Technical Disclosure Commons

Defensive Publications Series

September 06, 2018

# Programmable Test Pattern Generator and Controller

Sandeep Bhatia

# Programmable test pattern generator and controller

## ABSTRACT

Memories are typically tested using preset test patterns that exercise the memory core. In the case of embedded memories, e.g., where memory is integrated within an application-specific integrated circuit (ASIC), conventional testing is not effective in uncovering defects in the memory-ASIC interface. Stress conditions, e.g., excessive bus activity, saturated interface bandwidth, power surges, noise, etc., that arise in operation are not reproduced adequately in the test phase. Memory defects that manifest under specific conditions, e.g., after many cycles of memory access, under certain data/address sequences, when neighboring memory cells hold particular values, etc., are often not uncovered.

This disclosure describes a compact, programmable, built-in test unit capable of generating arbitrary sequences of memory accesses and read/write operations, pattern-loops, address-jumps, etc. The embedded memory receives test patterns from the test unit in a manner that mimics memory accesses from the ASIC. Both memory core and interface are subjected to stress or corner conditions as experienced in operation. Programmability of the test unit enables trapping of conditions that surface rarely-manifested defects.

## KEYWORDS

ASIC testing; memory testing; ATPG; test pattern generation; MBIST; built-in self-test; high-bandwidth memory (HBM); DRAM; design-for-test; embedded memory;

## BACKGROUND

Current memory built-in self-test (MBIST) controllers test for memory defects, e.g., stuck-at, transition, address decoder, and other defects. An MBIST controller typically traverses through the memory, reading and writing data in different patterns to detect faults. However, the focus is generally on core memory-cell defects, rather than on the memory interface or associated data path components. Also, current MBIST solutions rely on hard-coded state machines that only generate fixed test sequences, or have limitations in generating arbitrary test sequences at high speed. Conventional tests, e.g., tests based on ATPG or hard-coded linear feedback shift register sequences, currently lack the following coverage:

- testing the ASIC-memory interface, e.g., the interposer for high-bandwidth memory (HBM);
- saturating the memory and interface access bandwidths;
- measuring power under different types of address/data traffic and sequence of operations;
- switching activity control in data being read or written;
- randomizing address access order;
- measuring performance for relatively complex interfaces such as HBM; etc.

Consequently, stress conditions, e.g., excessive bus activity leading to excessive switching, saturated interface bandwidth, power surges, thermal conditions, noise, etc., that arise in operation are not reproduced adequately in the test phase. Memory defects that manifest under specific conditions, e.g., after many cycles of memory access, under certain data/address sequences, when neighboring memory cells hold particular values, when running under higher speeds with specific address/data sensitivity, when sequential data-access requests are from neighboring cells (or across address boundaries), etc., are often not uncovered.

## DESCRIPTION

This disclosure describes a flexible memory traffic generator for testing memories, memory controllers, and interface logic (e.g. HBM) using micro-controlled test execution. A simple instruction code enables the running of a rich variety of test patterns. Testing goes beyond current and traditional testing of memory cores, with controlled data activity to test the memory-ASIC interface. Test instruction code is loaded, e.g., via CSR or JTAG test access port. Leveraging the structured nature of test patterns, each step of the MBIST program is executed by a compact loop.

