

RAM BIST: Distributed BIST from Computer Generated VHDL

Robert E. Anderson
VLSI Technology, Inc.
Ste 214, 10250 SW Greenburg Rd.
Portland OR 97223
(rob.anderson@vlsi.com)

1.0 Abstract

A new RAM BIST (Built-In Self Test) methodology is described which provides a high quality at-speed, distributed test solution. Tests can be run at full speed, with multiple unrelated clocks, covering multiple RAM blocks, and to target different RAM topologies. The method suits modern ASICs with multiple RAM blocks and multiple clocks.

The implemented BIST is capable of running at the full system speed, and it is in many ways superior to previous solutions for in-system testing or production testing.

To develop the BIST generators, the author used an iterative prototyping method. As described in this paper, programs were written to automate the development process.

2.0 Introduction

RAM BIST is becoming increasingly important for ASICs. In large ASICs, the fault density is highest in RAM areas, and in many cases the RAM blocks which need to be tested are not directly accessible. Our experience at VLSI is that in-system failure rates of 3 to 5%, or higher, can be expected if RAM is inadequately tested.

This paper addresses a methodology for creating distributed BIST for RAMs. Several features of this solution are an improvement over existing test methods as they are applied to large ASIC designs. The emphasis is on optimal testing, minimal circuitry, and the ability to cover multiple clock domains.

3.0 Test Algorithms

In the ASIC environment, test algorithms have been adapted from more general algorithms aimed at production testing of memory devices.

3.1 Example BIST algorithm

A test algorithm is normally specified shorthand, as a series of marches. Each march is a one directional walk through memory, performing read and write operations at each location. Several marches are combined for a test sequence.

Marches are notated with the direction followed by the operations. For example, "+ra" means read pattern "a", incrementing through memory. "+(ra,wb)" means read pattern "a", write pattern "b", incrementing through memory.

Pattern "a" is chosen to represent a "1010..." pattern for one location of memory. "b" is usually the inverse. It is important that the "1010..." pattern is in physical memory, and not just the representation at the pins of the device.

- Basic BIST Algorithm:

```
+(wa)
+(ra,wb)
+(rb,wa)
-(ra,wb)
-(rb,wa)
+(ra,wb)
-(rb)
```

3.2 Algorithm Assumptions

The driving force behind the development of memory test algorithms was the need to test DRAMs without knowledge of the internal topology.

There are some assumptions which need to be considered when these algorithms are being used for ASICs:

1. It is assumed that read cycles are non-destructive.
2. These algorithms are specified as functional tests, and they ignore timing related failures. The assumption is that access time can be checked otherwise.
3. It is assumed that the address decoders access adjacent locations for incremental addresses.
4. It is assumed that the RAM only has one address, so the algorithm does not handle interactions between addresses.
5. It is assumed that the “a” and “b” patterns can be used to test for a majority of adjacent disturb problems.
6. Normally, for data retention testing, a pause occurs in the middle of the test. This assumes that a pause is the worst case test for data retention.

In practise, these assumptions are often misunderstood or ignored. The designer may have no way to alter the BIST implementation, or he may have no information to measure these assumptions against.

So in practise, using common BIST algorithms we may still see a significant system failure rate due to RAMs, depending on the usage and the RAM topology. However, the test algorithm normally used is a field proven algorithm which catches a majority of primary memory faults.

3.3 The VLSI Two-Port RAM

The VLSI Two-Port RAM is often called internally the “TCRAM”. It is a static RAM with a read address (AR), a write address (AW), a read data port (DR) and a write data port (WD). There is also a write enable (WE). This RAM is the main target of the BIST development, and it is typical of RAMs used in large ASICs.

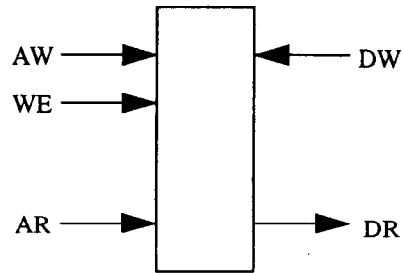


FIGURE 1 Two-Port RAM

We found that this RAM deviated in the following ways from common RAM testing assumptions:

1. Read cycles might be destructive, and are certainly worse than dormant cycles, so read cycles should be performed during “retention” testing.
2. Certain fault mechanisms in this RAM might cause failures which show up as access time violations. (Actually, this is common in memories.) So at-speed testing is desirable.
3. To accommodate the topology, the “a” and “b” patterns had to change based on a permutation of the address. So, if we had just used a normal memory test algorithm as a “black box” test, we wouldn’t have tested very much.
4. For some cases, it was not possible to decide ahead of time what constituted a sufficient variety of march patterns, so for SCAN driven BIST, some highly variable march and pattern generators were created.

