A Reusable BIST with Software Assisted Repair Technology for improved Memory and IO debug, validation and test time

Bruce Querbach
Intel Corporation
{Bruce.querbach

Rahul Khanna
Intel Corporation
rahul.khanna

David Blankenbeckler
Intel Corporation
david.blankenbeckler

Yulan Zhang
Intel Corporation
yulan2.zhang

Ronald T Anderson
Intel Corporation
ronald.t.anderson

David G Ellis
Intel Corporation
dg.ellis}@intel.com

Zale T Schoenborn
Intel Corporation
zale.t.schoenborn@intel.com

Sabyasachi Deyati
Georgia Tech
deyati@gatech.edu

Patrick Chiang
Oregon State University
pchiang@eecs.oregonstate.edu

ABSTRACT

As silicon integration complexity increases with 3D stacking and Through-Silicon-Via (TSV), so does the occurrence of memory and IO defects and associated test and validation time. This ultimately leads to an overall cost increase. On a 14nm Intel SOC, a reusable BIST engine called Converged-Pattern-Generator-Checker (CPGC) are architected to detect memory and IO defects, and combined with the software assisted repair technology to automatically repair memory cell defects on 3D stacked Wide-IO DRAM. Additionally, we also present the CPGC gate count, power, simulation, and silicon results. The reusable CPGC IP is designed to connect to a standard IP interface, which enables a quick turn-key SOC development cycle. Silicon results show CPGC can speed up validation by 5x, improve test time from minutes down to seconds, and decrease debug time by 5x including root-cause of boot failures of the memory interface. CPGC is also used in memory training and initialization, which makes it a critical part of Intel SOC.

Keywords

High speed IO, Memory IO, Memory Interface Training, Post Package Repair, Memory Array Test Engine, hardware/software calibration

1 Introduction

Shrinking process technology and faster memory bus data continue to drive complexity in the I/O design. This surge in complexity brings with it an increase design bugs. While some of these bugs (such as logic bugs) can be detected at the pre-silicon stage, others (such as cross coupling of adjacent data lanes, voltage droop, ground bounce, power bounce, timing error due to process shifts, etc.) remain undetected. Although logic bugs are possible to detect in pre-silicon, with increase in design complexity, shortened design time, and complex OS and software demands, this can lead to escalations of bug escapes. A modern microprocessor design can experience considerable design bugs specifically in memory interfaces. Sarangi et. al. [1] describe the percentages of memory interface design bugs found in AMD64 and Intel's Pentium 4 to be 17.4 and 20.2 respectively. Therefore there is a critical need for an efficient methodology to validate the memory interface in a microprocessor design.

Controllability and observability are key attributes in any post silicon validation methodology. One conceivable but expensive solution is to probe the prepackaged Dynamic Random Access Memory (DRAM) die. The easiest and most cost effective solution is to include Built In Self Test (BIST) circuitry inside the chip. The BIST engine provides an efficient mechanism to deterministically drive and receive patterns on the memory bus along with automated data checking capability. These features can be used to swiftly identify and isolate bugs in the silicon validation phase. Furthermore, given the continuous increase in system reliability requirements and the current trend of adding more complex features to achieve higher reliability goals, this paper aims to present an effective automated repair strategy that combines CPGC with software assistance in protecting against memory errors. In addition, we present possible directions for designing and testing future systems that are resilient to memory errors.