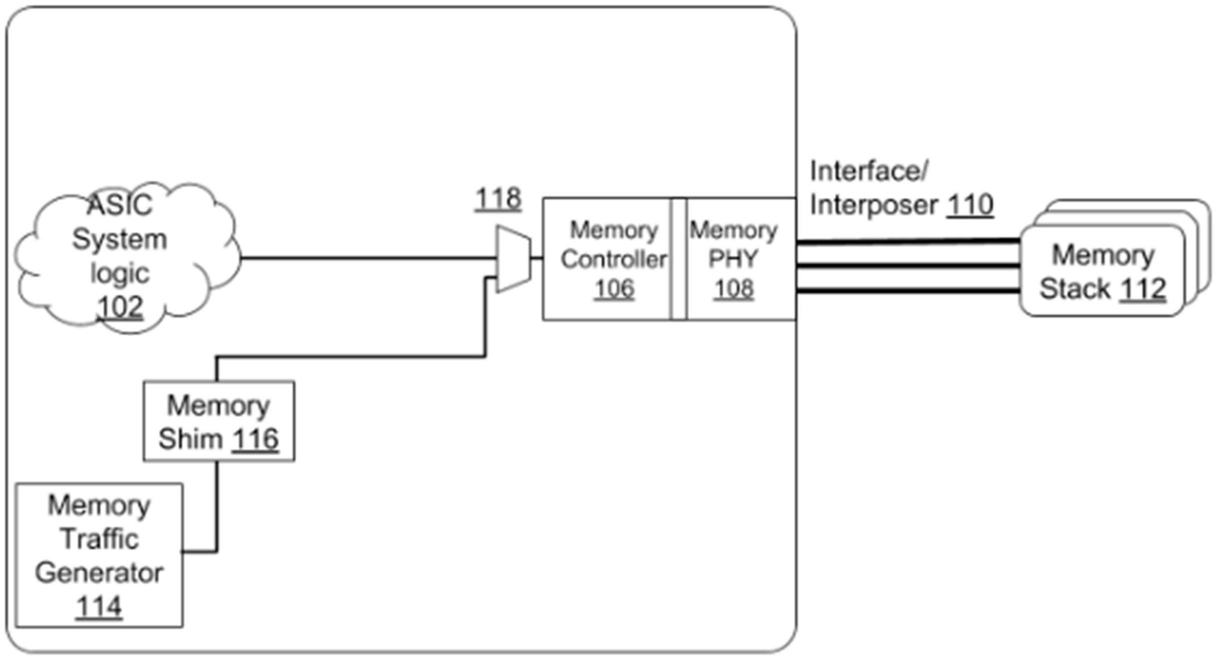

**Fig. 1: Built-in programmable test pattern generator and controller**

Fig. 1 illustrates an example of a programmable test pattern generator and controller integrated into a package that includes ASIC system logic and memory. During operation, the ASIC system logic (102), which is the functional part of the chip, communicates with the memory stack (112), requesting, e.g., data reads/writes from particular memory addresses. The memory can be of a variety of types, e.g., DRAM, HBM, etc. ASIC-memory communication passes through memory controller (106) and a physical interface (108, denoted as PHY). The memory PHY is an interface protocol that defines connectivity between memory controller (106) and physical interface or interposer (110).

Per techniques of this disclosure, a programmable memory traffic generator-controller (114) generates test patterns that are sent to the memory controller by selecting a switch on a multiplexer (118). Memory shim (116) serves to transparently pass through test patterns to the multiplexer (118).

In this manner, the custom, flexible, MBIST generator-controller described herein exercises a diversity of components along the ASIC-memory communications pathway, including memory traffic optimizer, memory controller, PHY, interface, DRAM/HBM stack, etc. Test patterns that originate from the generator-controller pass through various on-package componentry similar to patterns that originate from the ASIC system logic, thereby mimicking real-time operational mode traffic. Further, the controller enables focus on specific test quality measurements, e.g., power profile, performance measurements under different operating conditions, etc.

