Module 8

Testing of Embedded System

Lesson 42

On-line Testing of Embedded Systems

Instructional Objectives

After going through this lesson the student would be able to

- Explain the meaning of the term On-line Testing
- Describe the main issues in on-line testing and identify applications where on-line testing are required for embedded systems
- Distinguish among concurrent and non-concurrent testing and their relations with BIST and on-line testing
- Describe an application of on-line testing for System-on-Chip

On-line Testing of Embedded Systems

1. Introduction

EMBEDDED SYSTEMS are computers incorporated in consumer products or other devices to perform application-specific functions. The product user is usually not even aware of the existence of these systems. From toys to medical devices, from ovens to automobiles, the range of products incorporating microprocessor-based, software controlled systems has expanded rapidly since the introduction of the microprocessor in 1971. The lure of embedded systems is clear: They promise previously impossible functions that enhance the performance of people or machines. As these systems gain sophistication, manufacturers are using them in increasingly critical applications— products that can result in injury, economic loss, or unacceptable inconvenience when they do not perform as required.

Embedded systems can contain a variety of computing devices, such as microcontrollers, application-specific integrated circuits, and digital signal processors. A key requirement is that these computing devices continuously respond to external events in real time. Makers of embedded systems take many measures to ensure safety and reliability throughout the lifetime of products incorporating the systems. Here, we consider techniques for identifying faults during normal operation of the product—that is, online-testing techniques. We evaluate them on the basis of error coverage, error latency, space redundancy, and time redundancy.

2. Embedded-system test issues

Cost constraints in consumer products typically translate into stringent constraints on product components. Thus, embedded systems are particularly cost sensitive. In many applications, low production and maintenance costs are as important as performance.

Moreover, as people become dependent on computer-based systems, their expectations of these systems' availability increase dramatically. Nevertheless, most people still expect significant downtime with computer systems—perhaps a few hours per month. People are much less patient with computer downtime in other consumer products, since the items in question did not demonstrate this type of failure before embedded systems were added. Thus, complex consumer products with high availability requirements must be quickly and easily repaired. For this reason, automobile manufacturers, among others, are increasingly providing online detection and diagnosis, capabilities previously found only in very complex and expensive applications

such as aerospace systems. Using embedded systems to incorporate functions previously considered exotic in low-cost, everyday products is a growing trend.

Since embedded systems are frequently components of mobile products, they are exposed to vibration and other environmental stresses that can cause them to fail. Embedded systems in automotive applications are exposed to extremely harsh environments, even beyond those experienced by most portable devices. These applications are proliferating rapidly, and their more stringent safety and reliability requirements pose a significant challenge for designers. Critical applications and applications with high availability requirements are the main candidates for online testing.

Embedded systems consist of hardware and software, each usually considered separately in the design process, despite progress in the field of hardware-software co design. A strong synergy exists between hardware and software failure mechanisms and diagnosis, as in other aspects of system performance. System failures often involve defects in both hardware and software. Software does not “break” in the common sense of the term. However, it can perform inappropriately due to faults in the underlying hardware or specification or design flaws in either hardware or software. At the same time, one can exploit the software to test for and respond to the presence of faults in the underlying hardware.

Online software testing aims at detecting design faults (bugs) that avoid detection before the embedded system is incorporated and used in a product. Even with extensive testing and formal verification of the system, some bugs escape detection. Residual bugs in well-tested software typically behave as intermittent faults, becoming apparent only in rare system states. Online software testing relies on two basic methods: acceptance testing and diversity [1]. Acceptance testing checks for the presence or absence of well-defined events or conditions, usually expressed as true-or-false conditions (predicates), related to the correctness or safety of preceding computations. Diversity techniques compare replicated computations, either with minor variations in data (data diversity) or with procedures written by separate, unrelated design teams (design diversity). This chapter focuses on digital hardware testing, including techniques by which hardware tests itself, built-in self-test (BIST). Nevertheless, we must consider the role of software in detecting, diagnosing, and handling hardware faults. If we can use software to test hardware, why should we add hardware to test hardware? There are two possible answers. First, it may be cheaper or more practical to use hardware for some tasks and software for others. In an embedded system, programs are stored online in hardware-implemented memories such as ROMs (for this reason, embedded software is sometimes called firmware). This program storage space is a finite resource whose cost is measured in exactly the same way as other hardware. A function such as a test is “soft” only in the sense that it can easily be modified or omitted in the final implementation.