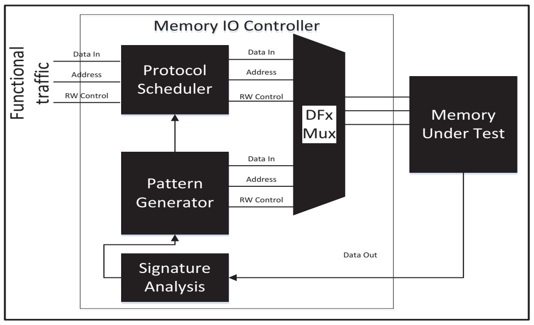


Figure 1: CPGC BIST engine with software assisted repair technology

In addition to enabling efficient validation of the memory

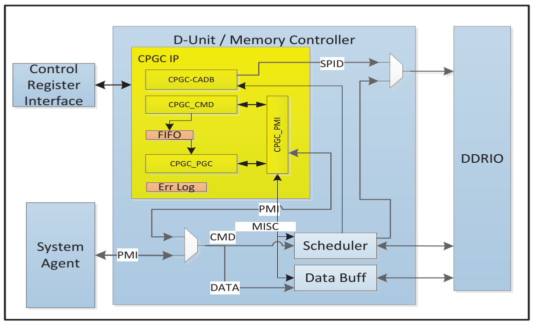


Figure 2: CPGC architecture overview

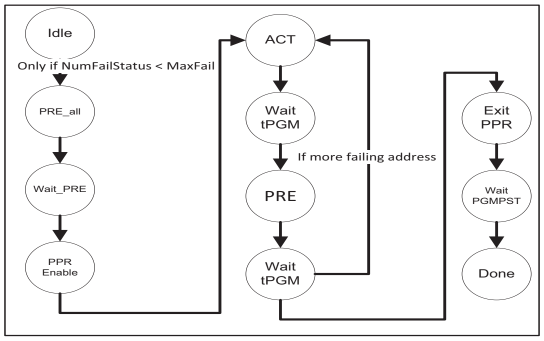


Figure 3: CPGC SW assisted repair technology flow

controller, a BIST engine facilitates effective identification of DRAM defects. DRAM errors are one of the leading causes of hardware failure in modern compute clusters [10, 11]. This has led to increasing requirement for Reliability, Availability, and Serviceability requirements (RAS), particularly in mission critical environments. Standard OS based memory stress tools pose a significant challenge in providing adequate coverage to the many types of DRAM faults due to the presence of architectural features in the memory controller, such as data scrambling, reordering, and various levels of interleaving. These features make it difficult to generate the types of patterns or coverage that the test writer intends. With a suitable design, a BIST engine can overcome these challenges by providing the means to precisely put the intended patterns on the bus, and thereby significantly improving the ability to identify DRAM faults. Once identified, the BIST engine with memory controller and software assisted repair technology can provide a means to enable automatic repair of memory cell defects in 3DS WIO DRAM.

In this paper, the authors propose a new architecture (Figure 1) for Converged-Pattern-Generator-Checker (CPGC). It is capable of IO [13–15, 21] and memory test, debug, validation, IO link training [16–18], and post package repair. The architecture is suitable for such technologies as embedded memory, DDR4, and WIO on System On Chip (SOCs) with 3D stacking and TSV topology. To speed up fault isolation and improve manufacturing yield, the CPGC BIST engine also includes an software repair technology for 3D stacked memory cell defects.

In section 2, the paper presents the memory and IO failures modes. In section 3, the paper describes the CPGC architecture. In section 4, the paper discusses how CPGC uses standardized interface to enable Intellectual Property (IP) reuse, which expedites rapid prototyping and a quicker SOC product development cycle. In section 5, the paper presents software architecture. In section 6, the paper presents simulation results based on CPGC architecture. In section 7, silicon results demonstrate that CPGC significantly reduces debug and validation time. In section 8, the paper summarizes all the results presented in this paper.

2 Failure Modes

This paper will briefly cover various failure modes that occur within the memory IO and cell due to cross coupling of electrical signals or charges.

There is significant literature available regarding the different types of fault modes for DRAMs and the associated algorithms to detect them. In [2] the authors have demonstrated a fault model (to model all plausible physical faults in memory) to test the BIST algorithms. The most prevalent memory BIST algorithms are "'march"' and "'modified march"' algorithms [3–5]. In [6] the authors propose a non- march ($O(n^2)$ complexity) algorithm for memory test where it employs galloping pattern tests and captures some of the failures escaping conventional march based tests. While the complexity is higher compared to march based tests, this technique is helpful for embedded memory testing. In [7] the authors introduce a hardware/software co-test for embedded memory in SOCs. The mentioned co-test is an online test; i.e., it can be performed when the device under test (DUT) is operational. In [8] the authors have shown a programmable MBIST suitable for stacked DRAM as well as standalone DRAM. It supports both march and Neighborhood Pattern Sensitive Faults (NPSF) tests. March based MBISTS alone are not sufficient for the state of the embedded memory on SOCs.

As described in [2, 9], following are the common memory faults that can be detected by this CPGC IP.

