Design for Test Techniques to Enable Automotive Functional Safety

Swathi G, Prasanth V, Maheedhar Jalasutram
Texas Instruments.
swathig@ti.com, prasanth.v@ti.com, maheedhar@ti.com

Abstract

Periodic testing of electronic circuits in safety critical systems is becoming increasingly important due to the increasing safety and reliability requirements. Functional safety standards recommend targeted coverage goals to quantify the safety level for a device. Achieving these coverage goals using the traditional software based techniques is not practical. Adding redundancy to enable safety will result in significant hardware overhead. In this context, providing low cost (lesser area overhead, lesser impact on application MIPS, etc.) on-chip solutions to address field-test requirements is becoming increasingly important. These tests should be able to run in minimal time in idle slots of the application without affecting the application throughput. We should also ensure that there is no significant area overhead incurred in supporting these solutions on-chip. In this paper, we discuss various Design for Test techniques to enable application time self-test of various critical components of an SOC.

Keywords: Self-test, Functional Safety, Design for Test

I. Introduction

Semiconductors usage in different applications is increasing rapidly. More importantly they are being used in many safety critical applications like automobile, industrial automation, health care devices etc. While at one end, semiconductors offer newer functionality and ease of use, they also pose an increased risk of failures. Consider the example of an automobile. Semiconductor devices are used to control multiple functionalities like braking, steering control, airbags, etc. A failure in one of the semiconductor component used to implement these critical functionalities can be fatal. To ensure the correctness of the application, it is required to periodically check the functionality of semiconductors used in safety critical applications.

Many functional safety standards like ISO26262 [1] and IEC61508 [2] have come up with guidelines to ensure that the systems are designed to meet the required safety level. Integrated Circuits (IC) play an important role in ensuring safety of the application. The device and development costs of “adding” safety on chip can be pretty high, calling for new solutions to be developed. While software based techniques can be used, it involves significant effort to develop these tests and quantify the coverage as required by the safety standards. Simple on-chip solutions when planned ahead in the design cycle can save significant effort / cost in enabling safety.

While it may call for adding new logic on-chip to enable field-testing of the device, it is important to ensure that the area and cost overhead due to the additional logic is minimal. To enable this, we should reuse the logic deployed for manufacturing test to enable field self-test. In this paper, we propose methods to reuse the on-chip Design for Test modules developed for manufacturing test of the device to be used for field-test. The challenges associated with supporting field-test and the design techniques to overcome them are also discussed in this paper.

This paper is organized into seven sections. Section 2 provides the background on need for tests in application and prior approaches used to support these tests. Section 3 discusses the safety concepts to which a device should adhere and the metrics to quantify the safety level of a device. Section 4 discusses the challenges and techniques to test digital logic at power-up and run-time. Sections 5 and 6 discuss the design techniques to test memories and analog in application. Section 7 concludes the paper.

II. Background and Related Work

Integrated circuits used in safety critical applications can fail due to various reasons like ionizing radiation, ageing, excess temperature or voltage conditions etc. We use a representative example (described in [3]) to illustrate the impact of IC failure on the application.

A. Application case study

Consider the example control IC (Digital Controller) used in traction control system of an electric vehicle. The control IC receives torque/speed set-point command from supervisor IC based on information from input functions like accelerator, brake pedal, etc. The control IC senses torque by measuring the motor phase current. The processing element in the control IC will determine the system error by comparing the set-point torque received from the supervisor IC with the sensed torque and execute control algorithm to make this error zero. The response will be conveyed to the motor as change of applied power, which is done by controlling the power-stage using pulse width modulation techniques as indicated in Figure 1. The closed loop operation happens periodically based on some system events. Once the set of operation completes, the Control IC will wait (called as the idle slot) for a system event to process the next set of information.

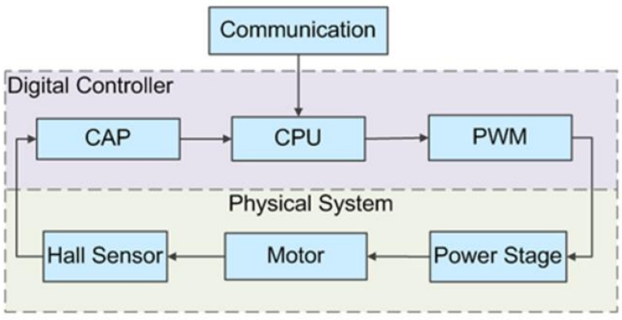


Figure 1: Closed loop control system

Impact of a fault in the control IC at the application level can be understood by examining the impact of a fault in each of the modules in that chip.

