Unified Test Generation and Application Flow for Automotive SoCs

David Sargsyan
Yerevan State University
Yerevan, Armenia
e-mail: davvad93@gmail.com

Gurgen Harutyunyan
Synopsys
Yerevan, Armenia
e-mail: gurgen.harutyunyan@synopsys.com

ABSTRACT
This paper presents an end-to-end test solution for an automotive system on chips (SoCs). The presented multi-purpose solution provides the possibility to select the appropriate test mechanisms at design stage, detect manufacturing faults during the production, check the safety mechanisms during power-up, as well as run periodic test during mission mode. A software automation tool and its application to a safety critical SoC are demonstrated to show the effectiveness and completeness of the proposed flow.

Keywords
Automotive, ISO 26262, ASIL, built-in self-test, periodic test.

1. INTRODUCTION
Nowadays the automotive market makes great strides in semiconductors industry. After increasing 11.5% in 2014, the automotive semiconductors market declined 2.5% in 2015, but then in 2016 it returned with 10.6%. There are several reasons driving the growth of this market (see Figure 1): the rising trend of vehicle electrification, rapid technological progress, the increasing vehicle production, etc. In order to meet automotive market requirements, automakers continually integrate Electronic Control Units (ECU) into vehicles to ensure better driving experience and safety. The fastest growing segments in automotive semiconductor market are Advanced Driver Assistance Systems (ADAS) and Electric vehicles. Some of the examples of such systems are Adaptive Cruise Control (ACC), Road Sign Recognition (RSR), Intelligent Speed Adaptation (ISA), Driver Monitoring System (DMS) and so forth. These electrical/electronic systems play a crucial role during autonomous driving, and since the health of people is put at stack any risk of failure needs to be minimized as much as possible. Thus, they have very high safety and reliability requirements.

One of the most important requirements in automotive is to have reliable test and repair solution for a product, not only at manufacturing but also in the field. This is related mainly to embedded memories in SoC, spanning most of the SoC area and being the major contributors to high yield. The most preferred approach for testing and repairing embedded memories is built-in self-test (BIST) solution. Before automotive requirements came to the scene, traditional BIST solutions were mainly focused on SoC production stage. Now when automotive has occupied a large sector in semiconductors market, mission mode safety has become one of the highest priority requirements for memory BIST solutions. Therefore, having a high-quality product after production stage is not enough, it is equally important to have an efficient test solution also in the field.

Different BIST approaches for mission mode memory testing have already been proposed in the literature. One of such techniques is transparent BIST described in [1]. The main goal of transparent BIST is to maintain memory content after testing. It transforms the existing BIST test algorithms into transparent ones, based on a set of predefined rules, meanwhile maintaining the coverage of the test algorithm. Several new features for in-field testing are presented in [2], for instance, the concept of non-destructive and destructive self-tests is presented.

There are several works, which show the benefits of structural testing for in-field test and fault diagnoses. For instance, [3] shows the advantages of reusing production mode test methods for in-system test. The disadvantage of this method is non-availability of test access port in the field.

The goal of this paper is to combine all the above-mentioned solutions and present an end-to-end test solution and generation flow for memories in automotive SoC.

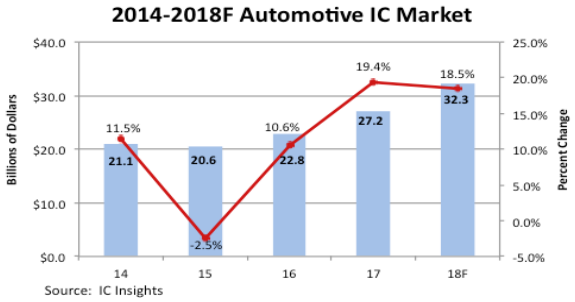

Figure 1. Automotive IC Market growth diagram [4]

In the next section of this paper ISO 26262 standard and its requirements for memory BIST are presented. In Section 3, a unified test generation flow for embedded memory testing is described. Section 4 presents the implementation details and, finally, Section 5 concludes the paper.