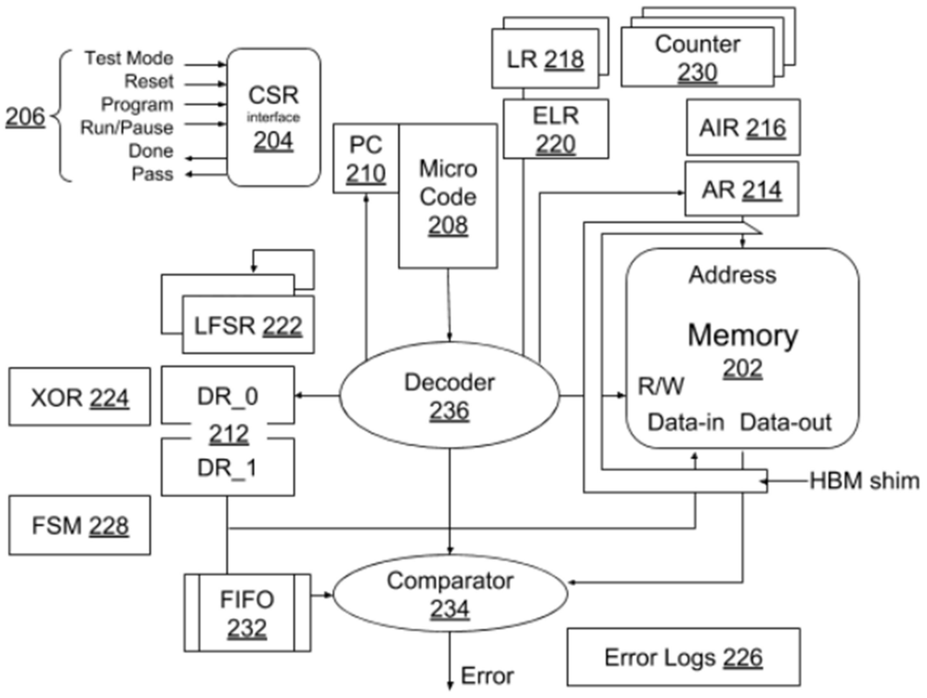

**Fig. 2: Architecture of MBIST controller**

Fig. 2 illustrates an example architecture for the MBIST controller, per techniques of this disclosure. Memory (202) and associated componentry is under test. A test developer uses CSR (or JTAG) interface (204) to control, program, or reprogram the MBIST controller. The interface is also used to log and observe test results. For example, a test developer may use a sequence of operations (206) comprising: switching on a test mode; resetting the chip (erasing memory); downloading test program; running or pausing test program as necessary; awaiting done and pass (or fail) signal; etc. The MBIST controller can be programmed to stop or pause for the purposes of inspection at the first, second, or nth error. A test developer can download test logs, modify the test program, and re-upload and re-run tests in order to isolate defects.

Programmable test code is typically micro-code that is compact, e.g., instructions of a few kilobytes, and is stored in on-board RAM (208) or registers implemented using flip-flops. For example, the micro-code may comprise 32-to-64 words each 12 bits wide. The micro-code memory is expandable as necessary. As explained before, this memory is loaded with a test program during configuration via CSR or JTAG test access port. Despite its compactness, the micro-code is capable of accessing a relatively large memory space.

The MBIST controller further includes the following:

- *Program counters* (210): These maintain a pointer to the current executing instruction and can be, e.g., 4-to-5 bits wide.
- *Data registers* (212): These specify data to be written to (or read from) the memory-under-test. There can be at least two such data registers (`DR_0` and `DR_1`), typically 8-to-16 bits wide, with a parameter to control the width of the data register as necessary. Data within a data register is repeated to drive full memory word for write and read comparison. A data register has operations such as:

  a. loading specific or immediate data;
  b. loading random data from the on-chip linear feedback shift registers;
  c. loading data from address registers, including from next state;
  d. holding data;
  e. swapping data with another data register;
  f. inverting data, e.g., with XOR register to control bits to invert;
  g. rotating-right data;
  h. shifting-left data; etc.

- *Address registers* (214): These specify address to write to (or read from) the memory-under-test. These are as wide as the address-width of memory being tested. An address register has operations including:
  a. holding immediate address;
  b. incrementing address, e.g., using address increment register;
  c. decrementing address, e.g., using address increment register;
  d. loading data from linear feedback shift registers for pseudo-random address access sequence;
  e. shift-loading data, e.g., eight bits at a time; etc.
- *Address increment registers* (216): These create jumps in test address or data patterns. They can be, e.g., 8 bits wide, and store the count by which to increment/decrement address registers (214). Address increment registers reset to 1, but can be loaded with a different value to increment or decrement an address register by a different count.
- *Loop registers* (218): These keep a count of loops within the micro-code and are used to create loops in test patterns. A loop register can be 8-to-16 bits wide, but can be of any

length to allow different desired loop count limits. Multiple loop registers enable nested loops, e.g., two outer loops using two loop registers, one inner loop using address register, etc. Infinite loops to run an unlimited number of read/write operations, e.g., for power measurements, transient error accumulation, etc., can be implemented by setting the loop register within the body of the loop. Loop registers are loaded with a loop value and decrement upon loop iteration.

