYS®

Logic BIST implementation using LPDDR5/4/4X Application Note

LBIST using LPDDR5/4/4X PHY

Version 1.00a
July 2020

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

Revision	Date	Description
1.00a	July 2020	Initial release

Reference Documents

The following reference documents are used in this application note:

- *DFTMAX Design-For-Test user guide*
- *Designware Cores LPDDR5/4/4X PHY Implementation Guide*
- *LPDDR5/4/4X OCC Application Note*
- *SolvNetPlus article 25733 "Finding optimal seed values for the LogicBIST PRPG"*
- *SolvNetPlus article 31040 "Setting the seed and signature values in a LogicBIST Design"*

Note	This application note focuses on LBIST implementation and testing using Synopsys tools • DFT Compiler- 2018.06-SP2 • TestMAX- 2019.12 • VCS- 2019.06-SP2

1 Logic BIST overview

1.1 Purpose

The purpose of this application note is to provide guidance for implementing Logic BIST (LBIST) using the LPDDR5/4/4X PHY IP. This application note applies to all the process nodes for which the LPDDR5/4/4X PHY IP is available.

In this document users will find:

- Introduction to LBIST
- LPDDR5/4/4X LBIST specifics
- Guidance on LBIST insertion and validation

Detailed functionality of LBIST inserted by Synopsys DFT tools can be found in

DFTMAX Design-For-Test User Guide

1.2 Overview

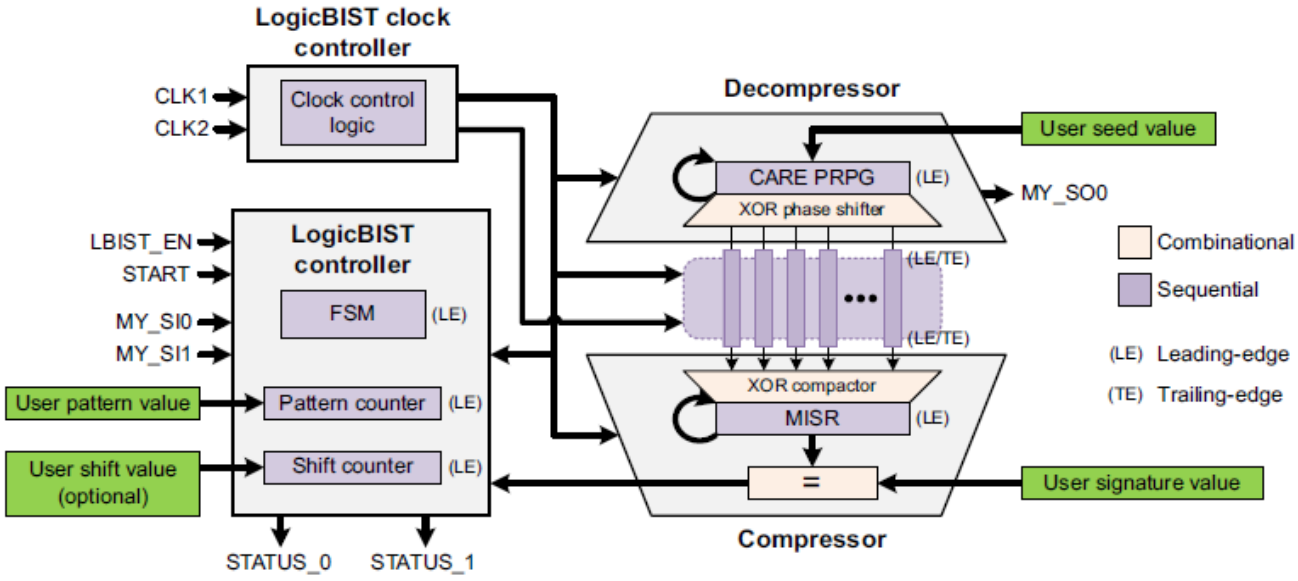
Logic BIST addresses functional safety requirements for digital integrated circuits that are used in the automotive semiconductor industry.

Self-test utilizes scan chains and a small amount of logic added by a DFT tool such as Synopsys DFT Compiler. LBIST uses pseudo-random pattern generator (PRPG) to create scan data, and a multiple-input signature register (MISR) to compare the design response to an expected value. At the end of the test, if the actual signature matches the expected signature, a PASS signal will be asserted.

