Architecture of a Reusable BIST Engine for Detection and Autocorrection of Memory Failures and for IO Debug, Validation, Link Training, and Power Optimization on 14-nm SoC

Bruce Querbach, Rahul Khanna, Sudeep Puligundla, and David Blankenbeckler
Intel Corp.

Patrick Yin Chiang
Oregon State University

Joseph Crop
Maxim Integrated

Editor's notes:
This paper presents the hardware and software architecture of a reusable BIST engine for 3D stacked 14-nm SoC, which also includes software-assisted autorepair of memory defects. Silicon results presented demonstrate the features of such engine such as easy silicon debug, validation time reduction by 3x, detection and repair of memory cell defects, etc. This solution has been successfully designed and used for seven Intel SoCs successfully debugged, tested, and launched into the market place.
—*Said Hamdioui, Delft University of Technology*

Process technology continues to shrink to 14 nm and beyond, which increases memory capacity and memory error rates [2]. Kim [12] has demonstrated 3-D stacked wide input/output (WIO) memory with one logic die followed by up to four memory die connected via through silicon via (TSV). WIO is similar to double data rate (DDR) memory, but offers much wider channel and input/output (IO) bandwidth than DDR through more parallel pins. TSVs are tiny vertical interconnects that pass through the die to enable connectivity of multiple dies. This further limits the probe ability of sandwiched die post bonding. When functional at-speed tests are required, built-in self-test (BIST) is the on-die solution that can test this three-dimensional stacked technology at speed compared to tester-based solution. BIST also solves the probe-ability

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/MDAT.2015.2445053
Date of publication: 15 June 2015; date of current version: 18 January 2016.

issue of three-dimensional stacked integrated circuit (3DS-IC) that other probe solutions cannot access, because BIST is built directly into the 3DS-IC.

Much prior work has been done related to memory test. Boundary scan [1] introduced direct current (dc) and stuck-at tests. In [3], Venkatesh et al. have demonstrated a physical memory fault model to test BIST algorithms. March and Modified-March algorithms [4]–[6] are commonly used to detect memory cell failures. A galloping pattern [7] can capture failures that escape the March algorithm. In [8], Bernardi et al. introduced a hardware/software cotest, which can be performed while the device under test (DUT) is operational. In [9], Bernardi et al. show a programmable memory BIST (MBIST) suitable for stacked DRAM, which uses March to detect neighborhood pattern sensitive faults (NPSFs). In [11], Querbach showed patterns that are quick to execute and provide good coverage for memory defects. This article presents the architecture of a BIST engine that is able to test memory cell failure according to the algorithms described in the prior works (scan, March, gallop, etc.), but goes beyond prior work by introducing autorepair of detected memory cell failure, IO link training, and power optimization/training. A comparison of features to prior work is shown in Figure 1.

The BIST engine tests system-on-a-chip (SoC) using the functional circuits themselves by exciting physical neighboring aggressors of a victim IO lane in Figure 2a, or a victim memory cell (called base) in Figure 2b. This new generation of reusable BIST engines is built into the SoC CPU, which is called converged pattern generator and checker (CPGC) [10]. Converged refers to combining interconnect BIST (IBIST) and MBIST capabilities into one BIST engine. The article will also describe an autorepair technology, which, combined with the fault detection capabilities, can physically remap defective memory using redundant memory cells built into the DRAM. This provides significant test time savings in post package manufacturing flow. CPGC also reduced SoC and platform power through closed-loop optimization of on-die termination (ODT) and equalization. Such BIST that supports IO and memory defect detection, memory autorepair, IO link training, and power optimization/training has not been previously demonstrated.

The rest of this article will be structured as follows. In Architecture of second-generation BIST engine for 14-nm SoC memory and IO, the architecture of CPGC is presented. CPGC software infrastructure architecture details the software architecture. In CPGC applications and results, silicon test results are presented. Last, in Summary, the article is summarized.

	Stuck at scan	Cell defect (march)	gallop	Link training	Auto repair	Power training
Treurn, etc [1]	Yes					
Yeh, etc. [4]	Yes	Yes				
Van de Goor, etc. [5]	Yes	Yes				
Shen, etc. [6]	Yes	Yes				
Mrugalski, etc. [7]	Yes	Yes	Yes			
CPGC	Yes	Yes	Yes	Yes	Yes	Yes

Figure 1. Comparison of CPGC features to prior work.