4.0 BIST Building Blocks

The problem of constructing a BIST can be broken down into several manageable pieces.

The BIST is running at full speed, so the data, control, and address paths are all pipelined. The pipelines might be clocked by separate clocks for the read and write paths, or by the same clock. The general scheme has to accommodate two clocks.

There may be a specialized controller to automate the BIST, or simple control registers from a SCAN chain may be used. In any case, the clock for this controller is different than any

clock used in the rest of the BIST, so synchronization is performed on the critical signals.

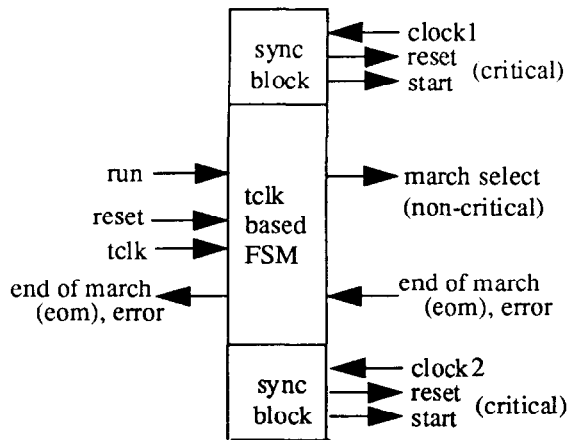


FIGURE 2 Controller for two clock BIST

The controller dictates a sequence of marches to perform. For example to perform the IFA9, 9 marches would be dictated and run by the controller. For each march, a reset is issued to each critical BIST circuit, the march is selected, then a start pulse is issued.

For each clock (critical region) there must be an **address generator**, which is a state machine with a few counters. The **address generator** runs a march based on inputs from a control register and a start signal. There are outputs to sequence the valid address and data patterns in such a way that the access time is tested.

There are two special signal groups worth noting on the address generator: There is an EOM_IN, EOM_OUT daisy chain which is used to cascade address units in case there are several. There is also a SYNC_IN, SYNC_OUT pair of signals, on the multi-clock version, to accomplish token passing.

Address generators may be shared by several RAMs.

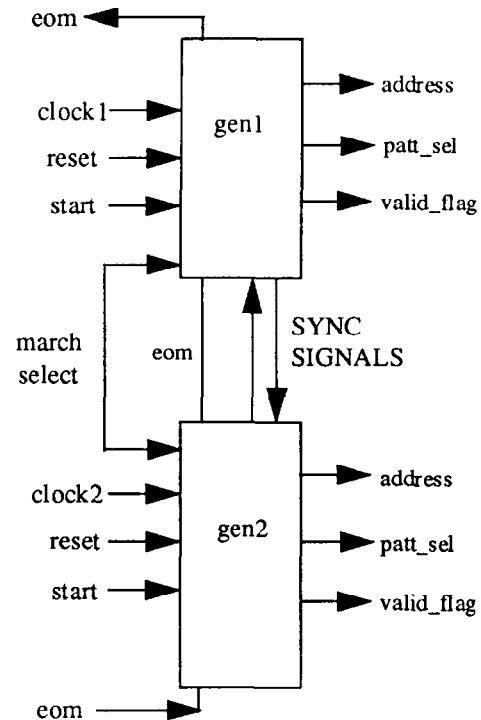


FIGURE 3 Two Connected Address Generators

In Figure 3, clock1 and clock2 are different, unrelated clocks. An operation on gen1 might take several cycles, then a single phase transition will be made on the gen1 sync_out signal. This signal is synchronized by gen2. When gen2 sees the transition, it accepts control (the token) and starts an operation. On completion of its operation, gen2 signals to gen1 in a similar manner. This type of signalling is called "single phase" signalling for obvious reasons.