The second answer involves the time that elapses between a fault’s occurrence and a problem arising from that fault. For instance, a fault may induce an erroneous system state that can ultimately lead to an accident. If the elapsed time between the fault’s occurrence and the corresponding accident is short, the fault must be detected immediately. Acceptance tests can detect many faults and errors in both software and hardware. However, their exact fault coverage is hard to measure, and even when coverage is complete, acceptance tests may take a long time to detect some faults. BIST typically targets relatively few hardware faults, but it detects them quickly.

These two issues, cost and latency, are the main parameters in deciding whether to use hardware or software for testing and which hardware or software technique to use. This decision requires system-level analysis. We do not consider software methods here. Rather, we emphasize the appropriate use of widely implemented BIST methods for online hardware testing. These methods are components in the hardware-software trade-off.

3. Online testing

Faults are physical or logical defects in the design or implementation of a digital device. Under certain conditions, they lead to errors—that is, incorrect system states. Errors induce failures, deviations from appropriate system behavior. If the failure can lead to an accident, it is a hazard. Faults can be classified into three groups: design, fabrication, and operational. Design faults are made by human designers or CAD software (simulators, translators, or layout generators) during the design process. Fabrication defects result from an imperfect manufacturing process. For example, shorts and opens are common manufacturing defects in VLSI circuits. Operational faults result from wear or environmental disturbances during normal system operation. Such disturbances include electromagnetic interference, operator mistakes, and extremes of temperature and vibration. Some design defects and manufacturing faults escape detection and combine with wear and environmental disturbances to cause problems in the field. Operational faults are usually classified by their duration:

- *Permanent faults* remain in existence indefinitely if no corrective action is taken. Many are residual design or manufacturing faults. The rest usually occur during changes in system operation such as system start-up or shutdown or as a result of a catastrophic environmental disturbance such as a collision.
- *Intermittent faults* appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. When intermittent faults are present, the system works well most of the time but fails under atypical environmental conditions.
- *Transient faults* appear and disappear quickly and are not correlated with each other. They are most commonly induced by random environmental disturbances.

One generally uses online testing to detect operational faults in computers that support critical or high-availability applications. The goal of online testing is to detect fault effects, or errors, and take appropriate corrective action. For example, in some critical applications, the system shuts down after an error is detected. In other applications, error detection triggers a reconfiguration mechanism that allows the system to continue operating, perhaps with some performance degradation. Online testing can take the form of external or internal monitoring, using either hardware or software. Internal monitoring, also called self-testing, takes place on the same substrate as the circuit under test (CUT). Today, this usually means inside a single IC—a system on a chip. There are four primary parameters to consider in designing an online-testing scheme:

- *error coverage*—the fraction of modeled errors detected, usually expressed as a percentage. Critical and highly available systems require very good error coverage to minimize the probability of system failure.
- *error latency*—the difference between the first time an error becomes active and the first time it is detected. Error latency depends on the time taken to perform a test and how often tests are executed. A related parameter is fault latency, the difference between the onset of the fault and its detection. Clearly, fault latency is greater than or equal to error latency, so when error latency is difficult to determine, test designers often consider fault latency instead.
- *space redundancy*—the extra hardware or firmware needed for online testing.
- *time redundancy*—the extra time needed for online testing.

The ideal online-testing scheme would have 100% error coverage, error latency of 1 clock cycle, no space redundancy, and no time redundancy. It would require no redesign of the CUT and impose no functional or structural restrictions on it. Most BIST methods meet some of these constraints without addressing others. Considering all four parameters in the design of an online-

testing scheme may create conflicting goals. High coverage requires high error latency, space redundancy, and/or time redundancy. Schemes with immediate detection (error latency equaling 1) minimize time redundancy but require more hardware. On the other hand, schemes with delayed detection (error latency greater than 1) reduce time and space redundancy at the expense of increased error latency. Several proposed delayed-detection techniques assume equiprobability of input combinations and try to establish a probabilistic bound on error latency [2]. As a result, certain faults remain undetected for a long time because tests for them rarely appear at the CUT's inputs.