Architecture of second-generation BIST engine for 14-nm SoC memory and IO

CPGC architecture overview

The typical computer architecture for SoCs consists of one or more processors with a memory subsystem that includes a memory controller and a main memory. Three-dimensional stacked SoC significantly reduces the foot print of the memory and IO subsystem, because memories are stacked on top of the CPU, as shown in Figure 2a. The CPGC BIST engine becomes a part of the memory subsystem and the memory controller, which is shown in Figure 2b. The CPGC architecture consists of test pattern generators (TPGs), defect detectors (DDs), a repository of failing addresses (RF), an optional self-repair logic circuit (SF), and configuration registers (CRs). These capabilities (TPG, DD, RF, SF, and CR) form the basis of CPGC, which are present for each memory channel. Shown on the right of Figure 2b is the main memory with spare memory along with two typical memory test algorithms. The main memory can be written (W) sequentially from top to bottom and left to right, or as a base address walks through memory, an offset can stress the neighbors around the base. The CPGC can be programmed through the CR to automatically or manually repair memory defects detected with software assistance

or a hardware finite state machine (FSM), as shown in Figure 2d, using the spare memory of the DRAM to map out the defect rows.

CPGC architecture detail

Similar to [6], the CPGC BIST engine includes command, address, and data generator. The CPGC supports four instruction types: address, data, algorithm, and command. An offset instruction type is optional. When all the instructions are programmed, the lowest instruction loop is executed (offset) first, followed by command, algorithm, data, and address instruction loops.

Address pipeline. Given the dependencies between command, address, and offset, generating a complete address back-to-back on every clock to achieve high-speed test similar to [6], a pipeline architecture design is required.

An example of an address generation pipeline is shown in Figure 2c. The diagram shows pipe stages n00 and n01. When bank is ordered lower than column, it is lumped with column into pipe stage n00. Carry count is a look-ahead carry propagation to ensure that row address has valid inputs in pipe stage n01. Rank address is lumped with row address into pipe stage n01.

Autorepair FSM and autorepair technology. The CPGC reusable BIST engine introduced an optional hardware repair engine that works in conjunction with software to provide autorepair technology. This repair capability has been adopted by the JEDEC as a part of the WIO industry standard, known as post package repair (PPR). In a high-volume manufacturing (HVM) environment, the automatic test equipment (ATE) can use the CPGC row hammer to detect memory cell defects. At the end of the detection phase of the test, the ATE test flow could initiate either a hardware-automatic or a software-assisted manual repair of the failed memory cells using spare rows of the DRAM device. The PPR JEDEC standard has been adopted by DRAM device manufacturers. Specifically, future DRAM are built with extra spare rows of memory to compensate for the increased likelihood of memory cell failures. The automatic modes require no operator intervention, thus significantly reducing HVM test time from many minutes down to seconds.