1. Stuck at Faults
2. Address Decoder Faults
3. Transition Faults
4. Inversion Coupling Faults
5. Idempotent Coupling Faults
6. State Coupling Faults
7. Retention Fault
8. Neighborhood Pattern Sensitive Faults

In addition, common IO faults which can be detected by this CPGC IP are:

1. ISI (time domain interference)
2. Crosstalk (Neighbor Victim/Agressor interaction)
3. Clock domain crossing issues
4. Power supply issues
5. Logic circuit IP integration issues

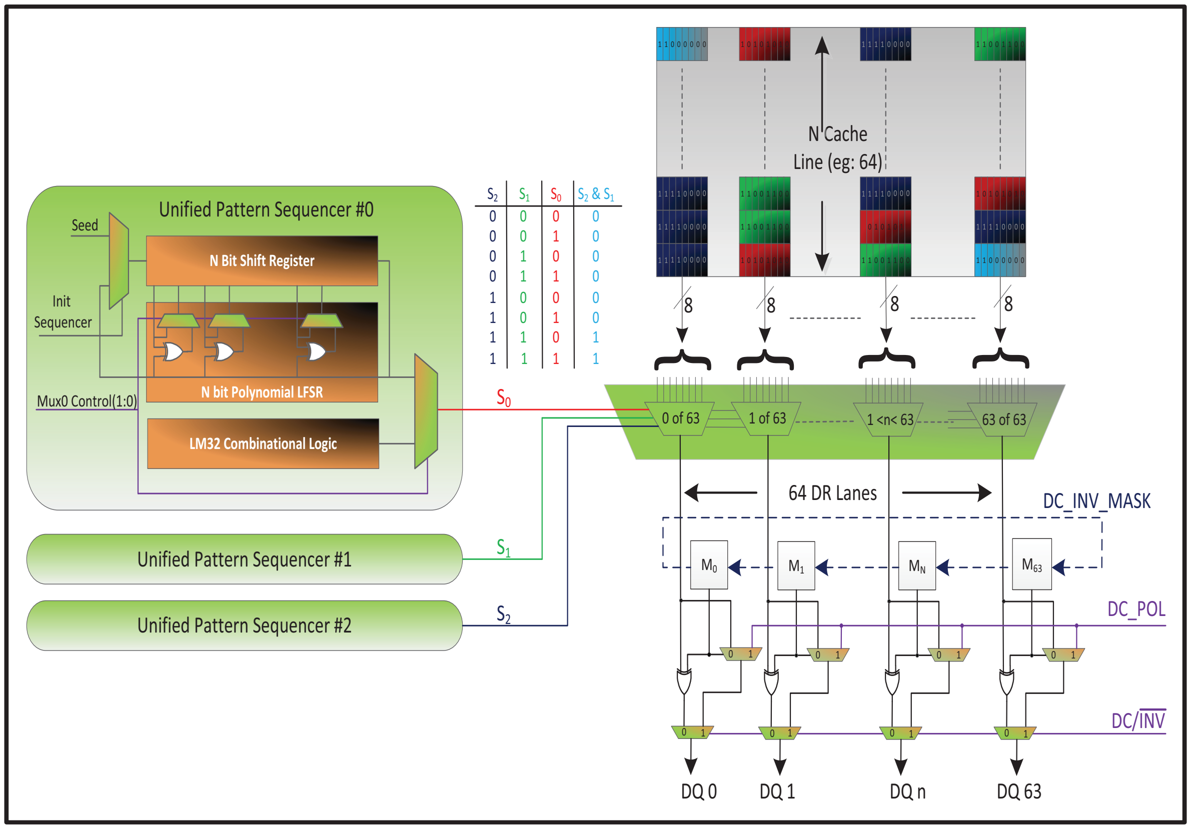


Figure 4: CPGC unified Pattern sequencer and buffer

IP block	Gate Count (um^2)
dpatunit	13621
cmdunit	1847
configunit	3751
errlogunit	100
errunit	874
unisequencer	169
msgifunit	1714
cpgct	21953
Total	44028

Table 1: CPGC architecture gate count

IP block	Active Power (mW)
dpatunit	1.796
cmdunit	0.244
configunit	0.495
errlogunit	0.013
errunit	0.115
unisequencer	0.022
msgifunit	0.226
cpgct	2.894
Total	5.805

Table 2: CPGC architecture power estimates

3 CPGC ARCHITECTURE

Figure 2 illustrates the block diagram of the CPGC architecture where the CPGC IP is embedded in the memory controller. CPGC supports both pre-scheduler (CPGC_T) and post-scheduler (CPGC_C) patterns. Functional traffic from the SOC core that originates from upstream is multiplexed with CPGC to form the command (CMD) and data of the DDR protocol. It is transmitted through the memory scheduler and Data-Buffer (Data Buff) to the DDRIO. CPGC control registers are programmed through a standard low bandwidth interface (Side Band), and an out of band protocol interface exists between CPGC and DDRIO. The CPGC IP comprises several PRBS (pseudo random binary sequence) generators such as the CPGC Pattern Generator Checker (CPGC_PGC), the CPGC command generator (CPGC2_CMD) and the CPGC Command-Address Data Buffer (CPGC_CADB). These pattern generators can generate fixed, low freq (LMN), and random patterns. The pattern can drive command, address, and data IO of the memory interface.