To cover all the operational fault types described earlier, test engineers use two different modes of online testing: concurrent and non-concurrent. Concurrent testing takes place during normal system operation, and non-concurrent testing takes place while normal operation is temporarily suspended. One must often overlap these test modes to provide a comprehensive online-testing strategy at acceptable cost.

4. Non-concurrent testing

This form of testing is either event-triggered (sporadic) or time-triggered (periodic) and is characterized by low space and time redundancy. Event triggered testing is initiated by key events or state changes such as start-up or shutdown, and its goal is to detect permanent faults. Detecting and repairing permanent faults as soon as possible is usually advisable. Event-triggered tests resemble manufacturing tests. Any such test can be applied online, as long as the required testing resources are available. Typically, the hardware is partitioned into components, each exercised by specific tests. RAMs, for instance, are tested with manufacturing tests such as March tests [3].

Time-triggered testing occurs at predetermined times in the operation of the system. It detects permanent faults, often using the same types of tests applied by event-triggered testing. The periodic approach is especially useful in systems that run for extended periods during which no significant events occur to trigger testing. Periodic testing is also essential for detecting intermittent faults. Such faults typically behave as permanent faults for short periods. Since they usually represent conditions that must be corrected, diagnostic resolution is important. Periodic testing can identify latent design or manufacturing flaws that appear only under certain environmental conditions. Time-triggered tests are frequently partitioned and interleaved so that only part of the test is applied during each test period.

5. Concurrent testing

Non-concurrent testing cannot detect transient or intermittent faults whose effects disappear quickly. Concurrent testing, on the other hand, continuously checks for errors due to such faults. However, concurrent testing is not particularly useful for diagnosing the source of errors, so test designers often combine it with diagnostic software. They may also combine concurrent and non-concurrent testing to detect or diagnose complex faults of all types.

A common method of providing hardware support for concurrent testing, especially for detecting control errors, is a watchdog timer [4]. This is a counter that the system resets repeatedly to indicate that the system is functioning properly. The watchdog concept assumes that the system is fault-free—or at least alive—if it can reset the timer at appropriate intervals. The ability to perform this simple task implies that control flow is correctly traversing timer-reset points. One can monitor system sequencing very precisely by guarding the watchdog- reset operations with software-based acceptance tests that check signatures computed while control

flow traverses various checkpoints. To implement this last approach in hardware, one can construct more complex hardware watchdogs.

A key element of concurrent testing for data errors is redundancy. For example, the duplication-with-comparison (DWC) technique5 detects any single error at the expense of 100% space redundancy. This technique requires two copies of the CUT, which operate in tandem with identical inputs. Any discrepancy in their outputs indicates an error. In many applications, DWC's high hardware overhead is unacceptable. Moreover, it is difficult to prevent minor timing variations between duplicated modules from invalidating comparison.

A possible lower-cost alternative is time redundancy. A technique called double execution, or retry, executes critical operations more than once at diverse time points and compares their results. Transient faults are likely to affect only one instance of the operation and thus can be detected. Another technique, re-computing with shifted operands (RESO) [5] achieves almost the same error coverage as DWC with 100% time redundancy but very little space redundancy. However, no one has demonstrated the practicality of double execution and RESO for online testing of general logic circuits.

A third, widely used form of redundancy is information redundancy—the addition of redundant coded information such as a parity-check bit[5]. Such codes are particularly effective for detecting memory and data transmission errors, since memories and networks are susceptible to transient errors. Coding methods can also detect errors in data computed during critical operations.

6. Built-in self-test

For critical or highly available systems, a comprehensive online-testing approach that covers all expected permanent, intermittent, and transient faults is essential. In recent years, BIST has emerged as an important method of testing manufacturing faults, and researchers increasingly promote it for online testing as well.