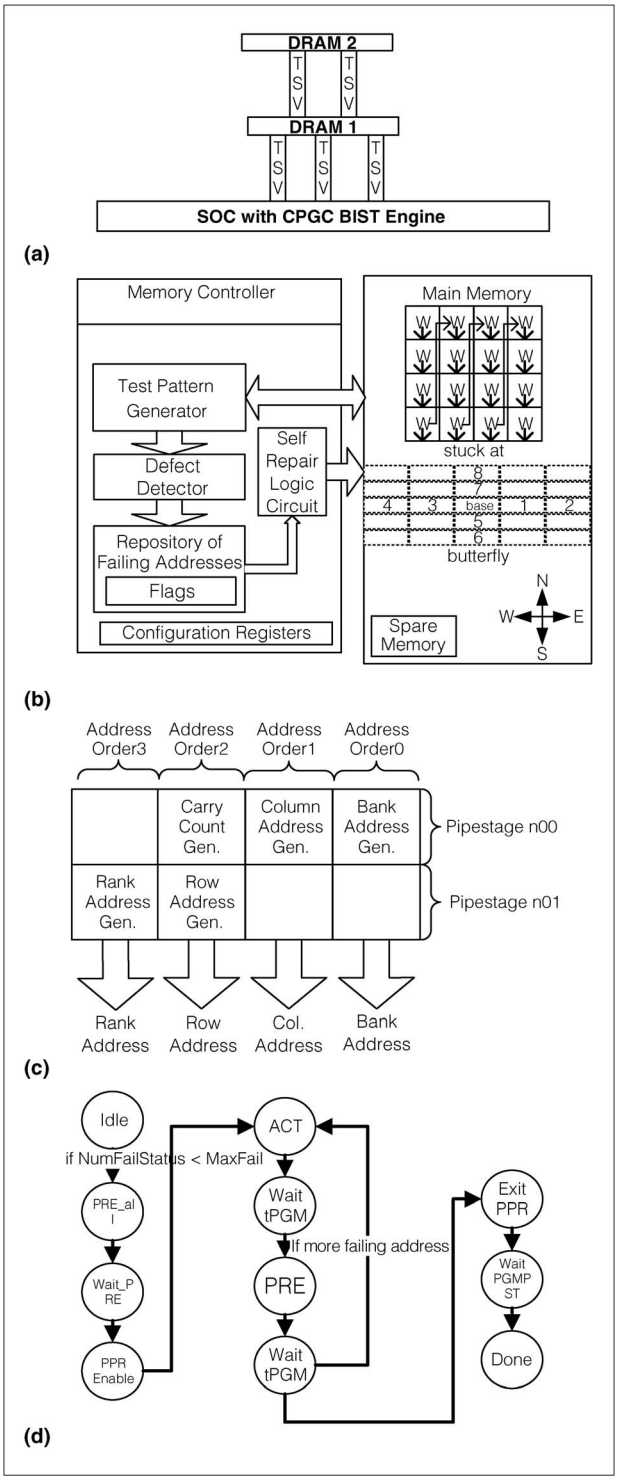

Figure 2. SoC memory subsystem and CPGC architecture overview. (a) CPU with CPGC BIST engine. (b) CPGC BIST engine architecture as part of Intel Memory Controller. (c) The CPGC BIST engine pipeline architecture. (d) The CPGC BIST engine autorepair architecture.

The autorepair and manual repair can also be run at boot via firmware or BIOS. This enables the field PPR of a highly integrated SoC with a WIO memory subsystem, such as a phone or a tablet. Such end-user autorepair can significantly improve manufacturing yield and end-user customer experience. Imagine that instead of throwing away a dead phone, we could get customer service to push an autorepair software update, or even have the phone simply automatically repair itself on the next reboot after a crash.

Figure 2d shows an 11-state FSM that loops between activate (ACT) and wait for programming time completion (Wait tPGM) if multiple failing addresses need to be repaired. All 11 states are used to complete PPR. The PPR flow will be activated if the number of failures (NumFailStatus) is lower than the maximum number of failures (MaxFail) that the spare memory could support.

Refresh control generator. Refresh control is important to maintain DRAM data integrity. It is also an excellent tool to test and stress the DRAM for cell retention failures. The CPGC BIST engine has a variable refresh signal generator to control and stress the DRAM refresh rates.

Algorithmic memory cell testing

The CPGC implementation has been designed with significant flexibility to accommodate a broad variety of different memory test algorithms efficiently. To accomplish these address walking algorithms, the CPGC architecture test pattern generator consists of the nested loops of address instruction (AI), data instruction (DI), algorithm instruction (GI), command instruction (CI), and offset instruction (OI). Some examples include the following.

Memory scan. A type of test that is commonly used to initialize memory and detect stuck at fault memory failures is called memory scan, which is shown in the upper right corner of Figure 2b. The scan writes "0" to the entire memory, which clears the memory contents at boot, and catches any command, address, and data bus communication errors. A subsequent read scan is done to detect any errors and provide coverage for stuck at "1" faults. The process is repeated by writing "1" and reading to verify to catch stuck at "0" faults. Some implementations follow with a second scan of write and read "0" to provide full stuck at coverage since the initial memory contents were random.

Row hammer and butterfly. Conceptually, DRAMs are fundamentally made up of a large array of capacitors in the form of holes in the substrate. A "1" is represented by a capacitor holding a charge; a "0" is represented by a discharged capacitor. There are various physical mechanisms where coupling effects between neighboring capacitors can result in bit flips. The opportunity for these cross-charge coupling fault failures increases as device size decreases. The charge sharing impact of the previous bit in time can cause intersymbol interference (ISI) failures, which increases as DRAM frequency increases. Interconnect and wires for SoC internal and external IO face similar crosstalk and ISI. Due to finite charge leakage, DRAM cells are refreshed to retain their stored value. However, slower refresh rates lead to additional DRAM retention failures.