Since scan patterns are generated on-chip with PRPG and test results are compressed into MISR to generate final signature for comparison, test time can be reduced since scan shift is not limited by external data. However, using pseudorandom patterns might limit detection of some random resistant logic or specific pattern types.

Figure 1.1 illustrates LBIST architecture.

Figure 1.1: LBIST architecture



The Logic BIST architecture consist of four components:

- LogicBIST controller
- Decompressor
- Compressor
- Clock controller

1.2.1 LogicBIST controller

The LogicBIST controller contains the following:

- Small finite state machine (FSM) that controls BIST operation
- Pattern counter that applies the user-specified number of test patterns
- Shift counter that applies the correct number of shift clock cycles for each pattern

When the design is in mission mode and LBIST is disabled, the LogicBIST controller is idle and its clock is disabled.

When the design is in a non-LBIST scan mode, the pattern counter and shift counter operate as scannable design logic so that their logic can be scan-tested. The FSM flip-flops are excluded from scan testing so any OCC, ICG, reset, or scan-enable control logic does not interfere with scan testing.

When the design is in mission mode and LBIST is enabled:

- The LogicBIST controller controls the scan-enable and wrapper-shift signals in the design.
- At the beginning of the test program, the FSM initializes the decompressor PRPG and compressor MISR to their initial states, and it loads the pattern and shift counters to their user-specified initial values.
- During the test program, the LogicBIST controller runs the pattern and shift counters through their sequences. As the shift counter counts through its sequence, the scan chains perform load/unload using the PRPG/MISR, respectively. When the shift counter reaches zero, the FSM issues the capture cycle(s), decrements the pattern counter, and begins a new shift counter sequence.
- When the pattern counter reaches zero, the current MISR signature value is compared with the user-specified expected signature value. If they match, the test passes; if not, the test fails.

1.2.2 Decompressor

The LogicBIST decompressor feeds data into the compressed scan chains in the core logic. It is responsible for generating target fault care bits.

A PRPG, or pseudo-random pattern generator, is comprised of the following two components:

- A linear feedback shift register (LFSR) that generates the next data bit of the next data word as a linear XOR function of its current data word
- An XOR phase shifter that removes the correlations that result from the shift-register nature of the LFSR output taps

1.2.3 Compressor

The LogicBIST compressor receives and compresses data from the internal chains during the unload process. It consists of an XOR-tree compressor and a multiple-input signature register (MISR). The XOR compressor has no X-tolerance masking.

In the MISR, each register input captures an XOR of the previous register's input and a data input signal from the XOR compressor to the MISR.

1.2.4 Clock controller

The LogicBIST clock controller operates as follows:

- When the design is in mission mode, the clocks operate normally.
- When the design is in a non-LogicBIST scan mode, the clocks operate normally.
- When LogicBIST self-test is active:
  - The clock controller gates the clock signal to the functional design logic as directed by the LogicBIST controller.
  - A free-running BIST clock must be available, running at the desired scan frequency, for the duration of the LogicBIST test operation (see Section 2.1 for exact details)

2 Logic BIST in LPDDR5/4/4X

Logic BIST flow requires that blocks must be X-clean. Chapter 2 presents details on the following aspects of the design that require special consideration:

- Clock control
- Design isolation

2.1 Clock control

During scan mode X-sources within clock domains are blocked inside LPDDR5/4/4X PHY. However, there are multiple clock domain crossing paths which means that, in order to avoid X generation, the design requires its clocks to be one-hot while running LBIST.

By default, the design has no on-chip clocking (OCC) sources, i.e. all clocks in scan mode are external (driven by input ports).

When LogicBIST self-test is active, if the design contains multiple scan clock domains, LogicBIST clock controller drives all scan clock domains with a single BIST clock. The non-BIST clock input ports do not clock any scan chains. However, this contradicts with the one-hot clock requirement.

If capture paths exist between clock domains, additional logic is required to selectively enable non-interacting capture clocks in each pattern. This avoids capturing an X value from an asynchronous clock domain that is also clocked in that pattern. To implement this logic, clocks are separated into groups and assigned a weight to each group. In each pattern, a single clock group is selected for capture, proportionally to the weight values.