2. ISO 26262 AND ITS REQUIREMENTS
Considering the discussed high demands of safety and reliability in automotive, the detection of potential risks of software and hardware gets high priority. This motivated the emergence of ISO 26262 standard [5]. ISO 26262 is an adaptation of IEC 61508 functional safety of electrical/electronic/programmable electronic safety-related systems for specific automotive requirements. It defines the requirements for achieving an acceptable level of risk for electrical and/or electronic systems in automotive. The qualification of the final product is done by automotive safety integrity levels (ASIL) A-D. In ISO 26262, ASIL classifications are used to express the level of risk reduction to prevent hazards. ASIL D is the highest level and ASIL A is the lowest. The ASIL level calculated for the given hazard is then assigned to the safety goal. ASIL is determined based on a combination of the probability of exposure, the possible controllability by a driver, and the possible outcome severity if a critical event occurs. In order to meet the safety and reliability requirements of ISO 26262 standard, it is necessary to have 3 stages of testing for automotive SoCs: manufacturing test, power-on self-test (POST) and mission mode test.

Manufacturing test

In production mode the main requirement of automotive is to have a high quality product, which is measured by high yield and low DPPM (Defective Parts Per Million) for the embedded memory. Considering this, the main goal of the testing during manufacturing process is to have an efficient and comprehensive set of test and repair algorithms. There are different test algorithms developed for memory testing [6]-[8]. The main difference between them is the complexity, thus they have different runtime requirements. Depending on the criticality and specifics of the final product, different algorithms may be utilized.

Power-on self-test

ISO 26262 standard has a strong requirement to test embedded memories not only at production, but also during functional mode. The reason of such requirement is to avoid occurrence of any hazardous failure in functional mode, which can originate, for instance, due to aging or electromigration effect. Therefore, the system should be tested during each power-up. This stage is called power-on self-test (POST). The main goal of POST is to check whether all the components in automotive SoC are functioning correctly or not. While in production mode there are options to select preferable test algorithms, and run test algorithms with high complexity, during POST the test algorithms are not programmable, they are hardcoded and cannot be modified afterwards. Besides, it is preferable to have test algorithms with the low complexity, due to time constraints before entering mission mode.

Based on ISO 26262 standard, there is also another requirement to "check the checkers". In other words, there is a need to check if fault detection mechanisms (also called safety mechanisms) are functioning correctly since due to aging, electromigration or other reasons, those mechanisms can become erroneous and behave incorrectly.

Mission mode test

After POST, the third stage of testing is to periodically test the system in mission mode. Periodic test checks whether the device has become unsafe after the last POST. The main goal of periodic test is to test the system periodically with the small bursts and warn the driver in case of any potential issue.

The other type of testing used in the mission mode is the error correction code (ECC) [9]. The techniques discussed above, target the testing of hard faults, nevertheless, in order to adhere safety and reliability requirements there is a need to handle also the soft errors occurring in the mission mode. The basic idea of ECC is based on the concept of adding check bits in the memory associated with each memory word, which are determined by even parity checks and reporting an error in case the parity check is failing.

3. TEST GENERATION FLOW

This section presents a unified test generation flow for automotive SoCs. Figure 2 shows a typical SoC hierarchical test structure. Hierarchical test system helps to test large SoCs within the desired schedule and cost. It makes SoC testing more flexible, by allowing to schedule the test of different memory and IP groups, running them in parallel or serial, thus optimizing the test time.

3.1. Manufacturing Test Generation Flow

As it was already stated, the main goal of manufacturing test is to have efficient test and repair algorithms. There are two main classes of memory faults:

- Static faults – this type of faults can be sensitized by single memory operation. Examples of static faults are stuck-at fault, coupling fault, read disturb fault, etc.
- Dynamic faults – this type of faults requires more than one memory operation to be sensitized, thus the number of dynamic faults is theoretically unlimited. Examples of dynamic faults are dynamic read disturb fault, dynamic incorrect read fault, dynamic deceptive read disturb fault, etc.