- *Execution loop registers* (220): These serve as loop counters in single-instruction execution. These are loaded with a loop value and decrement value upon loop iteration.
- *Linear feedback shift registers* (LFSRs; 222): These generate pseudo random address and data test patterns. Dedicated LFSRs can be used for data registers and address registers. LFSRs generate patterns, using, e.g., Galois LFSR, to minimize delay in the combinational logic, with taps e.g., at positions 33 and 20. Normally an LFSR does not generate the all-zero state (or all-one state, when using an inverted feedback). When using a subset, e.g., lower 32 bits, of LFSR bits to drive address/data, the all-zero (or all-one) pattern can appear in the chosen subset of the bits. Operations available for LFSR include loading the LFSR state and shifting to the next state.
- *XOR registers* (224): These perform XOR operations as necessary, e.g., for values stored in data registers. Upon reset, XOR registers are set to all-zero. Example XOR operations are:

  The `XorDR` operation: `DR[0] <= DR[0] ^ XR`;

  The `AR2DR` operation: `DR[0] <= nextAR ^ XR`; etc., where <= denotes “replace by”, `^` denotes the XOR operation, `DR[0]` is the content of a data register, `nextAR` is the content of an address register, and `XR` is the content of the XOR register.

- *Error registers or logs* (226): These store a variety of error-related variables including error counts; last error address; program counter at last error; expected-versus-erroneous data; time-stamp of error; instructions being carried out during/before/after the time of errors; data/address sequence that triggered error; etc. They have a programmable maximum error count parameter to automatically stop once a certain number of errors are reached.
- A *finite state machine* (228): This controls reset, loading, execution and operation of MBIST code, and return of status. The FSM has at least the following controls:
  a. reset, which resets all registers and deactivates the MBIST, e.g., puts the chip in function mode and sets the program counter to zero;
  b. load/program, which loads micro-code;
  c. run, which starts execution of the MBIST. If this signal is deasserted, the FSM pauses MBIST program, but keeps memory operation in test mode;
  d. done, which serves as status of the test run;
  e. pass/fail, which indicates test status; etc.

  The FSM can be paused by back-pressure mechanism, e.g., by controlling its *run* signal.
- *Counters* (230): These keep track of different events and cycles. They can be, e.g., 32-bits wide, and can count, e.g., number of cycles executed between start and end of an MBIST program, number of active read or write cycles, etc. Counters enable

comparisons of the efficiency of memory, as they exclude cycles when the finite state machine is paused.

- A *first-in-first-out* register (FIFO 232): This serves as queueing mechanism for read/write operations. In the case of certain memories, e.g., HBM, read and write operations can get queued, and take multiple and non-constant number cycles to complete. The FIFO is used to store the expected data and match against data being read when such data becomes available. During write, if memory is not ready, the controller pauses and waits until the memory is ready. Similarly, during read, if the read-fifo is full, then it pauses until read-fifo is ready to receive the next data.
- *Comparators* (234): These compare the actual response of the memory-under-test with expected response.
- *Instruction decoder* (236): This decodes instructions and accesses registers, comparators or memory-under-test as necessary.

A debug bus running through the MBIST controller enables the reading or trapping of data and register-values. Loading a register involves, e.g., left-shifting by eight bits, and loading new data in the least-significant eight bits. Before the MBIST runs a test, the support logic for memory subsystem, e.g., memory controller, PHY, HBM, etc., are reset, e.g., initialized or programmed for timing parameters. At the time of start of test, the CSR interface is up, along with a simple firmware image needed to bring up memory subsystem.