1) A fault in the sensor module (CAP) [4] can lead to incorrect sensing (higher/lower than actual) of speed. Incorrectly sensed speed input to the control algorithm can result in incorrect processing which can in turn lead to inadvertent acceleration or deceleration
2) A fault in the processing element (CPU) will cause incorrect processing during control algorithm execution causing inadvertent acceleration or deceleration.
3) A fault in the actuation module (PWM) [5] output can cause the output to stick to either zero or one, again resulting in inadvertent acceleration or deceleration.

As illustrated in the above case study, it is important to ensure correctness of each of the components of an IC, to get a desired functionality from the IC used in a safety critical application.

B. *Prior Work*

A variety of techniques are available to check for the correctness of a device. A commonly used technique is to test using software [6]. A set of software tests is created, to check for critical functionality of the device. These software tests are run periodically (at power-up and in idle slots) to screen for any defects in the device. However, these tests are application based and hence every new application requires a new test. This involves significant effort in generating optimized tests to reduce the test time and memory requirements. Quantifying the coverage attained with software tests through functional fault simulation is a very laborious process.

Design for Test (DFT) based structural tests are proposed for the digital logic. These tests overcome the limitations of software based tests as they are independent of the functionality. The effort involved in generating these tests is minimal and quantification of tests coverage can be done easily. Coverage and test time with software and DFT based structural tests for an IP with 3000 flip-flops is shown in TABLE **1**.

TABLE 1. SOFTWARE TEST VS STRUCTURAL TEST

Self-test type	Details	Coverage	Test cycles	Test time (us)
Software (S/W)	Software test suite	43.87%	21625	216
DFT based structural tests	Detecting all faults detected by S/W self-test	43.87%	10010	100
	Full Hardware test suite	97.77%	21625	216

Alternatively hardware redundancy [7] is used for some modules. In this method, result from redundant module is compared with result from the actual module to determine the correctness. However, this comes with an area overhead and it is practically not possible to have redundancy for all modules in a design.

The techniques to choose the right DFT solution and the tradeoffs associated are discussed in following sections. For modules like analog and memories, where DFT based structural tests are not possible, alternative tests are proposed.

III. FOUNDATIONAL SAFETY GUIDELINES AND CONCEPTS

Various functional safety standards have been established over the years to address hazards caused by malfunctioning behavior of electronic devices in safety applications. Of these, ISO 26262 functional safety standard addresses the requirements from automotive standpoint. The standard defines requirements at various stages of system life cycle including development. Various techniques should be incorporated in design stage to enable an SOC to comply with the requirements specified in ISO26262 standard.

As discussed in flowing paragraphs (taken from [8]) causes of application failure are classified into (i) Systematic failures and (ii) Random failures. Detailed classification is shown in Figure 2.

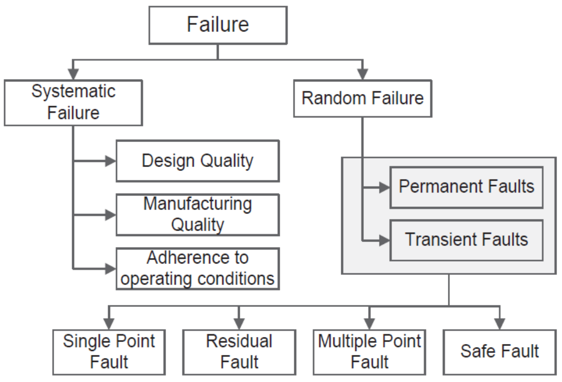


Figure 2. Failure classification as per ISO26262

Systematic failures typically arise due to design quality issues (e.g. bugs in the design), manufacturing issues (e.g. faults escaping due to insufficient manufacturing test), non-adherence to operating conditions (e.g. device operating in conditions outside the specified temperature, voltage, humidity conditions), etc. These failures are repeatable in nature and can be controlled to a large extent by following a regimented approach (process) during the product life cycle.