BIST is a design-for-testability technique that places test functions physically on chip with the CUT, as illustrated in Figure 42.1. In normal operating mode, the CUT receives its inputs from other modules and performs the function for which it was de-signed. In test mode, a test pattern generator circuit applies a sequence of test patterns to the CUT, and a response monitor evaluates the test responses. In the most common type of BIST, the response monitor compacts the test responses to form fault signatures. It compares the fault signatures with reference signatures generated or stored on chip, and an error signal indicates any discrepancies detected. We assume this type of BIST in the following discussion.

In developing a BIST methodology for embedded systems, we must consider four primary parameters related to those listed earlier for online-testing techniques:

- *fault coverage*—the fraction of faults of interest that the test patterns produced by the test generator can expose and the response monitor can detect. Most monitors produce a fault-free signature for some faulty response sequences, an undesirable property called aliasing.
- *test set size*—the number of test patterns produced by the test generator. Test set size is closely linked to fault coverage; generally, large test sets imply high fault coverage. However, for online testing, test set size must be small to reduce fault and error latency.
- *hardware overhead*—the extra hardware needed for BIST. In most embedded systems, high hardware overhead is not acceptable.

- *performance penalty*—the impact of BIST hardware on normal circuit performance, such as worst-case (critical) path delays. Overhead of this type is sometimes more important than hardware overhead.

System designers can use BIST for non-concurrent, online testing of a system's logic and memory[6]. They can readily configure the BIST hardware for event-triggered testing, tying the BIST control to the system reset so that testing occurs during system start-up or shutdown. BIST can also be designed for periodic testing with low fault latency. This requires incorporating a test process that guarantees the detection of all target faults within a fixed time.

Designers usually implement online BIST with the goals of complete fault coverage and low fault latency. Hence, they generally design the test generator and the response monitor to guarantee coverage of specific fault models, minimum hardware overhead, and reasonable test set size. Different parts of the system meet these goals by different techniques.

Test generator and response monitor implementations often consist of simple, counter like circuits; especially linear- feedback shift registers [5]. An LFSR is formed from standard flip-flops, with outputs of selected flip-flops being fed back (modulo 2) to its inputs. When used as a test generator, an LFSR is set to cycle rapidly through a large number of its states. These states, whose choice and order depend on the LFSR's design parameters, define the test patterns. In this mode of operation, an LFSR is a source of pseudorandom tests that are, in principle, applicable to any fault and circuit types. An LFSR can also serve as a response monitor by counting (in a special sense) the responses produced by the tests. After receiving a sequence of test responses, an LFSR response monitor forms a fault signature, which it compares to a known or generated good signature to determine whether a fault is present.

Ensuring that fault coverage is sufficiently high and the number of tests is sufficiently low are the main problems with random BIST methods. Researchers have proposed two general approaches to preserve the cost advantages of LFSRs while greatly shortening the generated test sequence. One approach is to insert test points in the CUT to improve controllability and observability. However, this approach can result in performance loss. Alternatively, one can introduce some determinism into the generated test sequence—for example, by inserting specific "seed tests" known to detect hard faults.

Some CUTs, including data path circuits, contain hard-to detect faults that are detectable by only a few test patterns, denoted T_{hard}. An N-bit LSFR can generate a sequence that eventually includes 2^N - 1 patterns (essentially all possibilities). However, the probability that the tests in T_{hard} will appear early in the sequence is low. In such cases, one can use deterministic testing, which tailors the generated test sequence to the CUT's functional properties, instead of random testing. Deterministic testing is especially suited to RAMs, ROMs, and other highly regular components. A deterministic technique called transparent BIST [3] applies BIST to RAMs while preserving the RAM contents—a particularly desirable feature for online testing. Keeping hardware overhead acceptably low is the main difficulty with deterministic BIST.

A straightforward way to generate a specific test set is to store it in a ROM and address each stored test pattern with a counter. Unfortunately, ROMs tend to be much too expensive for storing entire test sequences. An alternative method is to synthesize a finite-state machine that directly generates the test set. However, the relatively large test set size and test vector width, as well as the test set's irregular structure, are much more than current FSM synthesis programs can handle.