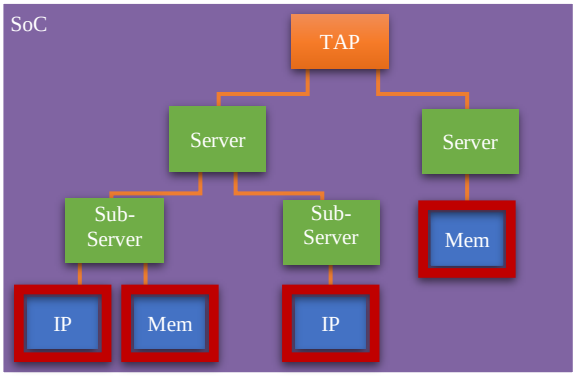

Figure 2. Hierarchical test architecture for SoC

Single-cell fault primitives (FPs) are described by <S/F/R> and two-cell (coupling) FPs are described by <Sa; Sv /F/R> [7]. In notations of FPs, S, Sa and Sv are the sequences of operations required for fault sensitization (S is applied to the faulty cell, Sa - to the aggressor cell and Sv - to the victim cell), $F \in \{0, 1\}$ is the observed memory behavior that deviates from the expected one. $R \in \{0, 1, -\}$ is the result of a read operation applied to the faulty cell, in case if the last operation of S is a Read operation. " - " is used when the last operation of S is not a Read operation. For example, if a cell has the fault <0W0/1/->, it means that if it contains value 0, then applying operation W0 on it will flip the cell value from 0 to 1. Or if two cells contain the fault <1; 0W1R1/0/1>, it means the following: if the aggressor cell has value 1, the victim cell has value 0 then applying two sequential operations {W1, R1} will fail. Though the read operation will return the correct value 1, the victim cell value will remain 0.

Using the above notation, Tables 1-4 present unlinked static and two-operation dynamic faults [8]. The symbol "~" used in the tables denotes logical negation, and $x, y, z, t \in \{0, 1\}$.

In addition to the described faults, advanced technology nodes have brought their own types of faults like FinFET-specific faults [10]. Since automotive SoCs are mainly using the advanced nodes, then the testing of memories should cover also those faults.

Table 1. Single-cell static faults

Functional fault models	Fault primitives
State Fault (SF)	<x/~x/->
Transition Fault (TF)	<xW(~x)/x/->
Write Destructive Fault (WDF)	<xWx/~x/->
Read Destructive Fault (RDF)	<Rx/~x/~x>
Deceptive Read Destructive Fault (DRDF)	<Rx/~x/x>
Incorrect Read Fault (IRF)	<Rx/x/~x>

Table 2. Two-cell static faults

Functional fault models	Fault primitives
State Coupling Fault (CFst)	<x; y/~y/->
Transition Coupling Fault (CFtr)	<x; yW(~y)/y/->
Write Destructive Coupling Fault (CFwd)	<x; yWy/~y/->
Read Destructive Coupling Fault (CFrd)	<x; Ry/~y/~y>
Deceptive Read Destructive Coupling Fault (CFdrd)	<x; Ry/~y/y>
Incorrect Read Coupling Fault (CFir)	<x; Ry/y/~y>
Disturb Coupling Fault (CFds)	<Rx; y/~y/->, <xWy; z/~z/->

Table 3. Two-operation single-cell dynamic faults

Functional fault models	Fault primitives
dynamic Read Destructive Fault (dRDF)	<xWyRy/~y/~y>, <xRxRx/~x/~x>
dynamic Deceptive Read Destructive Fault (dDRDF)	<xWyRy/~y/y>, <xRxRx/~x/x>
dynamic Incorrect Read Fault (dIRF)	<xWyRy/y/~y>, <xRxRx/x/~x>
dynamic Transition Fault (dTF)	<xWyW(~y)/y/->, <xRxW(~x)/x/->
dynamic Write Destructive Fault (dWDF)	<xWyWy/~y/->, <xRxWx/~x/->

Table 4. Two-operation two-cell dynamic faults

Functional fault models	Fault primitives
dynamic Read Destructive Coupling Fault (dCFrd)	<x; yWzRz/~z/~z>, <x; zRzRz/~z/~z>
dynamic Deceptive Read Destructive Coupling Fault (dCFdrd)	<x; yWzRz/~z/z>, <x; zRzRz/~z/z>
dynamic Incorrect Read Coupling Fault (dCFir)	<x; yWzRz/z/~z>, <x; zRzRz/z/~z>
dynamic Transition Coupling Fault (dCFtr)	<x; yWzW(~z)/z/->, <x; zRzW(~z)/z/->
dynamic Write Destructive Coupling Fault (dCFwd)	<x; yWzWz/~z/->, <x; zRzWz/~z/->
dynamic Disturb Coupling Fault (dCFds)	<xWyWt; z/~z/->, <xWyRy; z/~z/->, <xRxWy; z/~z/->, <xRxRx; z/~z/->

In addition, there is another class of faults, which mainly appear during the system operating mode (in the field). Those are called aging faults. Long-term performance degradations may activate physical defects in the system, due to transistors aging. The main aging effects cause NBTI (Negative Bias Temperature Instability) and PBTI (Positive Bias Temperature Instability) [11]. These effects shift the threshold voltage of a transistor, which causes bit-cell stuck-at, bit-cell transition, coupling, delay faults and sense amplifier failures. Different types of memory tests were proposed, but the most efficient way of testing memories is the class of March tests [6]. It consists of a sequence of March elements, where each of them consists of March operations applied to a memory cell (write 0, write 1, read 0, read 1) with a fixed addressing order. Most of the currently used test algorithms are based on March tests.