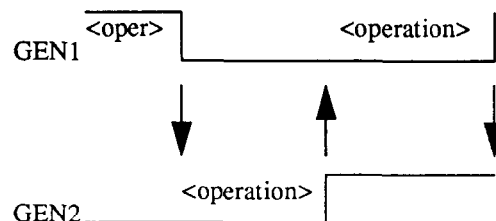


FIGURE 4 Sync_out Signalling to pass Token

Specific to each type of RAM, there is a **pattern generator**, which translates the encoded pattern selection from the **address generator** into the “a” or “b” pattern necessary for the RAM. There are two pattern generators for each RAM to be tested.

To check the output data, there is another block called a **comparator**, which compares the

cycle and compared the following cycle, setting the error flag if appropriate.

As soon as the address is set invalid, the token is passed to the WRITE x1_addr unit. This unit puts out a valid address for one cycle, and valid data for one cycle. It also puts out a valid flag to be used for the WE signal.

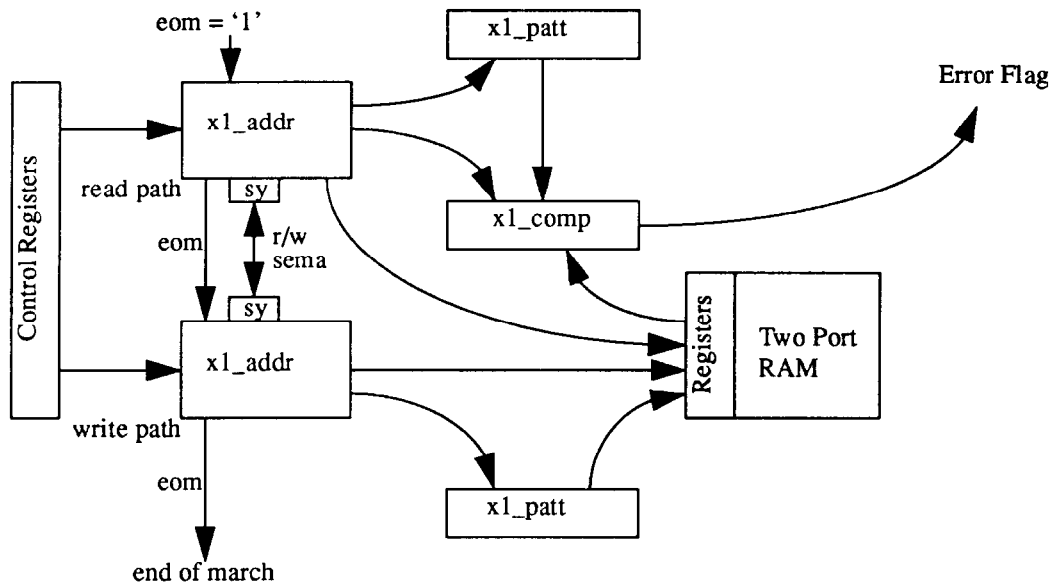


FIGURE 5 BIST System Diagram

memory data to a pattern from the **pattern generator**, and updates an error flag at the right time. There is one comparator for each RAM to be tested, corresponding to the read data port.

For an example of the operation, assume there is a march +(ra,wb).

The controller resets both the read and write x1_addr units and all other BIST registers. Then the march type and eom flag are set.

The controller then asserts start to the read channel x1_addr unit. This gives the token to that unit.

The read channel then feeds a valid address into the read address register, for one cycle. The data which comes out is registered after one

For each x1_addr unit, after a cycle the counter is checked. If this is the last operation, the eom flag is set and if another token is received it will be dropped.

Several “valid_cycle” flags and pattern selection vectors can be issued from one address generator, to accommodate several RAMs of different sizes and topologies.

Multiple address generators can be connected in the same EOM daisy chain to allow for different clock schemes. Each will need a separate “start” signal from the controller, synchronized to its clock.

5.0 Developing the Code Generators

A prototyping approach was used to develop the code generators. Early on, several factors were identified as being important:

- The VHDL RTL code must compile without changes, into a minimum size circuit that would meet the speed requirement.
- The address generators must be shareable, so they would need to have different port declarations and internal implementations depending on the target.
- There might be several different comparators or pattern generators per BIST circuit, so these should be separately generated.
- A large variety of march types should be accommodated.

These factors in mind, an iterative, prototyping approach was taken to develop the generators.