An example test algorithm, typically called "butterfly," for detecting neighbor cross coupling is shown in the right side of Figure 2b. In this test, the nearest neighbors in the *x*- and *y*-location are written to with opposite data as the victim, or base, cell in a crosshair shape and up to two memory cell locations away (Wr1, Wr2, Wr3, Wr4, Wr5, Wr6, Wr7, and Wr8). The base is read at the end of these write commands to check for errors. The butterfly algorithm can test near neighbors (crosshair), all neighbors (crosshair and diagonal), and selective neighbors.

Other cell test algorithms. The CPGC engine is fully capable of supporting many different types of memory cell test algorithms, such as MATS+, March LR, March C-, PMOVI, and even more complicated algorithms to target certain types of DRAM faults. In addition, the CPGC can easily set up the background data pattern prior to beginning the test.

CPGC software infrastructure architecture

The CPGC technology has built-in customizable flexibilities enabling the testing and diagnosis of entire memory subsystems. In case of impending failure, it is

possible to isolate the memory field-replicable unit (FRU) for further testing through CPGC IP. The FRU would take some or all of the memory offline, and use CPGC to test the memory. The BIOS-based unified extensible firmware interface (UEFI) architecture supports a middle-ware layer that can incorporate the CPGC application program interface (API) at boot-time as well as the runtime. At boot-time, the CPGC-based test could execute at the initialization of the memory reference code. At runtime, the code could execute in the system management interrupt (SMI), which is a virtualized space in the system. The API libraries are incorporated in both boot-time environments as well as in runtime environments. While boot-time CPGC code could execute at every boot, runtime code cannot be executed frequently. Execution of runtime code can cause system perturbations that can degrade the performance of the workload running on it.

A firmware level approach to host CPGC tests and APIs requires building a memory-map infrastructure for identifying memory addresses. Memory initialization code is modified to CPGC test code. The SMI provides the necessary APIs to assist in constructing and configuring the CPGC tests. Figure 3 illustrates the UEFI BOOT SERVICES layer that hosts the CPGC test modules. Similar modules are replicated into the SMI for runtime execution.

UEFI is a standard-based modern software architecture (originated by Intel) that offers a rich extensible pre-operating system (OS) environment with advanced boot and runtime services. It has an intrinsic networking capability and is designed from the ground up to work with multiprocessor (MP) systems. The UEFI standard is architected for dynamic modularity. This has opened up opportunities for collaboration on the technology that allows reducing development costs and time to market through code sharing and easy integration of plugin modules from different partners and vendors.

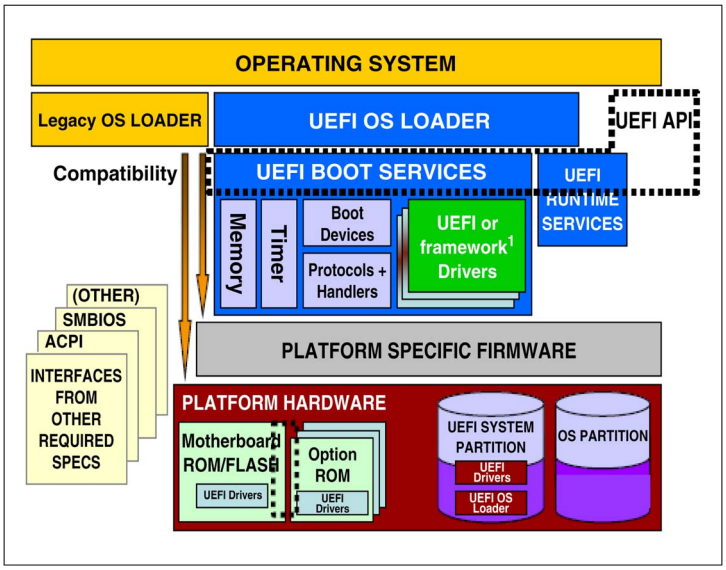

Figure 3. UEFI specification compliant BIOS that boots to an operating system that hosts and supports CPGC tests.

CPGC applications and results

Memory initialization and IO link training using CPGC

One of the important applications where CPGC is extensively used is memory initialization and training during system power up. Typical link training algorithms use eye centering techniques on the received signal to ensure maximum possible margins on either side of sampling instants in voltage and timing axis.