3.1 Architecture for Software Assisted Repair Technology

The CPGC reusable BIST engine introduces a software assisted repair technology. The flow is shown in Figure 3, which has been adopted by the JEDEC as a part of the WIO industry standard. Due to heat from solder reflow during the attachment TSV and 3DS, additional memory cell defects could be introduced during the packaging process. Through JEDEC and WIO industry standards, agreement has been reached with the DRAM manufactures to provide spare rows of additional memory cells for Post Package Repair (PPR). The CPGC architecture for software assisted repair technology will conduct this PPR if necessary at the end of memory cell defect discovery. After CPGC detects memory failures, it supports both manual and automatic modes depending on the software configuration to repair the defective cell using the spare cells that the DRAM manufacturer provided. The flow is illustrated below. In the manual mode, the detection phase is executed separately from the repair phase, which is fitting for test and debug to ensure accurate detection of failures. In the automatic mode, the detected failures are automatically repaired by the BIST engine with software assistance. The repair is stored in the NVRAM of the DRAM device, hence repairs are permanent, and persistent through subsequent reboots and power cycles. The combining of the two phases can speed up High Volume Manufacturing (HVM) test significantly from many minutes down to seconds.

The detection and the repair phase can also be run via BIOS or at boot via firmware. This enables post packaging repair of highly integrated SOC with a WIO memory subsystem in the field. In the field, repair is especially valuable to the smart phone and tablet market. Since memory are soldered down, any defective memory could cause the whole phone/tablet to be tossed out. The in the field repair can improve yield and reduce debug and validation time, thus accelerating the product development and launch cycle.

3.2 Unified Pattern Sequencer

Each of the pattern generators supports different bus widths, but is built upon the Unified Pattern Sequencer (UPS). As illustrated in Figure 4 it consists of PRBS with programmable length of 16, 23, 32, a low freq LMN counter that can generate a clock waveform as low as 10 Khz, and a 32-bit fixed pattern buffer.

3.2.1 LMN

L, M, N are three letters in the alphabet that denote three 32 bit counters. The L counter counts the number of clocks to drive the initial value. The M counter counts the number of clocks to drive the high value, and the N counter counts the number of clocks to drive the low value. Combined, the LMN generator can generate a very low frequency (on the order of 10Khz) clock waveform of any duty cycle to excite low frequency resonance of the system.

3.2.2 Fixed Pattern Buffer

Advanced software algorithms frequently derive a large set of user defined patterns to try. These can be programmed through the 32bit fixed pattern buffer. The pattern is simply repeated over and over again until changed by the software.

3.2.3 LFSR

Multiple LFSR tap lengths (16, 23, 32) of maximal running pseudo random pattern are included in the PRBS generator.

3.2.4 Flop Based Data Pattern Buffer

Three instances of the uni-sequencer (UPS) are multiplexed together to construct the downstream pattern. These 8:1 per lane MUX are controlled by an eight bit wide by 64 lanes pattern buffer in a small gate count flop based design as illustrated in Figure 4.

3.2.5 Register File based pattern

Depending on the IP instantiation and die area constraint, the CPGC IP can support a register file (RF) based design, achieving 64 cache line deep of pattern, with each cache line being 8 bits deep in time. The RF design has 1 write port and 2 read ports to minimize gate count and area.

3.2.6 Data Pattern Dynamic Inversion/DC

CPGC supports DC patterns such as all zero/one, as well as dynamic inversion of the data dependent on the address. The dynamic inversion feature enables rapid checkerboard testing of memory cell integrity.

3.2.7 Post-Scheduler Command and Address Pattern Generation

This supports both In-protocol and Out-of-protocol enables early investigation of future memory protocols, or isolating a specific debug failure by stressing rare or difficult sequences.