Random failures occurring in a device can be due to permanent faults or transient faults. Permanent faults are caused by ageing phenomena like Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) [10]. Transient faults are caused due to alpha particle and neutron strikes. A permanent fault, as the name indicates, causes permanent damage to the chip and will force the chip to be replaced in most of the cases. Random faults can be further classified into single point fault, residual fault, multiple point fault and safe fault [9]. The ISO26262 definitions for these faults are as follows: A **single point fault** is a fault which is not covered by any safety mechanism and whose failure can lead to violation of the system safety goal. **Residual faults** refer to a subset of faults that can lead to violation of a safety goal, where this subset of faults is not covered by the existing safety mechanisms. A **multi-point fault** is a fault which standalone cannot cause the violation of safety goal, but the same fault in combination with other independent faults can lead to violation of safety goal. A fault whose occurrence will not significantly increase the probability of violation of a safety goal is called a **safe fault**.

Quantitative analysis for a circuit failure mode provides an objective means to ascertain the safety worthiness of a given circuit. These metrics can be used to (i) objectively assess safety effectiveness of the design to cope with the random hardware failures, (ii) guidance towards making enhancements for safety in the design based on afore-identified effectiveness. These are measured in terms of

Single Point Fault Metric (SPFM) and Latent Fault Metric (LFM).

$$\text{SPFM} = 1 - \frac{Residual\ Faults + Single\ Point\ Faults}{Total\ Safety\ Related\ Faults}$$

$$\text{LFM} = 1 - \frac{Undetected\ Multiple\ Point\ Faults}{Safety\ Related\ Faults - Residual\ Faults}$$

In this paper, we focus on techniques to provide random fault coverage.

IV. Digital Logic Test

Digital logic on an SOC includes the CPUs, digital peripherals, etc. Scan based DFT techniques are used for manufacturing test of this logic. The same on-chip infrastructure can be utilized to screen for defects in field as well.

A. *Power-on self-test*

At power-up, pre-operational checks need to be performed to ensure that CPU can boot-up correctly and perform the required operations. Full chip logic self-test can be used for addressing the latent fault coverage requirements of ISO26262 applications with key on-off hours <24 hours. Start-up self-test directly impacts the boot time of the application and we had a requirement from the end application to reach 90% coverage within 2ms. In addition, we wanted to reduce the area to be minimum so as to reduce the overall cost incurred due to the safety solution.

We used LBIST compression solution from Cadence [10] for power-on self-test to help in reducing the on-chip memory required to store the patterns.

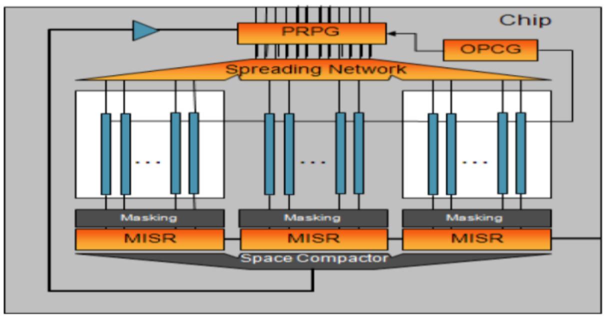


Figure 3. LBIST Architecture

The architecture for LBIST is shown in Figure 3. A golden seed is loaded into pseudo-random pattern generator (PRPG), which generates input patterns applied to scan channels. A multiple-input signature register (MISR) is used to capture the response from scan channels. The coverage vs patterns count for a regular LBIST is as shown in Figure **4Error! Reference source not found.**. As can be seen from the graph, the test coverage with a given seed tapers-off at a certain point, leading to huge pattern count for a minimal increment in coverage. Higher pattern count translates to higher test time at power-up, thereby impacting the boot time. Also, 90% coverage cannot be achieved with single golden seed loaded in PRPG.

To achieve the coverage goal of 90% with less numbers of patterns, experiments have been done by inserting test points on-chip. Test points increase controllability and observability for the faults that cannot be easily detected by ATPG tool, thereby improving the coverage. The number test points inserted is increased from 100 to 7000. With 100 test points 90% coverage could not be achieved. With 2500 test points, test coverage of 90% was achieved with 10240 patterns. As can be seen in Figure **5**, pattern count for desired coverage decreases with addition of test points. With 7000 test points, 90% coverage was attained with just 3200 patterns.