Another group of test generator design methods, loosely called deterministic, attempt to embed a complete test set in a specific generated sequence. Again the generated tests must meet the coverage, overhead, and test size constraints we've discussed. An earlier article [7] presents a representative BIST design method for data path circuits that meets these requirements. The test

generator's structure, based on a twisted-ring counter, is tailored to produce a regular, deterministic test sequence of reasonable size. One can systematically rescale the test generator as the size of anon-bit-sliced data path CUT, such as a carry-look-ahead adder, changes. Instead of using an LFSR, a straightforward way to compress test response data and produce a fault signature is to use an FSM or an accumulator. However, FSM hardware overhead and accumulator aliasing are difficult parameters to control. Keeping hardware overhead acceptably low and reducing aliasing are the main difficulties in response monitor design.

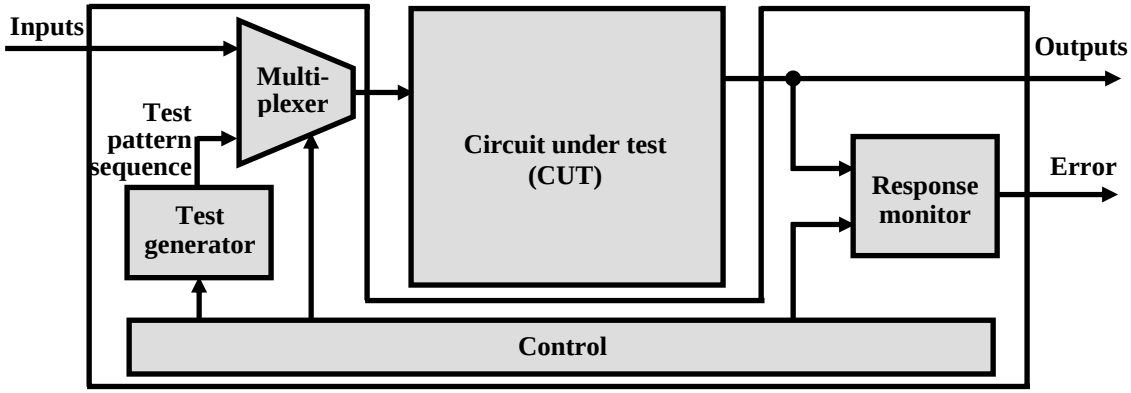

Fig. 42.1 A General BIST Scheme

An Example

IEEE 1149.4 based Architecture for OLT of a Mixed Signal SoC

Analog/mixed signal blocks like DCDC converters, PLLs, ADCs, etc. and digital modules like application specific processors, micro controllers, UATRs, bus controllers etc. typically exist in SoCs. The have been used as cores of the SoC benchmark "Controller for Electro-Hydraulic Actuators" which is being used as the case study. It is to be noted that this case study is used only for illustration and the architecture is generic which applies for all Mixed Signal SoCs.

All the digital blocks like instruction specific processor, microcontroller, bus controller etc. have been designed with OLT capability using the CAD tool descried in [8]. Further, all these digital cores are IEEE 1149.1 compliant. In other words, all the digital cores are designed with a blanket comprising an on-line monitor and IEEE 1149.1 compliance circuitry. For the analog modules the observer have been designed using ADCs and digital logic [9]. The test blanket for the analog/mixed signal cores comprises IEEE 1149.4 circuitry. A dedicated test controller is designed and placed on-chip that schedules the various lines tests during the operation of the SoC. The block diagram of the SoC being used as the case study is illustrated in Figure 42.2. The basic functionality of the SoC under consideration is discussed below.

Electronic Controller Electro Hydraulic system

Actuator systems are vital in the flight control system, providing the motive force necessary to move the flight control surfaces. Hydraulic actuators are very common in space vehicle and flight control systems, where force/ weight consideration is very much important. This system positions the control surface of aircraft meeting performance requirement which acting against external loads. The actuator commands are processed in four identical analog servo loops, which command the four coils of force motor driving the hydraulic servo valve used to control the

motion of the dual tandem hydraulic jack. The motion of the spool of the hydraulic servo valve (Master control Valve), regulates the flow of oil to the tandem jacks, thereby determine the ram position. The Spool and ram positions are controlled by means of feedback loops. The actuator system is controlled by the on-board flight electronics. A lot of work has been done for On-line fault detection and diagnosis of the mechanical system, however OLT of the electronic systems were hardly looked into. It is to be noted that as Electro Hydraulic Actuators are mainly used in mission critical systems like avionics; for reliable operation on-line fault detection and diagnosis is required for both the mechanical and the electronic sub-systems.