3.2.8 Error status, logging and Masking

CPGC supports per lane, and per bit time error counters. Similarly CPGC offers mask capability at per lane and per bit time level. CPGC error and data logging enables precision debug of memory and IO failures.

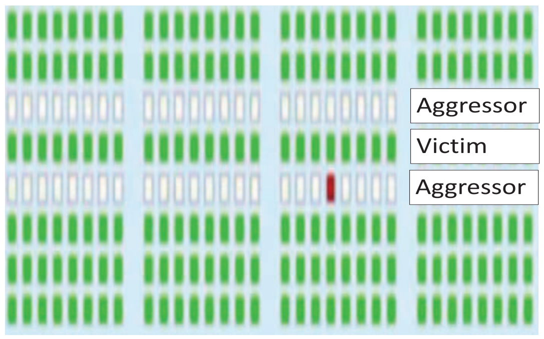


Figure 5: CPGC Row Hammer Stress pattern

3.2.9 Command and Address Generator

Through CPGC2_CMD, the command and address generator has the ability to detect latest faults with the current DRAM process and topology, specifically for WIO that uses 3D stacking and Through Silicon Via (TSV).

Row hammer problems [19,20] occur when excessive ACTIVATE commands targeted at a specific row (aggressor) introduce cross talk that can cause a bit-flip error in an adjacent row (victim) (Figure 5). Repeated aggressor row charging and discharging can result in one or more failures, as indicated by the red cell in the figure. The CPGC not only can row hammer the +1 and -1 row to detect the traditional row hammer failures, but it can also hammer and detect problems caused by the +2 and +3 rows. These CPGC features are specifically suited for the WIO TSV testing of 3D stacked memory.

3.2.10 Variable retention time stress

Contingent on the DRAM die layout, different memory cells can experience variable retention time failures before refresh. JEDEC spec calls for a 64 ms refresh interval. CPGC can test at intervals of 64ms, 128ms, etc. Each interval CPGC can repeat write, read scan patterns anywhere between 1 to 7 times with a variable pause of 64ms, 128ms in between. CPGC accomplishes this by adjusting memory controller timing. This can be used to simulate longer refresh intervals to emulate the accelerated decay at higher temperatures.

3.3 Gate Count

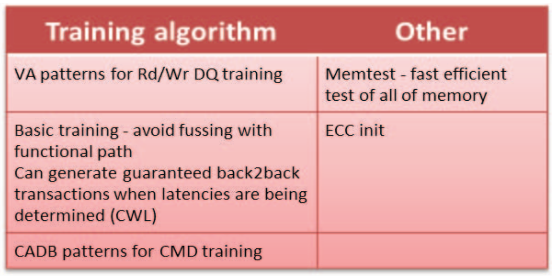
The overall gate count of CPGC IP is shown in Table 1 to occupy 44K micrometer squared. Compared to Intel Atom S1200 die area of 9.86mm x 10mm [22, 23], the BIST engine would occupy 0.045% of total area.

3.4 Power

Assuming a 13W Thermal Design Power (TDP) [22, 23], the CPGC IP has an total active power (as illustrated in Table 2) of 5.8mW. When the IP is not in use, it can be power gated off to draw almost no power except leakage.

3.5 Protocol

CPGC IP supports a simple request/acknowledge protocol for all write and read transactions. It can interface with memory controller and IO physical layers using standardized interface protocols.



Training algorithm	Other
VA patterns for Rd/Wr DQ training	Memtest - fast efficient test of all of memory
Basic training - avoid fussing with functional path Can generate guaranteed back2back transactions when latencies are being determined (CWL)	ECC init
CADB patterns for CMD training	

Table 5: CPGC architecture usage for memory initialization and training

4 CPGC USAGE AND IP REUSE MODEL

The key applications of this CPGC reusable BIST IP are in the following four areas: Validation, debug, memory initialization/training using the BIOS, and Test/HVM. For validation, CPGC can be used in System Margin Validation (SMV), Bench Device Validation (BDV), and Signal Integrity Validation (SIV) as shown in Table 3.

For SMV, a victim and aggressor (VA) pattern can be used on the Data pins (DQ) to measure the worst case margin. Similarly, Command Address Data Buffer (CADB) provides the pattern for margin measurements on Command and Address pins. Due to the bidirectional nature of the DDR bus, whenever the command switches from Read to Write or from Write to Read, the turnaround time margin can be validated using CPGC patterns. CPGC can purposely generate high or low bus utilization traffic. Even when the OS or the core is not powered up or functional, CPGC IP can be used as a very basic debug tool to test the DDR physical layer. Lastly, CPGC can quickly get margins of all DQ pins in a few seconds, and support full automation, which speeds up the validation time significantly compared to OS based software test patterns.