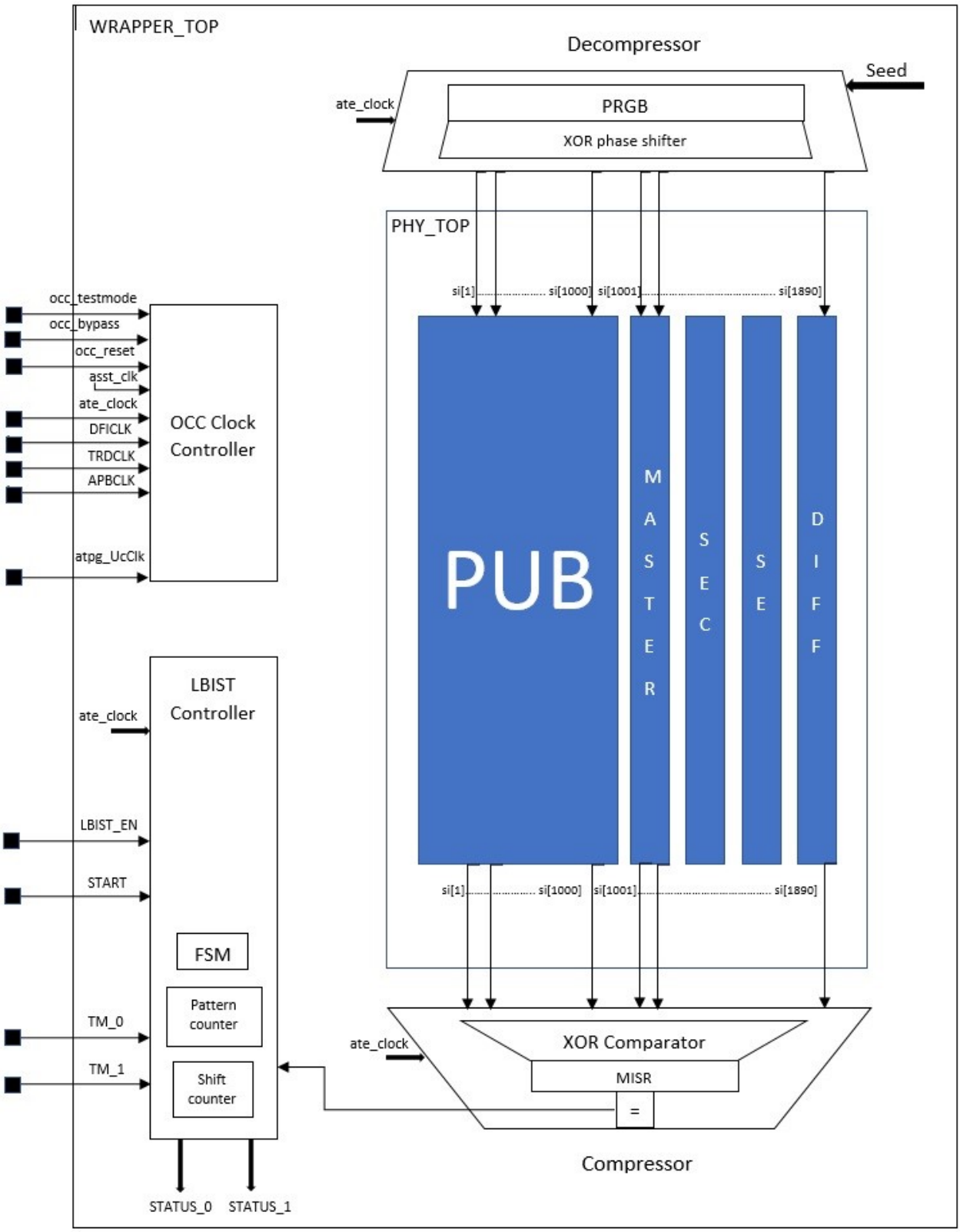
Weighted clock group separation can be achieved during LBIST if OCC clock controllers are inserted on each scan clock.

Weight values should be decided based on the amount of registers controlled by each clock, for example the highest weight should be assigned to DfiClk because the majority of flip-flops inside the design are clocked by this clock.

For more details on OCC customers can refer to On-Chip Clocking Application note.