CPGC writes a stream of patterns on the IO link, which are either generated via built-in linear feedback shift registers (LFSRs) or via user-programmable registers. This stream of data is stored in memory, read back, and compared to the original by CPGC. CPGC intentionally degrades and margins the IO write or read path independently while transmitting, receiving, and checking for errors to identify the high-side and low-side limits of voltage margin, and the left-side and right-side limits of timing margin to generate a 2-D eye diagram (Figure 3c) for the link under test. The CPGC applies the eye centering technique by calculating the midpoint between the high-side and low-side limits of the voltage margin, and between right-side and left-side limits of the timing margin. These voltage midpoint and timing midpoint become the optimal eye center as a result of CPGC IO link training. CPGC applies the link training algorithm to all the command, address, and data pins of each memory channel, and all memory and high-speed IO attached to the CPU (e.g., PCIe). During the training, eye margin is automatically measured and reported.

ISI and crosstalk debug using CPGC

Another application of CPGC is identifying IO link performance loss due to data-dependent jitter

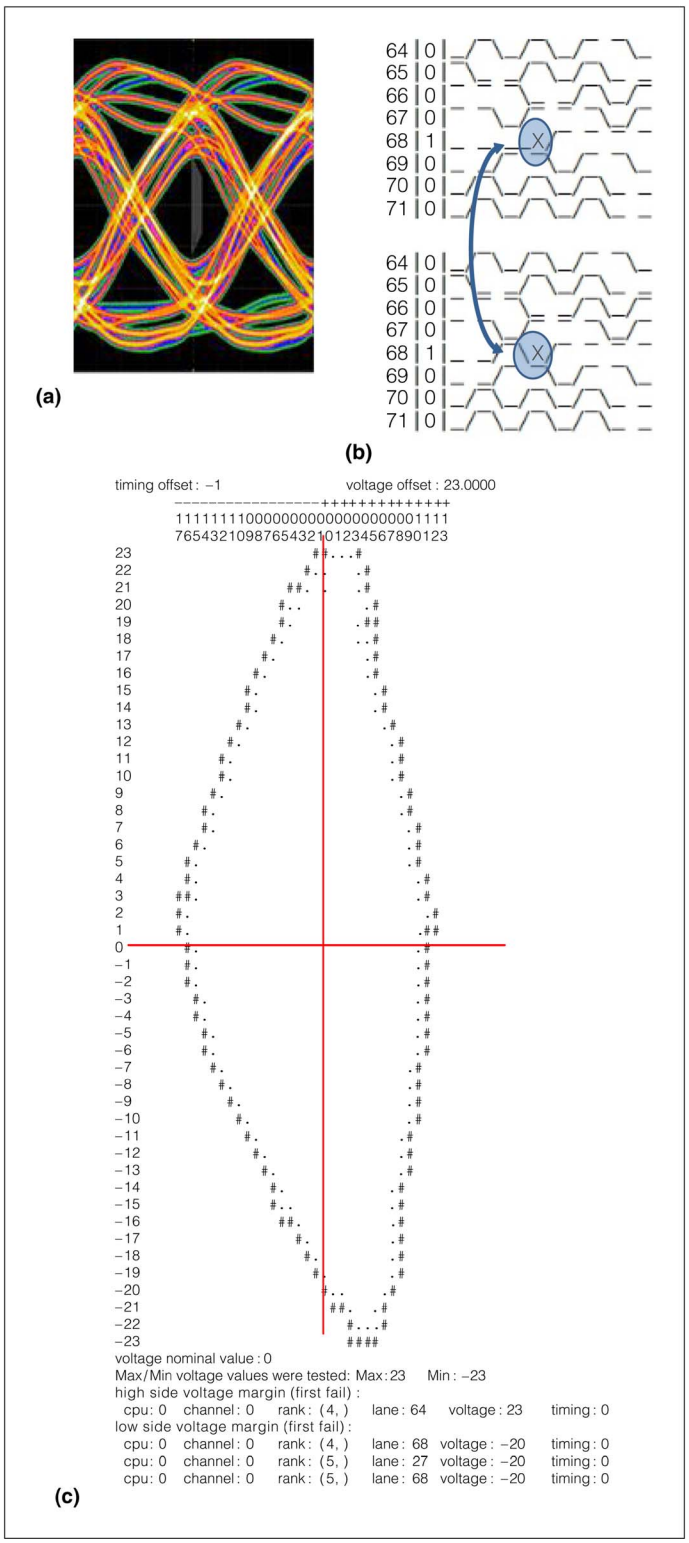

Figure 4. CPGC silicon results. (a) Scope capture of DDR IO eye. (b) ISI debug changing precursor from "0" to "1." (c) CPGC BIST-based memory IO initialization, training, and eye measurement.