Bench Device Validation is accomplished on a test bench with the silicon device isolated. Similar VA patterns can induce noise on the DQ pins. CPGC offers precise control of patterns on the DQ pins. Since BDV boards typically do not support OS boot, CPGC IP is the perfect test and debug tool for BDV.

Signal Integrity Validation (SIV) is performed on either prototype or production boards with the focus on neighbor crosstalk (Xtalk) and inter symbol interference (ISI). Yet again, the VA pattern and the CADB pattern can provide a precise and stressful condition for SIV. SIV also requires explicit control of the pattern bits in time and space to create simultaneous switching of bits to excite even and odd mode resonance on the IO bus. CPGC IP also offers explicit trigger signals for scope capture and post processing of the waveforms.

CPGC can be used on a variety of debug scenarios, including power-on debug, margin debug, signal debug, and circuit debug as shown in Table 4. At power-on, CPGC offers explicit pattern control and stress testing capabilities to prove the bus is stable. This helps to isolate any issues away from the memory subsystem. CPGC IP requires very little enabling at power-on to become a useful tool.

To debug margin failure or low margin, CPGC offers precise turnaround control. CPGC can easily target any rank and channel to isolate the low margin rank/channel. CPGC can get margins of all byte lanes and individual lanes, while OS tools tend to hang at the first failure. To debug signal integrity, CPGC's explicit pattern control helps to generate ISI and Xtalk. The subseq_wait feature helps to control the gap between transactions.

In circuit debug, CPGC's control of read and write helps to isolate the transmitter and receiver of the memory physical layer independently. CPGC's precise control on rank targeting and turnarounds helps to catch any tri-state buffer issues. LFSR addressing provides semi-repeatable but randomized rank targeting, which helps to debug address transmitter circuits. As an independent data source from core functional traffic, CPGC helps to isolate circuit issues separate from memory controller or core issues.

In BIOS, CPGC IP supports memory training algorithms, helps to initialize all memory content at boot, and helps to complete Memtest, which is a quick test of all available memory. These usages are shown in Table 5. For training, the same VA pattern can be used to margin and center the strobe and reference voltage setting. Basic training can be done without using the functional path, simplifying the implementation. CPGC can generate back to back transactions, and provide CADB patterns for command training.

CPGC is also used in board high volume manufacturing test (HVM). In HVM, minimizing test time is the key to reducing product cost. CPGC provides an automated way to screen and margin the memory subsystem. As the memory capacity increases, the number of memory cell defects gains even greater significance. CPGC IP is fully equipped to test the memory cell through industry standard algorithms

System Margin Validation	Bench Device Validation	Signal Integrity Validation
Patterns: Victim Aggressor on DQ	DQ stress to induce noise	Patterns: VA on DQ
Patterns: CADB on CMD	no OS dependency (BDV boards sometimes don't boot)	Patterns: CADB on CMD
turnarounds: rank to rank	pattern control on DQ	Patterns: Explicit Control, Simultaneous switching, EVEN/ODD, etc..
high utilization of bus		turnaround events
no OS dependency		explicit control to help facilitate triggering, and post-processing of waveforms
get margins of all DQ's	Qualify IO	Qualify interconnect

Table 3: CPGC architecture usage for validation

Power-On	Margin Debug	Signal Debug	Circuit Debug
Explicit pattern control	Turnaround control	ISI: Explicit pattern control	Very controlled read / write control
Stress test to prove bus is stable (isolate from other issues)	Per rank & channel targetting is easy, isolation of margins per rank/channel	Xtalk: Mask signals from toggling	Controlled rank targetting & turnarounds
Very little enabling required (and MRC would have to pave the way)	Can get margins of all bytelanes & lanes (OS tools tend to hang at first failure)	subseq_wait - sort of controllable gaps between transactions	LFSR address - semi repeatable but randomized rank targetting
			Isolate between MC and core boundary

Table 4: CPGC architecture usage for debug

such as March C, March G, March SL, Butterfly, Galloping, and MATS+. As a programmable reusable BIST engine, CPGC IP also supports many complex memory test algorithm through the use of a sophisticated Memory Test Language (MEMTEL).