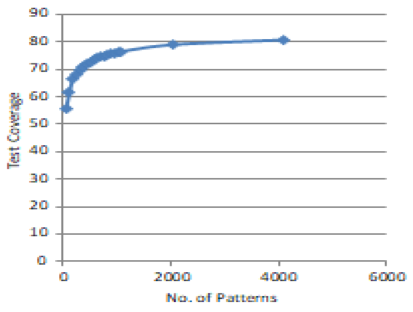


Figure 4. Coverage vs Pattern count with LBIST

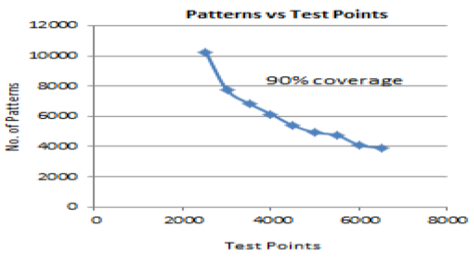


Figure 5. Pattern count decrease with test points

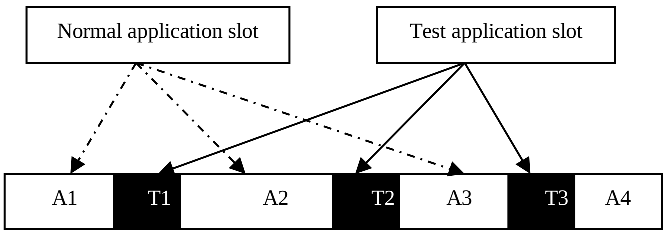
However, though the test time decreases, increased number of test points result in area overhead. So, trade-off has to be considered to arrive at the final solution. We achieved the test time target of <2 ms (10000 patterns running at 100MHz frequency with chain length of 185) and minimal area of 0.2mm^2 (combination of LBIST code area and test points) with 2500 test points.

B. *Run-time Build-In Self-test for CPU*

CPU is a critical component of SOC and it needs to be checked at regular intervals during the application. We used Run-time Build-In Self-Test (RT-BIST)[11] capability to perform testing of the CPU logic during application. This will help address the SPFM requirements of ISO26262. Various challenges to support run-time test of a CPU and design techniques to overcome them are discussed below.

1. Easier integration with application environment

RT-BIST run is triggered in the idle slots of the CPU. The application and test slots are interleaved as shown in Figure 6. To make use of the smaller idle slots of CPU, the



overall test should be split into multiple small intervals.

Figure 6. Interleaving Application and Test slots

The compression architecture should be chosen such that the complete test for full coverage can be split into multiple smaller test intervals. Each pattern should be a separate entity to provide interruptible capability after every pattern run. OPMSR compression architecture from Cadence [12] is chosen in which each input pattern is a separate entity. Hence the test slot duration can be as small as one pattern application time, which is twice the chain length. OPMISR compression architecture is shown in Figure **7**. The number of internal stumps is increased to reduce the chain length. We designed to support a chain length 20, so that test time for one pattern execution (shift-in, capture, shift-out along with other state-machine overheads) is 50 cycles. This ensures that even small idle periods in application environment can be used for triggering RT-BIST operation. Also, the intermediate states (context and result) of RT-BIST should be preserved between intervals so that the BIST operation can be resumed after it is interrupted. This is ensured by shifting in the last shift data of the previous test slot, when RT-BIST is triggered in next test slot. The result is stored in MISR (Multiple Input Signature Register) and is accumulated across tests. Design should take care that signature is not disturbed when we get back to application.

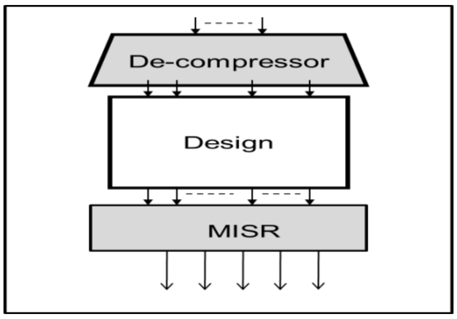


Figure 7. OPMISR Compression Architecture

2. Isolation of BIST logic

Any activity in the CPU during the RT-BIST run should not disturb the peripherals connected to it. During BIST, random test data is shifted into the scan chains, and CPU outputs change every cycle. The outputs can assume invalid state and may cause spurious device operation, if they propagate to the peripherals connected to the CPU. Similarly, the inputs to the CPU may vary during test causing invalid values to be captured and can cause test to fail. To avoid these, the CPU is completely isolated during the execution of RT-BIST.