such as ISI and crosstalk through bit-by-bit pattern searches. A DDR memory IO bus is a single-ended voltage termination bus with very closely placed lanes between processor and DDR dual inline memory module (DIMM), which makes this bus susceptible to ISI and crosstalk issues. CPGC can be used to program and transmit stress patterns on the bus that can cause a worst case 1 or 0 transition after a long run of identical bits. One such example, used in DDR validation to measure eye height loss due to ISI, is shown in Figure 3b. In this case, CPGC implemented in silicon is programmed to change the precursor bit on a DDR pin (pin 68) in a deterministic method from a 0 to 1 after a long sequence of 0 s. This 0 to 1 transition is immediately followed by a 1 to 0 transition to create a worst case 1 during this bit period that corresponds to the inner contour of the eye diagram shown in Figure 3a, thus reducing the eye height margin. Similarly, a worst case 0 can be created to stress the bus. Such programmability in CPGC helps to debug any ISI. CPGC also allows simultaneous programming of multiple lanes on the bus with different deterministic or random patterns and provides independent control of offset/skew in timing between the lanes. This feature is used to create a victim–aggressor condition among the lanes for crosstalk studies and mitigation.

Another way of stressing a memory cell is through repeated writes to the neighboring cells, called hammering. A row hammer issues repeated "activate" commands to specific aggressor rows to cause victim row/bit failures. CPGC can stress additional +2 and +3 aggressor rows, which is specifically suited for the WIO TSV testing of memory, when physical probing of a sandwiched die is impossible.

Silicon and system validation using CPGC

The CPGC engine is also used in silicon and system validation. A unique feature of CPGC is its ability to drive patterns through memory IO without requiring the core of the processor to be working. This feature enables early silicon debug and characterization of the IO circuit physical layer independent from the processor. In an HVM system validation environment, after the initial bringup of the complete system. CPGC provides a pattern autorotation feature to automatically feed test patterns from one lane to the next and switch

between victim and aggressor roles. The autorotation of victim and aggressor patterns is shown to decrease HVM validation time by 3x.

Platform power savings using CPGC

In addition to memory IO initialization and training, CPGC is also used to optimize the power and performance of DDR IO. CPGC implementation provides controls via configuration registers to closed-loop compensation circuits such as reference voltage (Vref), slew rate, drive strength, and on-die termination. These control features are optimized and tuned for the lowest power while maintaining acceptable eye opening and bit error rates at the receivers. The results showed that a power savings of 6 and 12 mW/b were achieved on CPU power (CpuPwr) and DIMM power (DimmPwr), respectively, according to the plot average shown in Figure 4 when ODT was tuned. By calibrating for power, margin reductions of −5, −8, +11, and −19 ticks were observed on receiver timing (RdT), transmitter timing (WrT), receiver voltage (RdV), and transmitter voltage (WrV). There are a total of 64 controllable states of voltage and timing offset in the *x*- and *y*-direction. Each shift in state is called a tick. However, the reduction in margin did not affect the overall system bit error rate (BER) of 10^{-12}. This margin/power tradeoff led to 5.28% CPU power savings on 22-nm 15-W CPU and 11.03% power savings on the platform.

The CPGC engine addresses the complexity of designing and testing today's IC that prior tester and probe solutions no longer work due to probe ability and higher frequency IO and memory. This article presented the hardware and software architecture of a reusable BIST engine for Intel SoC that includes the autorepair of detected memory defects.

Key CPGC silicon results and benefits are: debug isolation of ISI and crosstalk; memory IO initialization, training, and margin; autorotation that led to 3x validation time improvement; isolation of memory subsystems to enable early debug that saved Intel up to three months of product delay; row hammer, butterfly, and galloping that enabled WIO TSV testing of memory; and 5%–11% of CPU and platform power savings through closed-loop calibration of DDR IO circuits.