CPGC's software assisted repair technology for post package defects led to the currently defined JEDEC protocol standard where the DRAM manufactures provides finite reserved spare rows for memory cell failures introduced during the packaging and assembly of 3D stacked TSV packages. These failures can be introduced during the solder reflow and heating process, which cannot be predicted prior to packaging and assembly. The PPR flow can automatically repair failures detected at HVM in seconds. The CPGC offers huge test time savings compared to the operator intervention method, which takes multiple minutes to identify and repair failing bits.

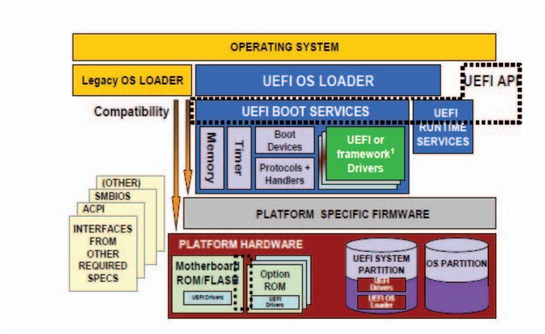
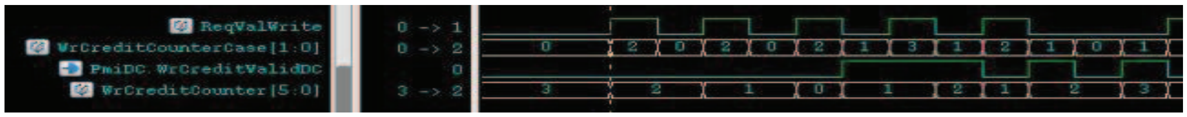


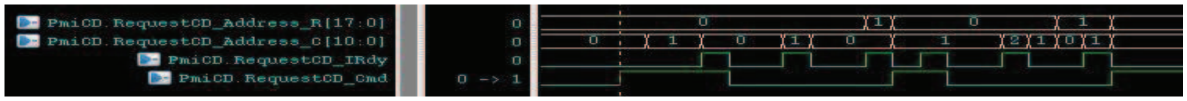
Figure 6: UEFI specification compliant BIOS that boots to an operating system

5 Software Architecture

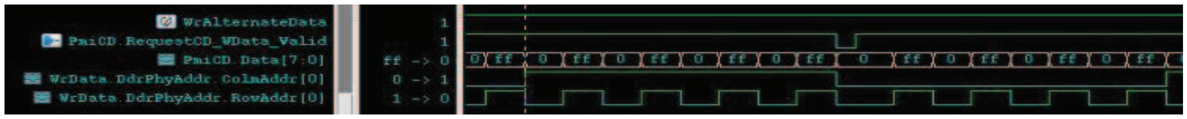
This integrated CPGC technology has built-in customizable flexibilities enabling the testing and diagnosis of the entire memory sub-system. In case of impending failure, it is possible to isolate the memory Field-Replicable-Unit (FRU) for further testing through CPGC IP. BIOS based UEFI architecture supports a middle-ware layer that can incorporate the CPGC API(s) at the boot-time as well as the run-time. At the boot time the CPGC-based test executes at the initialization of the memory reference code (MRC). At run-time, the code executes in the System management Interrupt (SMI), which is a virtualized space in system. The API(s) libraries are incorporated in both boot-time environments as well as run-time environments, while boot-time CPGC code is executes on a as needed basis. Execution of run-time code can cause system perturbations that can degrade the performance of the workload running on it. In order to execute the CPGC test through the SMI-based middle-ware, we have to prepare the system by executing these two conditions.



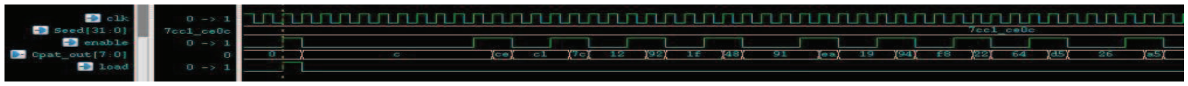
(a) Read/Write credits for standard IP interface


PmiCD.RequestCD_Address_R[17:0]
PmiCD.RequestCD_Address_C[10:0]
PmiCD.RequestCD_IRdy
PmiCD.RequestCD_Cmd