3. Response time to events should be low

System performance should not be impacted because of test. Any external interrupts to the CPU during the RT-BIST operation should not be lost. In case of interrupts, RT-BIST operation should be interrupted as early as possible and interrupts occurring during RT-BIST run should be addressed with minimum interrupt service latency so that the system performance is not impacted. In our approach, the amount of time for which interrupts can wait is made programmable to the user. We have built a mechanism inside BIST controller to log any interrupts to the CPU during RT-BIST operation as shown in Figure 8. This ensures that no interrupts are lost during RT-BIST operation when the CPU is not able to service those interrupts.

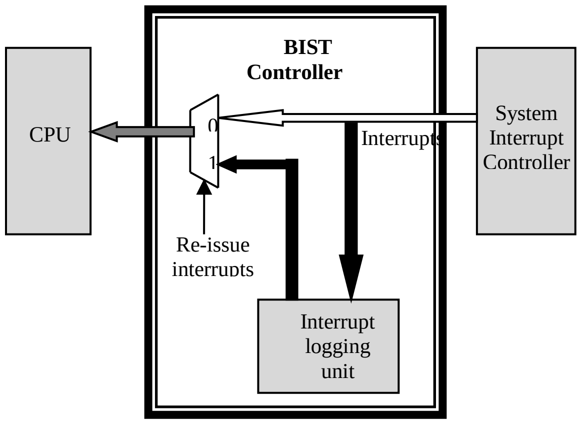


Figure 8. Interrupt logging unit

4. Achieving minimal on-chip memory and test time

The tests should give reasonable amount of confidence about the correctness of the system. This can be obtained by attaining high fault / defect coverage, e.g. 99%+ for stuck-at fault model. The pattern count inflates as the coverage reaches near 99%. Also, not all application will require the very high covered offered by RT-BIST. The BIST operation can be programmed to run so as to obtain a desired coverage. As shown in Table 2, there is a 3x pattern count difference between achieving 95% and 99% test coverages. Our architecture supports user configurable targeted coverage to reduce the total test time. At start-up if a complete check has to be done, 99% option can be targeted. On the other hand, if the test has to be run on few modules very frequently in less time, then 95% option can be chosen, where critical portions of design will be covered more frequently.

TABLE 2. COVERAGE VS TEST CYCLES

Sl. No	Structural coverage on total core faults	Test cycles
1	95%	12350
2	99.4%	37180

5. Quick response to failure

The failure information should be provided to the system in minimal time so that the error propagation to other parts of the system is contained. In typical BIST operations, the computed signature is compared against golden signature at the end of the test to certify that the device is functioning correctly. However, this is not recommended for field test and the failure has to be intimated as soon as possible. RT-BIST architecture supports comparison of the computed signature with the golden signature after every few patterns to reduce the error detection latency. The number of patterns after which a comparison is done is configurable. It must be remembered that targeting aggressive reduction in error detection latency directly impacts the pattern memory size.

6. Power during test

Shift-in and shift-out of the test data using the system clock at its full-speed can cause IR drop issues because of high toggling activity during test. To address this we have an option to use a divided clock to reduce the shift frequency while using the high frequency clock for capture, to ensure that the at-speed faults are detected.

V. MEMORY TEST

Memories (SRAM and ROM) should be tested in field to detect both permanent and transient faults. Test of memories at start-up can be used as an effective method for providing latent fault coverage requirements for the memories. Using a CPU to test all memories sequentially consumes significant amount of time and adds to start-up time. There are instances where memories inside peripherals are not directly accessible by CPU and hence cannot be tested in field if CPU based tests are used.

For manufacturing test of memories, Programmable memory BIST (PBIST) [13] is used. We have used the same infrastructure to enable testing of memories in field. PBIST controller consists of a small processing unit with an instruction set targeted specifically towards testing of memories. PBIST helps to have access to all the device SRAM and ROM instances. PBIST interface to memories is shown in Figure **9**.