This reusable CPGC BIST engine IP solution has been successfully designed into Intel products,

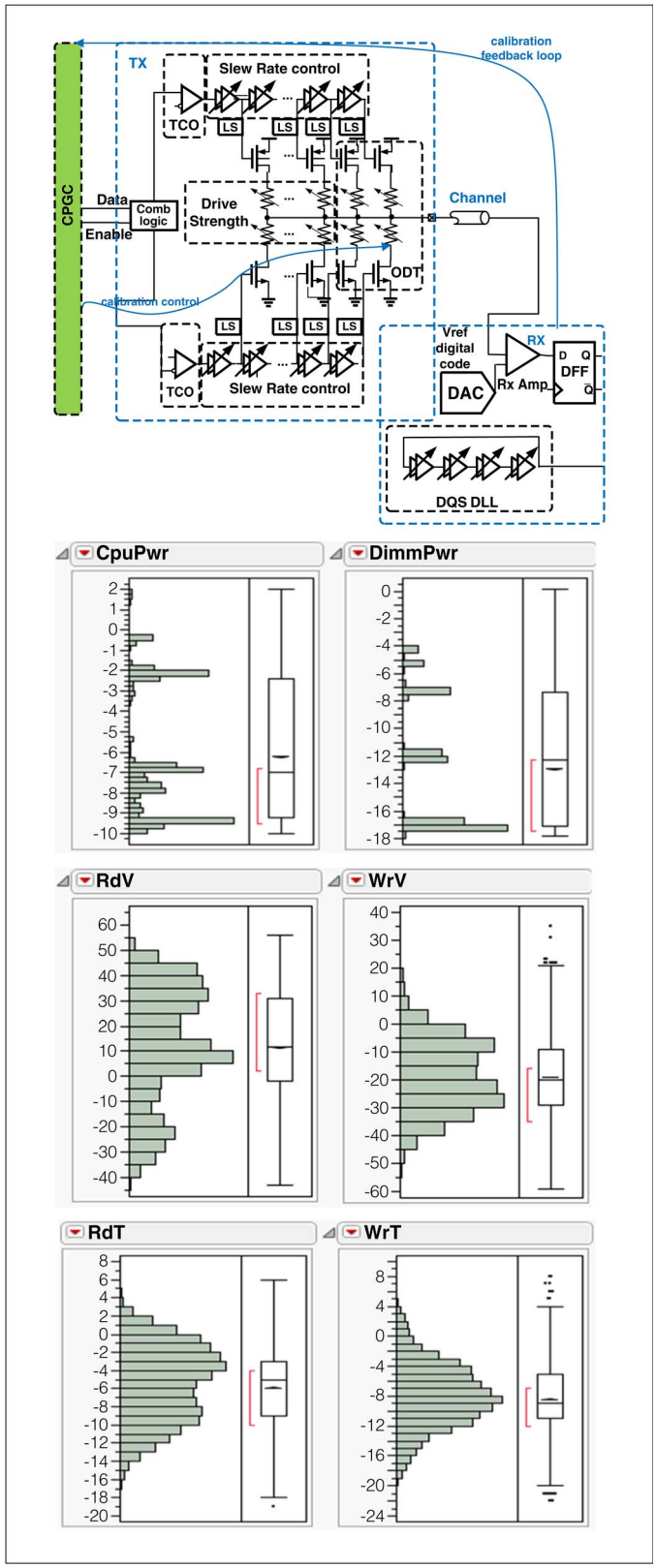

Figure 5. CPGC closed calibration loop saves CPU and DRAM power.

with at least 11 Intel SoCs using CPGC for IO link training, validation, and debug. At least seven Intel SoCs have been successfully debugged, tested, and launched in the marketplace using the reusable CPGC BIST engine. ■

Acknowledgment

We would like to thank A. Taylor and T. K Poriyani House Raju for their wonderful support and effort on this CPGC project.

■ References

[1] B. G. Van Treuren, C. Chiang, and K. Honaker, "Problems using boundary-scan for memory cluster tests," in *Proc. IEEE Int. Test Conf.*, Oct. 28–30, 2008, pp. 1–10, DOI: 10.1109/TEST.2008.4700645.

[2] D. Blankenbeckler, A. Norman, and M. Shepherd, "Comparison of off-chip interconnect validation to field failures," in *Proc. 3rd Int. Conf. Adv. Syst. Test. Validat. Lifecycle*, Barcelona, Spain, 2011, pp. 121–125.