Test generation flow for manufacturing test

The test generation flow for manufacturing test consists of the following steps:

1. Create test sessions, i.e., which group of memories will be tested (in parallel) in the first session, which group in the second session, and so forth;
2. Select the test algorithm dedicated for production mode;
3. If a fault is detected, then repair analysis is done and if the fault is repairable, then repair signature is calculated and stored in non-volatile memory (e.g., Fuse). Further on, during POST, this information will be read from non-volatile memory and the corresponding memory will be repaired before running BIST in POST mode.

3.2. In-Field Test Generation Flow

Memory BIST approach for in-field testing has some architectural differences from the production mode BIST. Typical memory BIST architecture being used in production mode is shown in Figure 3. It consists of the following modules:

- Test access port (TAP) – JTAG port [12], which is used to apply test vectors to the device under test;
- Controller;
- Test algorithm register (TAR) – contains memory test algorithms;
- Background pattern generator (BPG);
- Address generator (AG) – selects the memory range to be tested;
- Comparator – checks whether the test has passed or failed.

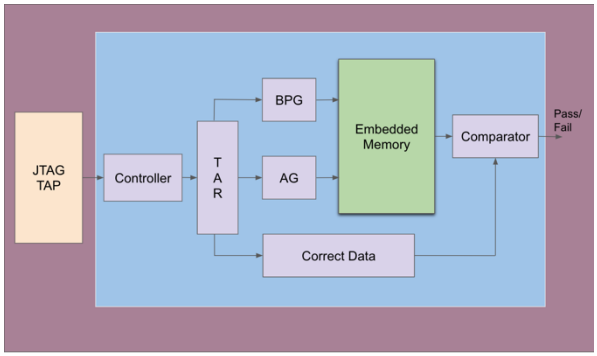

Figure 3. Memory BIST architecture in production mode

This architecture is not applicable for the memory testing in mission mode for the following reasons:

- There is no JTAG test access port in mission mode;
- The content of the memory must be maintained after the test;
- Test time constraints require to test memory with the small bursts.

To solve this problem, BIST architecture is modified with the following components (Figure 4) [13]:

- New test access port is added for system test;
- Reserve register is added to maintain content of the memory under the test;
- Address generator is modified to enable burst mode.

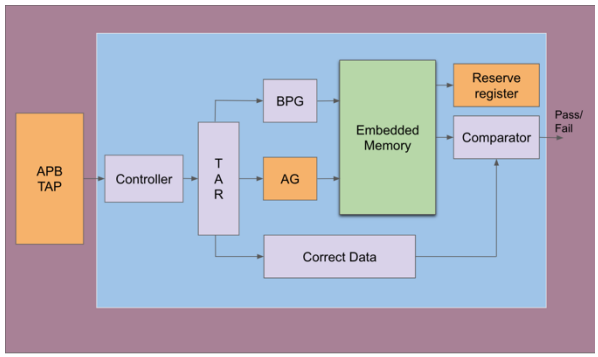

Figure 4. Memory BIST architecture for automotive

Considering all the above mentioned, the in-field test generation flow will be as follows:

Test generation flow for POST

1. Check the checkers, i.e., the safety mechanisms, such BIST fault detection capability, ECC error detection and correction capability, etc.;
2. Load repair information from non-volatile memory collected during production mode;
3. Select the test algorithms dedicated for POST mode;
4. Run the BIST.

Test generation flow for mission mode

1. Enable ECC for detecting and correcting transient faults;
2. Identify idle memories (not used in mission mode);
3. Select the test algorithms dedicated for mission mode;
4. Periodically run transparent test for the selected memories.

4. IMPLEMENTATION OF THE FLOW

The proposed methodology is implemented in a software automation tool called Yield Accelerator (YA) of Synopsys DesignWare STAR Memory System [14]. It takes the following inputs:

1. SoC design information (e.g., the architecture described in Figure 4).
2. User information on test algorithms, test mode and test conditions.

Design planning stage

Every SoC project needs to have a planning and designing stage, where the certain components are developed and combined together to form an SoC [15]. One of these components is Design-for-Testability (DFT) infrastructure. DFT design flow for automotive projects consists of the following steps:

1. Define memory instances, which should be used in a project;
2. Configure ECC controllers for memories with soft error protection requirements;
3. Define memory groups;
4. Set up BIST controller per memory group;
5. Specify test algorithms for production test, POST and mission mode periodic test, since the set of test algorithms are different for each stage;
6. Define test access interface for each testing stage.