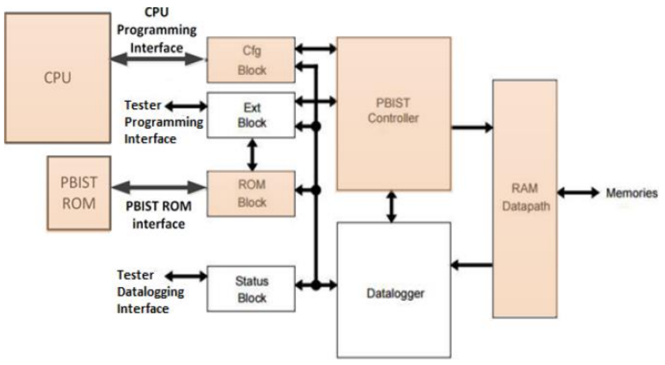


Figure 9. PBIST Interface to memories

A dedicated on-chip ROM is used for storing the instruction codes, which are then loaded into the PBIST controller. The memory BIST controller can be configured to do complete memory test through the ROM, hereby enabling memory test in the customer's application. Using PBIST controller, memories of the same type can be grouped together and tested in parallel. This allows for significant test time reduction for power up self-test. As shown in TABLE 3, by using PBIST to test memories in parallel, test time for testing all memories at start-up, reduced by 5x on a design with 16 memory instances.

TABLE 3.CYCLES FOR MEMORY TEST WITH PBIST AND CPU

Memory	Number of memories tested together	Address locations	Test cycles	
			PBIST	CPU TEST
Mem1	4	4096	53248	276889.6
Mem2	8	1024	13312	138444.8
Mem3	2	256	3328	8652.8
Mem4	2	512	6656	17305.6
Total cycles for testing all memories at start-up			76544	441292.8

However, enabling multiple memories in parallel can cause increased power consumption. Analysis has been done to determine the number of memories that can be enabled in parallel to contain the power during test within spec limits.

Interval based PBIST execution can also be used to meet the permanent fault coverage requirements of memories. Typically, ECC / Parity mechanism is used as the primary diagnostic for memories. However, ECC and Parity safety mechanisms do not provide coverage of the address decoder faults. PBIST based memory tests can be used to augment the coverage provided by primary diagnostic mechanisms.

FTTI is the time duration within which any fault in the chip need to be detected. 10ms is used as the typical FTTI value for many automotive applications like power steering, braking, etc. The tests for all the memories need to be completed within this duration. However, with just PBIST this may not be practical at times. Hence, we have implemented a background memory checking engine [14] which can perform the test without consuming bandwidth from PBIST and CPU. As shown in Figure 10, the background memory checking engine snoops for idle cycles and perform CRC check of Non Volatile Memory and static SRAM contents (e.g. look-up tables, code copied from Flash, etc.). This feature enables a zero overhead memory test capability with minimal hardware addition.

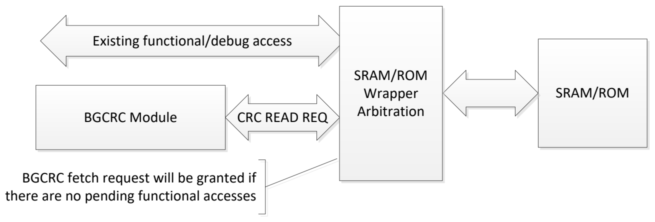

Existing functional/debug access
SRAM/ROM Wrapper Arbitration
SRAM/ROM
BGCRC Module
CRC READ REQ
BGCRC fetch request will be granted if there are no pending functional accesses

Figure 10. BRCRC implementation

VI. ANALOG TEST

Our SOCs typically consist of many on-chip analog IPs like Voltage regulators (VREG), Oscillators, ADCs, DACs, and Programmable Gain Amplifiers (PGA) etc. For a reliable SOC operation, it is critical to ensure that these analog IPs are functionally well.

During production test, oscillators and PLLs need to be trimmed to set their clock outputs to desired frequency range. To enable these tests, an on-chip Dual Clock Comparator (DCC) module is supported, which measures the frequency of a selectable clock source, using the input clock as a reference. Two independent counter blocks count clock pulses from each clock source. The clocks decrement counters in order to compare their relative frequencies. The same infrastructure can be utilized to determine the accuracy of a clock signal during the execution of an application. Accurate clock frequency is critical for correct functioning of the device and any change in frequency should be immediately detected. Using DCC, a 2% drift from the expected frequency in a clock as required by the functional safety requirements, can be detected within few micro-seconds of time.