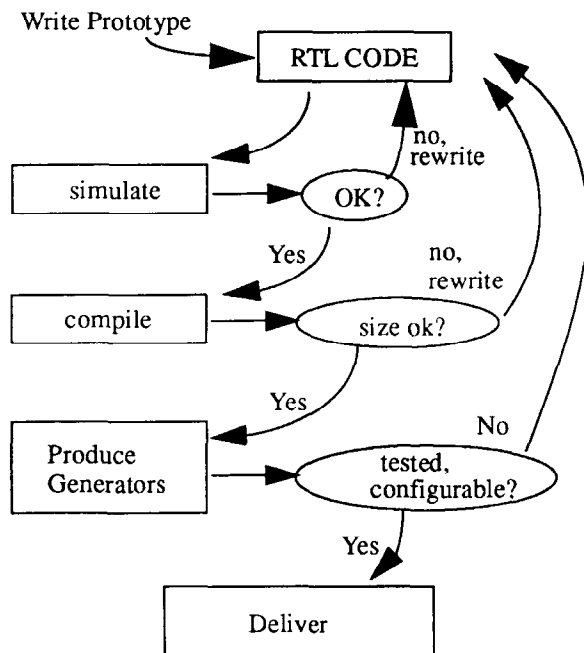


FIGURE 6 Iterative Development Flow

First the RTL VHDL code was written for the various blocks. This was test simulated in a test-bench.

Then the code was test compiled (synthesized). Various parts had to be rewritten, and

blocks had to be partitioned so that the optimizer could handle them.

The code size expanded quite a bit at the stage where the size and speed were checked. For example, it was found that the range comparison on several counters could be done more economically by noting the transition for one carry bit. To take advantage of the savings, the control logic had to be rewritten in a much more complicated form.

The generator programs were produced from a semi-automated flow:

- A program was written to read in a VHDL source, and write a C++ program which would regenerate that source.
- Several such C++ programs were hand edited and combined.
- Sections of customized code were added to change the entity names, set the proper sizes for vectors, add a help screen, and parse input parameters.

This way of producing the generator programs proved to be very productive. Of course, once the generators were written the C++ code had to be modified instead of the VHDL code but by this time the code was relatively stable.

6.0 Open Issues, Research Topics

There are some open issues related to BIST, some modelling issues, and some productization issues. These issues were encountered but only partly resolved in this development.

6.1 BIST issues

- Fault Coverage

People are used to seeing fault coverage claims in the literature, and often these have been misapplied. A proper statistical analysis of fault coverage would require fault analysis data. The latter is not likely to be tractable if it comes from ASIC fault analysis, because there is not enough data and the focus of ASIC vendors is not there.

- Topology Generalization

This is an issue which comes up when you are trying to do a general BIST program. The suggestion is made "why don't you come up with a simple syntax to describe memory topol-

ogies, then the program would not have to be changed?" There are two problems here:

1) These architectures are extremely variable and it is hard to predict the differences.

2) It is hard, in practise, to cover the nuances of even one class of RAM as it varies for different sizes and shapes. The layout information of interest is not available in a useful form.

6.2 Modelling Issues

- Stochastic Testing

It is hard to write testbenches to do stochastic testing, which would be necessary to really verify this kind of design. One problem would be to produce two unrelated clocks.

- Testing the Synthetic Model

ASIC libraries are too simple for this kind of modelling. For example, ASIC library flipflops and other components simply give errors and assume unknown values ('u') when there are timing violations. For this kind of modelling, better control of such "error" handling is necessary.

6.3 Productization Issues

- Complexity

Designers are used to creating one BIST circuit per RAM block. It is difficult to productize and deliver a distributed solution. Perhaps what

is needed is a combined SCAN/BIST product for general logic as well as memory.

- Tester Use

Special vectors are required to run, say, 64000 clocks for each step in the test program. The generation of test programs needs to be addressed closer to the system level where testbenches can be used to automate the task. This is not normally done today.

7.0 Conclusions

The need for BIST in general, and RAM BIST in particular, is increasing as ASIC densities increase. Difficult access, and high relative defect densities are the driving forces.

This paper describes a novel distributed approach which can accommodate complex clocking requirements, at-speed, with a flexible level of testing.

As a development expedient, VHDL RTL code can be produced from a BIST generator, without losing any efficiency compared to a gate level (neelist) generator. A development flow which worked well is described in this paper.