Figure 2.1 shows LBIST controller, clock controller and PHY_TOP.

Figure 2.1: LBIST + OCC



2.2 Design isolation

The design must be isolated on-chip during LogicBIST self-test. Its inputs must be controlled to avoid X capture and its outputs should be observed to ensure coverage.

Core Wrapping is used as isolation method. The core wrapping feature inserts a wrapper chain that isolates the design during self-test. With this approach, the LogicBIST test mode becomes an inward-facing mode. It drives LogicBIST-generated data into the input wrapper chain and incorporates the captured output wrapper chain data into the MISR

Certain primary inputs and outputs need to be excluded from isolation to maintain correct functionality and to handle the mixed signal nature of the design.

Primary I/O that are excluded from Core Wrapping isolation are listed in Table 2.1

Table 2.1: I/O excluded from Core Wrapping

Port name	Type	Description
PllRefClk	Input	Refence clock used for PLL calibration
atpg_Asst_Clk	Output	Output of PLL that can be used for OCC controllers
dccm_data_ce	Output	Digital output excluded because of UcPhasor register
iccm_data_ce	Output	Digital output excluded because of UcPhasor register
BP_ZN	Output	Analogue output
ZCAL_INT	Output	Analogue output
BP_ATO	Output	Analogue output
RxBypassDataPad_DT*	Output	Analogue output
RxBypassDataPad_DF*	Output	Analogue output
RxBypassDataRcv_DFI*	Output	Analogue output
RxBypassData_DFI*	Output	Analogue output

3 Steps to implement Logic BIST

Chapter 3 focuses on LBIST implementation:

- LogicBIST insertion in DFT Compiler
- Seed generation
- Validation

3.1 LogicBIST insertion

For general LPDDR5/4/4X DFT guidelines, refer to Chapter 7 of DesignWare Cores LPDDR5/4/4X PHY Implementation Guide.

3.1.1 Defining Control signals

To support LBIST insertion *DWC_DDRPHY_LBIST_EN* needs to be defined when synthesizing PHY TOP. Ports added to PHY_TOP interfaced when *DWC_DDRPHY_LBIST_EN* is used are listed in Table 3.1

Table 3.1: LBIST ports

Port name	Type	Description
lbist_mode	Input	LBIST mode Enable. 1 means LBIST mode
LBIST_TM0	Input	LBIST mode select
LBIST_TM1	Input	LBIST mode select
LBIST_EN	Input	LBIST operation enable
START	Input	LBIST operation start
STATUS_0	Input	LBIST run status bit 0
STATUS_1	Input	LBIST run status bit 1

Ports listed in Table 3.1 can be defined with the following commands:

```
set_dft_signal -view existing_dft -port lbist_mode -type Constant -active 1
set_dft_signal -view spec -type TestMode -port {LBIST_TM0 LBIST_TM1}
set_dft_signal -view spec -port LBIST_EN -type lbistEnable
set_dft_signal -view spec -port START    -type lbistStart
set_dft_signal -view spec -port STATUS_0 -type lbistStatus_0
set_dft_signal -view spec -port STATUS_1 -type lbistStatus_1
```

3.1.2 Defining the LogicBIST Self-Test Mode

DFT commands and options related to LogicBIST self-test contain the word “logicbist”. To enable LogicBIST self-test insertion, use the following command:

```
set_dft_configuration -logicbist enable
```

To insert LogicBIST self-test in your design, define a test mode with a usage of `logicbist`. Then, use the `set_logicbist_configuration` command to configure the self-test configuration parameters.

The following example is for a design that uses core wrapping for boundary testability- define the uncompressed inward-facing mode and its corresponding inward-facing scan compression mode:

```
define_test_mode Internal_scan -usage scan -encoding {LBIST_TM1 0 LBIST_TM0 1}
define_test_mode LBIST -usage logicbist -encoding {LBIST_TM1 1 LBIST_TM0 0}
define_test_mode WRPIF -usage wrp_if    -encoding {LBIST_TM1 1 LBIST_TM0 1}

set_logicbist_configuration -test_mode LBIST -base_mode WRPIF \
    -clock ate_clock \
    -chain_count 256 \
    -shift_counter_width 15 \
    -pattern_counter_width 16
```