After design finalization it should be integrated into real SoC design and verified by testbench simulation. If the simulation passes, the chip can be sent to production.

For each test mode YA has certain capabilities to enable testing of a given automotive SoC.

Manufacturing stage

At manufacturing stage, the chip testing is being done by ATE (automatic test equipment), which runs the test patterns from top level JTAG interface. For this stage, YA generates test pattern in STIL (Standard Test Interface Language) file format [16]. STIL provides an interface between digital test generation tool and test equipment to transfer the large volumes of digital test vector data to ATE environments.

POST and mission mode periodic testing stage

During POST and mission mode JTAG is usually disabled (mainly due to security reasons). Hence, other interfaces are used for performing POST and mission mode periodic test. In automotive SoCs there is a safety manager, such as Synopsys ARC [17], which takes care of POST and mission mode periodic test.

One of the test interfaces widely used between safety manager and test mechanisms of SoC is APB interface [18]. While in order to run a given test pattern in safety manager's environment a firmware in C code is required. YA has a capability:

1. To generate a given test pattern working through APB interface;
2. To generate a firmware in C code for a given test pattern to be loaded into safety manager for the execution during POST or mission mode periodic test [19].

5. CONCLUSION

This paper presents an end-to-end test methodology and unified test architecture for automotive SoCs. The ISO 26262 functional safety standard and its requirements for testing automotive SoCs are discussed. Three main stages of test are described, and for each stage the corresponding test solution is proposed. Synopsys Yield Accelerator software automation tool is described where the proposed test methodology is implemented.

REFERENCES

[1] M. Nicolaidis, "Theory of Transparent BIST for RAMs", *IEEE Transactions on Computers*, vol. 45, no. 10, 1996, pp. 1141-1156.

[2] A. Dutta, S. Alampally, A. Kumar, R. A. Parekhji, "A BIST Implementation Framework for Supporting Field Testability and Configurability in an Automotive SOC", *Workshop on Dependable and Secure Nanocomputing*, 2007.

[3] A. Cook D. Ull M. Elm H. Wunderlich, H. Randoll, S. Dohren, "Reuse of Structural Volume Test Methods for In-System Testing of Automotive ASICs", *IEEE Asian Test Symposium (ATS)*, 2012, pp. 214-219.

[4] http://www.icinsights.com/

[5] https://www.iso.org/standard/43464.html

[6] A.J. van de Goor, "Testing semiconductor memories: Theory and Practice", *John Wiley & Sons*, Chichester, England, 1991.

[7] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, "Testing Static and Dynamic Faults in Random Access Memories", *IEEE VLSI Test Symposium*, 2002, pp. 395-400.

[8] G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, "A New Method for March Test Algorithm Generation and Its Application for Fault Detection in RAMs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Volume 31, Number 6, June 2012, pp. 941-949.

[9] R. W. Hamming, "Error Detecting and Error Correcting Codes", *Bell System Technical Journal*, Vol. 29, No. 2, pp. 147-160, 1950.

[10] G. Harutyunyan, G. Tshagharyan, V. Vardanian, Y. Zorian, "Fault Modeling and Test Algorithm Creation Strategy for FinFET-Based Memories", *IEEE VLSI Test Symposium (VTS)*, 2014, pp. 49-54.

[11] Z. Qi, J. Wang, A. Cabe, S. Wooters, T. Blalock, B. Calhoun, M. Stan, "SRAM-Based NBTI/PBTI Sensor System Design", *Design Automation Conference*, 2010, pp. 849-852.

[12] IEEE Std. 1149.1, IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2001.

[13] D. Sargsyan D, "ISO 26262 Compliant Memory BIST Architecture", *International Conference on Computer Science and Information Technologies (CSIT)*, 2017, pp. 164-167.

[14] K. Darbinyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, "A Robust Solution for Embedded Memory Test and Repair", *IEEE Asian Test Symposium*, 2011, pp. 461-462.

[15] G. Tshagharyan, G. Harutyunyan, Y. Zorian, "An Effective Functional Safety Solution for Automotive Systems-on-Chip", *IEEE International Test Conference (ITC)*, 2017, Paper ET 2.2, pp. 1-10.

[16] 1450-1999 - IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data.

[17] https://www.synopsys.com/dw/ipdir.php?ds=sw_metaware

[18] http://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf

[19] D. Sargsyan, "Firmware Generation Architecture for Memory BIST", *IEEE East-West Design & Test Symposium (EWDTS)*, 2018.