The IEEE 1149.1 and 1149.4 circuitry are utilized to perform the BIST of the interconnecting buses in between the cores. It may be noted that on-line tests are carried only for cores, which are more susceptible to failures. However, the interconnecting buses are tested during startup and at intervals when cores being connected by them are ideal. The test scheduling logic can be designed as suggested in [10].

The following three classes of tests are carried in the SoC:

1. Interconnect test of the interconnecting buses (BIST)

Interconnect testing is to detect open circuits in the interconnect betweens the cores, and to detect and diagnose bridging faults anywhere in the Interconnect --regardless of whether they are normally carry digital or analog signals. This test is performed by EXTEST instruction and digital test patterns are generated from the pre-programmed test controller.

2. Parametric test of the interconnecting buses (BIST)

Parametric test: Parametric test permits analog measurements using analog stimulus and responses. This test is also performed by EXTEST instruction. For this only three values of analog voltages viz., V_H=VDD, V_{Low}=VDD/3, V_G= VSS are given as test inputs by the controller and the voltages at the output of the line under test is sampled after one bit coarse digitization as mentioned in the IEEE 1149.4 standard

3. Internal test of the cores (Concurrent tests)

This test is performed by INTEST instruction and this enables the on-line monitors placed on each of the cores present in the SoC. This test can be enabled concurrently with the SoC operation and need not be synchronized to start up of the normal operation of the SoC. The asynchronous startup/shutdown of the on-line testers facilitates power saving and higher reliability of the test circuitry if compared to the functional circuit.

7. References

1) M.R. Lyu, ed., *Software Fault Tolerance*, John Wiley & Sons, New York, 1995.
2) K.K. Saluja, R. Sharma, and C.R. Kime, “A Concurrent Testing Technique for Digital Circuits,” *IEEE Trans. Computer-Aided Design*, Vol. 7, No. 12, Dec. 1988, pp. 1250-1259.
3) M. Nicolaidis, “Theory of Transparent BIST for RAMs,” *IEEE Trans. Computers*, Vol. 45, No. 10, Oct. 1996, pp. 1141-1156.

4) A. Mahmood and E. McCluskey, "Concurrent Error Detection Using Watchdog Processors—A Survey," *IEEE Trans. Computers*, Vol. 37, No. 2, Feb. 1988, pp. 160-174.
5) B.W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.
6) B.T. Murray and J.P. Hayes, "Testing ICs: Getting to the Core of the Problem," *Computer*, Vol. 29, No. 11, Nov. 1996, pp. 32-45.
7) H. Al-Asaad, J.P. Hayes, and B.T. Murray, "Scalable Test Generators for High-Speed Data Path Circuits," *J. Electronic Testing: Theory and Applications*, Vol. 12, No. 1/2, Feb./Apr. 1998, pp. 111-125 (reprinted in *On-Line Testing for VLSI*, M. Nicolaidis, Y. Zorian, and D.K. Pradhan, eds., Kluwer, Boston, 1998).
8) "A Formal Approach to On-Line Monitoring of Digital VLSI Circuits: Theory, Design and Implementation", Biswas, S Mukhopadhyay, A Patra, Journal of Electronic Testing: Theory and Applications, Vol. 20, October 2005, pp-503-537.
9) S. Biswas, B Chatterjee, S Mukhopadhyay, A Patra, "A Novel Method for On-Line Testing of Mixed Signal "System On a Chip": A Case study of Base Band Controller, 29th National System Conference, IIT Mumbai, INDIA 2005, pp 2.1-2.23.
10) "An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller", A.T. Dahbura, M.U. Uyar, Chi. W. Yau, International Test Conference, USA, 1998, pp 55-62.