To apply weights on different clocks, customers can use the following command:

```
set_logicbist_configuration -test_mode LBIST -base_mode WRPIF \
-reset_weights [list [list 2 Reset_async]] \
-occ_clock_weights [list [list 32 OCC_INBUF_DfiClk/X] \
[list 4 OCC_INBUF_atpg_TxDllClk/X] \
[list 4 OCC_INBUF_atpg_RDQSClk/X] \
[list 4 OCC_INBUF_APBCLK/X] \
```

```
[list 4 OCC_INBUF_atpg_PClk/X] \
[list 8 OCC_INBUF_atpg_UcClk/X] \
[list 2 OCC_INBUF_TDRCLK/X] \
[list 2 OCC_INBUF_atpg_DlyTestClk/X]]
```

OCC_INBUF*/X are outputs of each clock controller- see OCC Application Note.

3.1.3 Configuring Wrapper Chain Isolation Logic

To use wrapper chains for self-test isolation, customers can enable both the core wrapper and LogicBIST clients:

```
set_dft_configuration -wrapper enable -logicbist enable
```

To exclude I/O from core wrapping, customers can use:

```
set_boundary_cell -class core_wrapper -type none -ports [get_ports PllRefClk]
-safe_state 0
```

3.1.4 DFT Compiler output

Required outputs:

- Netlist without seed signature
- Test Protocol
- Testbench file

```
write -format verilog -output ./netlist/${current_design}_no_seed_signature.v
-hierarchy

write_test_protocol -test_mode LBIST -output ./netlist/${current_design}_lbist.spf

write_test -format stil -output ${current_design}_lbist_tb
```

3.2 Seed and signature generation

3.2.1 Calculate seed in Tetramax

To calculate the seed and signature value in Tetramax (TMAX), customers can use the netlist and SPF file generated from DFT Compiler.

```
set_drc -seq_comp_jtag_lbist_mode light_lbist
set_drc -allow_unstable_set_resets

run_drc ./netlist/dwc_ddrphy_topwrapper_lbist.spf

source ./find_seed.tcl

find_seed -seed_count 20 {run_atpg -auto -jtag_lbist { 1 32 1 }}
```

The *find_seed* TCL procedure is described in detail in:
https://solvnetplus.synopsys.com/s/article/Finding-Optimal-Seed-Values-for-the-LogicBIST-PRPG-1576170373851

Once seed and signature values are calculated, they are written out to a STIL file:

```
write_patterns ./tmax_lbist.serial.stil -format stil -replace -unified -serial
```

3.2.2 Setting the Seed and Signature Values in DFT Compiler

To set seed and signature values into netlist, customers can use *set_logicbist_constant* TCL procedure in DFT Compiler:

```
source ./set_logicbist_constants.tcl
set_logicbist_constants -file_name ./tmax_lbist.serial.stil
write -f verilog -o ./netlist/${current_design}_seed_signature.v -hier
```

Details about *set_logicbist_constants* can be found in:
https://solvnetplus.synopsys.com/s/article/Setting-the-Seed-and-Signature-Values-in-a-LogicBIST-Design-1577133842512

3.3 Validation

To simulate autonomous LogicBIST operation in VCS, create a testbench from the STIL file generated by the *write_test* command in DFT Compiler:

```
exec         stil2verilog         -replace         ${current_design}_lbist_tb.stil
./netlist/${current_design}_lbist_tb
```

Once testbench is written out, customers can compile and run using standard VCS commands:

```
vcs \
./netlist/dwc_ddrphy_topwrapper_lbist_tb.v \
./netlist/dwc_ddrphy_topwrapper_seed_signature.v \
-F ./sims_files.list  \
+define+BIAS_PINS \
+define+DWC_DDRPHY_PG_PINS \
+define+DWC_DDRPHY_PLL_ACCURATE_MODEL \
+nospecify \
+race=all \
-debug_pp \
-debug_all \
-Mdir=csrc \
-l ./log/run_lbist_vcs_cmp.log \
-o simv \
-full64

./simv -ucli -i ./force.ucli  -l ../log/run_lbist_vcs_sim.log
```

Simulation should complete with no errors:

```
XTB: Starting serial simulation of 0 pattern
XTB: Simulation of 0 patterns completed with 0 mismatches
V C S S i m u l a t i o n R e p o r t
```