In the presence of an on-chip ADC, voltage outputs from circuits like DACs and VREG can be measured on-chip by looping them back to the ADC as shown in Figure 11. This mechanism can be used during application to check voltage outputs from critical components like VREG and bandgaps. At power-up, voltage outputs from critical components like VREG, bandgaps are looped back to ADC and measured to ascertain that they are within the permissible limits. Software routines are developed to trigger these tests by the CPU at power-up. Loop back tests are used for providing SPFM permanent fault coverage and LFM coverage.

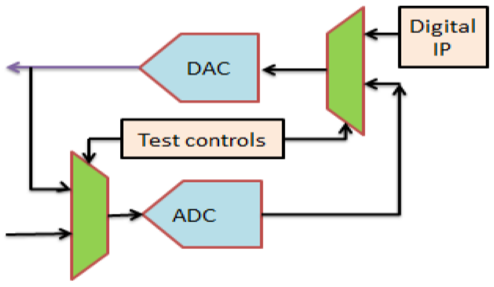


Figure 11. On-chip ADC and DAC loopback

VII. Conclusion

We have described a set of DFT techniques which can be used to enable functional safety for integrated circuits. We described how the test coverage and start-up time goal of 90% and 2ms respectively for start-up self-test was achieved. We demonstrated the techniques to achieve shorter test interval of 50 cycles for run-time logic self-test. Techniques to address test power and handling of critical interrupts to CPU during field test are also discussed. Test time for memories in field has been reduced 5x by enabling testing of memories in parallel. On-chip techniques like DCC and analog loopback to ADC are described to enable test of analog modules. The techniques described are generic and are applicable to a wide range of SOCs targeted for safety critical applications.

Acknowledgment

The authors would like to thank R. Parekhji, who has helped in architecting the run time field test solution, D. Paul and S. Banerjee who have helped in experiments and analysis for test points insertion to aid power-up self-test.

References

[1] *ISO 26262, International standard for functional safety of electrical and electronic systems in production automobiles*, Std., 2011.

[2] *IEC 61508, International standard for functional safety of electrical/electronic/programmable electronic safety-related systems*, Std., 2010.

[3] V. Prasanth, R. Parekhji, and B. Amrutur, “Safety analysis for integrated circuits in the context of hybrid systems,” in *International Test Conference*, 2017.

[4] Enhanced Capture Module (eCAP) Reference Guide. [Online]. Available: http://www.ti.com/lit/ug/sprufz8a/sprufz8a.pdf

[5] Enhanced Pulse Width Modulator (ePWM) Reference Guide. [Online]. Available: http://www.ti.com/lit/ug/spruge9e/spruge9e.pdf

[6] P. Bernardi, L. Ciganda, M. de Carvalho, M. Grosso, J. Lagos-Benites, E. Sánchez, M. S. Reorda, and O. Ballan, “On-line software-based self-test of the address calculation unit in risc processors,” in *2012 17th IEEE European Test Symposium*. IEEE, 2012, pp. 1–6.

[7] J. Borcsok, A. Hayek, and M. Umar, “Implementation of a 1oo2-risc-architecture on fpga for safety systems,” in *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*. IEEE, 2008, pp. 1046–1051.

[8] V. Prasanth, D. Foley, and S. Ravi, “Demystifying automotive safety and security for semiconductor developer,” in *Test Conference (ITC), 2017 IEEE International*. IEEE, 2017, pp. 1–10.

[9] M. Bellotti and R. Mariani, “How future automotive functional safety requirements will impact microprocessors design,” *Microelectronics Reliability*, 2010.

[10] B. Keller and T. Bartenstein, “Use of misrs for compression and diagnostics,” in *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*. IEEE, 2005, pp. 9–pp.

[11] S. Gangasani, S. Alampally, P. Chowdhury, S. B. Chakravarthy, P. Sampath, and R. A. Parekhji, “Interruptible non-destructive run-time built-in self-test for field testing,” Aug. 5 2014, US Patent 8,799,713.

[12] B. Keller, “Encounter test opmisr/sup+/on-chip compression,” in *IEEE International Conference on Test, 2005*. IEEE, 2005, pp. 2–pp.

[13] R. Damodaran and A. Ramamurti, “Unique pbist features for advanced memory testing,” Jan. 6 2009, US Patent 7,475,313.

[14] P. V. Pillai and S. G. Langadi, “Background memory test apparatus and methods,” 2016, US Patent App. 15/346,737.