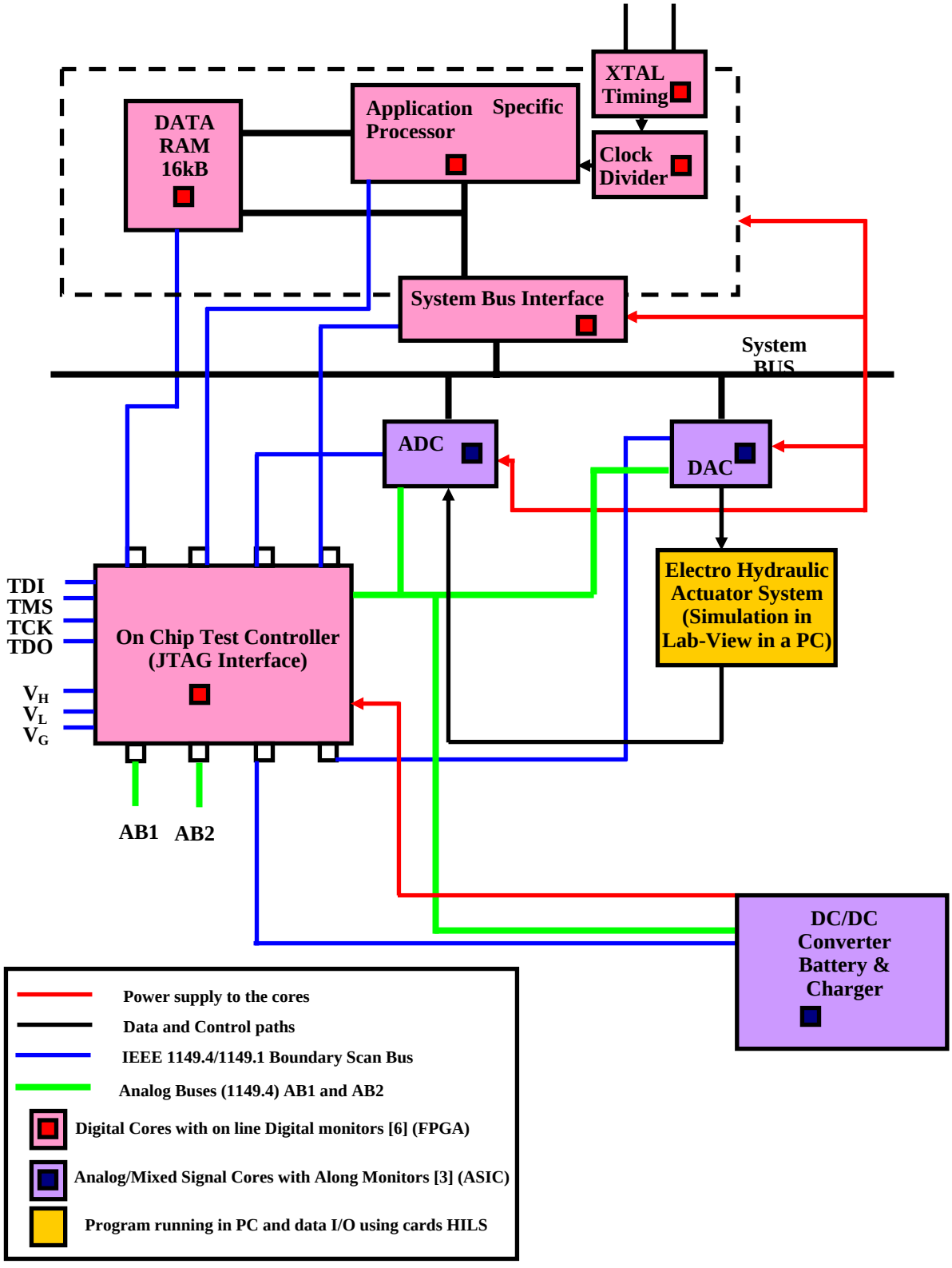

Fig. 42.2 Block Diagram of the SOC Representing On-Line Test Capability