The MBIST controller can run a burst of test patterns in loops with controlled, e.g., pseudo-random yet reproducible, data, address, read/write sequences with or without gaps in read/write access cycles. It is important that read/write operations complete in one cycle so as to saturate the memory and interface bandwidth. The techniques of this disclosure are based on VLIW (very long instruction word), such that a load operation occurs in one cycle.

Some advantages of the memory built-in self-test controller and test pattern generator described herein are as follows:

- Ability to test the entire memory *and* interface with controlled, e.g., low/high**,** switching activity with specific or pseudo-random data patterns.
- Ability to programmatically control and mix read/write sequences, e.g., read bursts, write bursts, alternate read/write, controlled bubbles between read/write bursts, etc.
- Ability to access memory in ascending, descending, or (pseudo-)random order of address, within or without specific range, with controllable skip range (to enforce bank hopping).
- Ability to run a structured, or arbitrary pseudo-randomly addressed, set of read/write operations in an endless loop, e.g., for power and performance measurement.
- Ability to perform random read/write operations and memory access by using the address as data for writes and reads.
- Ability to control or change test sequences and patterns at run time.
- Flexibility to program different sequences into the memory of the ASIC system.
- Ability to make the test-generated data traffic appear as if driven by the ASIC system.
- Saturating the memory access bandwidth, e.g., by performing one read/write operation per cycle per controller.
- Coverage of entire memory, e.g., ability to write 1/0 at each bit location, ability to create a specified switching activity on the data path, etc.
- Enabling multiple controllers to connect to memory system and operate synchronously such that controlled yet realistic stress conditions are generated.

- Option to add extension bits to instruction to drive special control signals that can alter every cycle.
- Option to add custom extension registers to drive special control signals that may change less frequently.
- Ability to collect error data, and ability to stop at nth error and alter test program to change the test sequence from that point onward.
- Option to add extension to perform system test applied at bootup for specific components shared with vendors.
- Ability to alter address/data/operation pattern on-the-fly for purposes of diagnosis, by making use of the JUMP instruction and the reprogrammability of micro-code.
- Ability to measure memory performance, power consumption, and bandwidth for specific types of memory access and data patterns.
- Applicable to a variety of memory interfaces, e.g., HBM, LPDDR, etc., by plugging in a specific shim or interface logic.
- Ability to support masking and partial writes, e.g., by applying a mask for write/read comparison operations.
- Ability to support data traffic bursts, e.g., read n words in sequence.
- Controllability of the traffic generator via a variety of interfaces, e.g., JTAG, iJTG, I2C, custom CSR, APB, etc., such that stand-alone setup, e.g., from tester, is possible.
- Independent of vendor of memory PHY/controllers.
- Applicable to a variety of memory types, e.g., DRAM, HBM, etc.
- Integrated testing of components associated with memory, e.g., HBM controller, PHY, interface, HBM stack, etc., in their system mode operation.

- Ability to control and program rich varieties of test patterns and sequences without having to bring up entire system.
- Customizable and tunable to specific regimes of operation, e.g., high data activity, specific read/write access patterns, saturated interface bandwidth, power surges, etc.

In this manner, the techniques of this disclosure enable the generation of flexible, programmable and reconfigurable test patterns that exercise memory and associated data path components, e.g., the memory-ASIC interface, in a manner that mimics operational load conditions. The techniques enable fast debugging of difficult-to-pinpoint issues, in turn leading to quick turnaround of design changes. The test patterns are generated by a compact, built-in self-test unit that can be reprogrammed to diagnose and isolate defects. The techniques contribute to an improvement of the quality of ASICs and embedded memories by enabling quick screening-out of defective parts prior to integration into a larger system.

## CONCLUSION

This disclosure describes a compact, programmable, built-in test unit capable of generating arbitrary sequences of memory accesses and read/write operations, pattern-loops, address-jumps, etc. The embedded memory receives test patterns from the test unit in a manner that mimics memory accesses from the ASIC. Both memory core and interface are subjected to stress or corner conditions as experienced in operation. Programmability of the test unit enables the trapping of conditions that surface rarely-manifested defects.