[3] R. Venkatesh, S. Kumar, J. Philip, and S. Shukla, "A fault modeling technique to test memory BIST algorithms," in *Proc. IEEE Int. Workshop Memory Technol. Design Test.*, 2002, pp. 109–116.

[4] J.-C. Yeh et al., "Flash memory built-in self-test using March-like algorithms," in *Proc. 1st IEEE Int. Workshop Electron. Design Test Appl.*, 2002, pp. 137–141.

[5] A. J. Van de Goor, S. Hamdioui, and H. Kukner, "Generic, orthogonal and low-cost March Element based memory BIST," in *Proc. IEEE Int. Test Conf.*, 2011, pp. 1–10, DOI: 10.1109/TEST.2011.6139148.

[6] S. C. Shen, H.-M. Hsu, Y.-W. Chang, and K.-J. Lee, "A high speed BIST architecture for DDR-SDRAM testing," in *Proc. IEEE Int. Workshop Memory Technol. Design Test.*, 2005, pp. 52–57.

[7] G. Mrugalski et al., "Fault diagnosis in memory BIST environment with non-march tests," in *Proc. 20th Asian Test Symp.*, 2011, pp. 419–424.

[8] P. Bernardi, L. Ciganda, M. S. Reorda, and S. Hamdioui, "An efficient method for the test of embedded memory cores during the operational phase," in *Proc. 22nd Asian Test Symp.*, 2013, pp. 227–232.

[9] P. Bernardi, M. Grosso, M. S. Reorda, and Y. Zhang, "A programmable BIST for DRAM testing and diagnosis," in *Proc. IEEE Int. Test Conf.*, 2010, pp. 1–10, DOI: 10.1109/TEST.2010.5699247.

[10] B. Querbach, "CPGC2, A buit-in self test and auto-repair engine for DRAM (WIO, DDR4)," U.S., Patent pending 14/141,239.

[11] B. Querbach, "Comparison of hardware based and software based stress testing of memory IO interface," *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, 2013, pp. 637–640.

[12] D. W. Kim et al., "Development of 3D through silicon stack (TSS) assembly for wide IO memory to logic devices integration," in *Proc. IEEE 63rd Electron. Compon. Technol. Conf.*, May 28–31, 2013, pp. 77–80.

Bruce Querbach is an Architect in the Data Center Cloud Computing Group at Intel Corporation, Hillsboro, OR, USA. His interests include high-speed IO, memory, design for test, and CPU/memory computer architecture. Querbach has a BaSc from Simon Fraser University, Burnaby, BC, Canada, an MS from Georgia Institute of Technology, Atlanta, GA, USA, an MBA from the University of California Davis, Davis, CA, USA, and a PhD from Oregon State University, Corvallis, OR, USA.

Rahul Khanna is a Principal Engineer in the Software Solutions Group at Intel Corporation, Hillsboro, OR, USA. His interests include wireless sensor network, autonomics, and design for test. Khanna is currently working toward a PhD at Oregon State University, Corvallis, OR, USA (expected in 2016).

Sudeep Puligundla is an Architect in the Data Center Cloud Computing Group at Intel Corporation, Hillsboro, OR, USA. His interests include high-speed IO, design for test, and signal integrity of platform and systems. Puligundla has a PhD from the University of Florida, Gainesville, FL, USA.

David Blankenbeckler is a Principal Engineer at Intel Corporation, Hillsboro, OR, USA, where he drives I/O and memory electrical validation strategies. Blankenbeckler has a BS in electrical engineering from Clemson University, Clemson, SC, USA and an MS in management in science and technology from Oregon Graduate Institute, Portland, OR, USA.

Joseph Crop has been with Maxim Integrated, Hillsboro, OR, USA, since 2014, where he designs mixed-signal SoCs and data converters. His current research interests include near-threshold circuits and systems, digitally enhanced analog circuits, and integrated circuit reliability. Crop has a PhD in

electrical engineering from Oregon State University, Corvallis, OR, USA (2014).

Patrick Yin Chiang is currently an Associate Professor in Electrical and Computer Engineering at Oregon State University, Corvallis, OR, USA. He has published more than 120 conference/journal publications, in the areas of energy-efficient optical and wireline interconnects, robust near-threshold computing, and biomedical sensors. Yin Chiang has a PhD from Stanford University, Stanford, CA, USA.

■ Direct questions and comments about this article to Bruce Querbach Intel Corp., Hillsboro, OR 97124, USA; Bruce.querbach@intel.com.