

Built-In Self-Test architectures

Oprîtoiu Flavius
flavius.opritoiu@cs.upt.ro

September 18, 2023

Introduction

Objectives:

- ▶ Configure a BIST architecture

Errors are defined with respect to a system's service (its intended functionality) [ALRH04]. The system's service represents a sequence of system's external states and, in this context, an error occurs when at least one of system's external states deviates from the intended, correct behavior [ALRH04].

A *fault* is the hypothesized cause of an error and, in this context, *fault tolerance* offers means for attaining dependability and security in computer systems by avoiding service failures in the presences of faults [ALRH04].

Error detection of combinational circuits

Error detection techniques identify the presence of an error and can be classified into:

- **concurrent**, or
- **preemptive**

Concurrent error detection mechanisms function during system's normal operation while preemptive methods suspend the normal operation and bring the system into a test mode [ALRH04].

Adding a concurrent test method to a design enhances it with *self-checking* capabilities, in that the system is capable of verifying its correct operation.

Concurrent error detection

The concurrent error detection mechanisms include the following methods:

- hardware duplication,
- code-based redundancy, and
- time-redundancy

Hardware duplication adds a copy to the module that needs protection, whose outputs are compared. Difference between outputs of the protected and the copy signal presence of errors.

Code redundancy verifies that the output of the protected module preserve a so called *invariant property*. *Example*: For a unit receiving 2 unsigned values that are multiples of 3 and calculates the sum of the two, an invariant property (consider no overflow occurs) is that the result needs to be a multiple of 3.

Preemptive error detection

Preemptive error detection methods suspend the normal operation of the system and brings it into a test mode [ALRH04]. Two preemptive detection mechanisms, in used today, are:

- **scan testing**, and
- **Built-In Self-Test (BIST)**

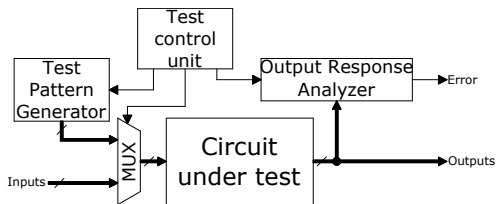
Scan testing modify a sequential design by serially connecting all/the majority of storage elements into dedicated **scan channels**, which are nothing more than shift registers with external access.

BIST method, transforms a design into a self testable architecture, capable of detecting the presence of errors in an autonomous manner, making it suitable for safety-critical applications.

BIST

BIST error detection method transforms a design into a self testable architecture, capable of detecting the presence of errors in an autonomous manner.

The typical BIST architecture is depicted bellow:



The Test Pattern Generator (TPG) generates test vectors to be delivered on Circuit Under Test (CUT)'s inputs. The Output Response Analyzer (ORA) analyzes CUT's results for detecting errors. For a combinational CUT, for each test vector applied a response vector is obtained at CUT's output.

TPG unit

The TPG unit can be implemented using:

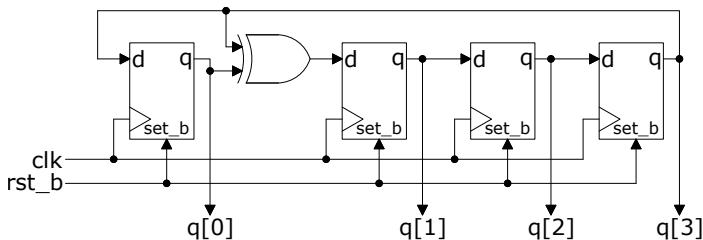
- binary counters, and
- Linear Feedback Shift Registers (LFSRs)

Binary counters generate all input configurations for the CUT, exhaustively.

LFSR represent typical mechanisms for generating test vectors in BIST architectures. They are constructed as shift registers with a feedback connection, operated by EXOR gates.

LFSRs

The figure below describes a LFSR structure, on 4 ranks:



















When initialized with a non-zero vector, a LFSR generates, at its output, a pseudo-random, repeating sequence.

For the architecture above, the output sequence, composed of 4-bit vectors, repeats with a periodicity of 15 (all 4-bit vectors are generated, excluding the all-zero configuration).

LFSRs (contd.)

The 15 vectors sequence, generated at the output of the above LFSR architecture is depicted bellow:

rst_b	clk	q[3]	q[2]	q[1]	q[0]
0	<i>d</i>	1	1	1	1
1		1	1	0	1
1		1	0	0	1
1		0	0	0	1
1		0	0	1	0
1		0	1	0	0
1		1	0	0	0
1		0	0	1	1
1		0	1	1	0
1		1	1	0	0
1		1	0	1	1
1		0	1	0	1
1		1	0	1	0
1		0	1	1	1
1		1	1	1	0
1		1	1	1	1
1		1	1	0	1

↑ Output sequence periodicity ↓

ORA unit

ORA performs data compaction (with loss of information) by processing all CUT's responses while exercised with the test vectors generated by the TPG. At the end of the compaction process, ORA provides a *signature*. The signature is a reduced, fixed-size vector, characterizing the entire set of results.

The signature for a CUT is associated to the TPG unit generating CUT's input vectors. The *gold signature* refers to the signature obtained for the correct, fault-free, circuit. It is usually procured through simulation.

The presence of errors in a CUT is detected by comparing the obtained signature with the gold signature.

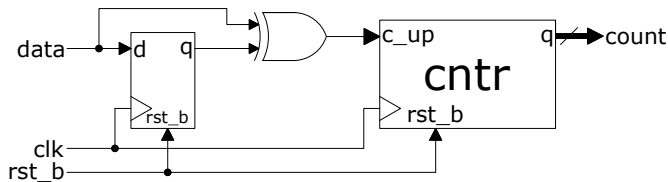
ORA components can be implemented using:

- counting techniques, and
- signature registers

Counting techniques

Counting technique methods can count either the number of times a logic value is generated on an output line, or the number of transitions on the output line. For counting a specific logic value (1 or 0) on an output line, binary counters can be used.

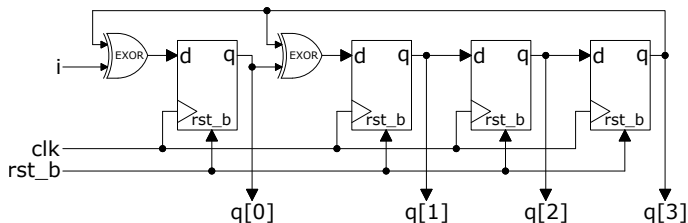
A transition counter is depicted bellow:



A counter unit is connected on each line of CUT's output. In consequence, a 4-bit output requires 4 counter instances. For an output line, the final signature is represented by the content of the counter after receiving all bits of that line.

Signature registers

A **Single Input Signature Register (SISR)** is constructed around a LFSR architecture, having one, additional, data line input. The SISR constructed from the LFSR architecture previously presented is depicted below:



The SISR needs to be initialized with the all-zero configuration.

A SISR unit is connected on each line of a CUT's output and the signature represents the content of the SISR after processing all bits on that line.

References

- [ALRH04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.