(b) Base offset memory stress algorithm


WrAlternateData
PmiCD.RequestCD_WData_Valid
PmiCD.Data[7:0]
WrData.DdrPhyAddr.ColmAddr[0]
WrData.DdrPhyAddr.RowAddr[0]

(c) Checker board memory data stress algorithm



(d) PRBS patterns stress patterns

Figure 8: CPGC architecture simulations

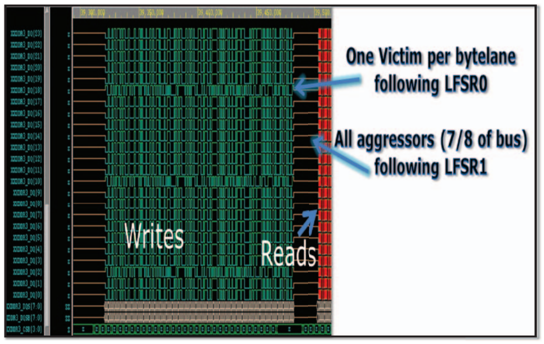

One Victim per bytelane following LFSR0
All aggressors (7/8 of bus) following LFSR1
Writes
Reads

Figure 7: CPGC architecture simulation of Victim and Aggressor Pattern

1. *Reason for executing* the test by continually analyzing the performance to predict an upcoming failure. Normally, this failure is detected through statistical modeling. This modeling scheme includes evaluating the memory soft-error patterns to predict hard-failure in future. Whenever a non-spurious pattern of soft-failures is detected, it is compared against a predetermined threshold to forecast hard failures. When this threshold is exceeded, the CPGC test is marked to be initiated either while the system is still functional or at the next boot.
2. *System Quiescing* - Once the CPGC test is marked (for a given memory sub-system), system resources are quieted by migrating the existing workload. One such OS assisted mechanism is called Memory Hot Removal that can remove the memory (marked for failure) dynamically. Once this memory is isolated, we can perform certain CPGC tests in a contained manner. Once the test is completed, memory can then be hot-added using OS assisted methods supported by ACPI specifications.

A firmware level approach to host CPGC tests and API(s) require building memory-map infrastructure for identifying memory addresses. Memory initialization code is modified to CPGC test code. System Management Interrupt (SMI) provides necessary application programming interfaces (API) to assist in constructing and configuring the memory CPGC assisted tests. Figure 6 illustrates the UEFI BOOT SERVICE layer that hosts the CPGC test modules. Similar modules are replicated into the SMI for run-time 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 intrinsic networking capability and is designed from the ground up to work with multi-processor (MP) systems. The UEFI standard is architected for Dynamic Modularity. This has opened up opportunity for collaboration on the technology that allows reducing development costs and time to market through easily integrating plugin modules from different partners and vendors.

1) *Software Assisted Dynamic Margining* In order to reduce the system guard-band, a CPGC assisted technique can identify the sufficient margins required to operate the DRAM rank with optimal performance/WATT. The goal in this case is to identify the excess margins and convert to voltage reduction. This checking is done in UEFI assisted agents in the SMI handler. These handlers are invoked opportunistically to test the availability of sufficient margins.

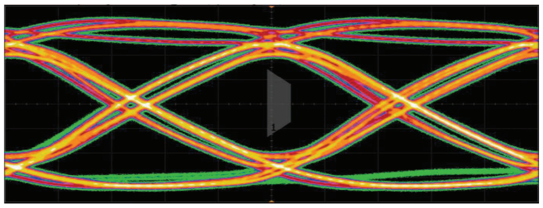


Figure 9: CPGC architecture silicon measurement of DDR IO eye diagram

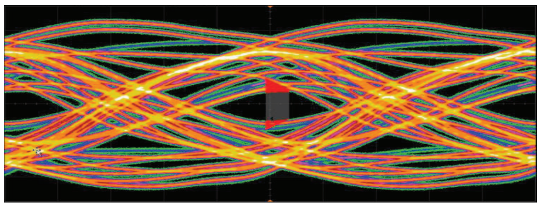
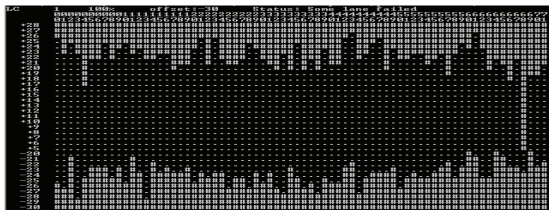
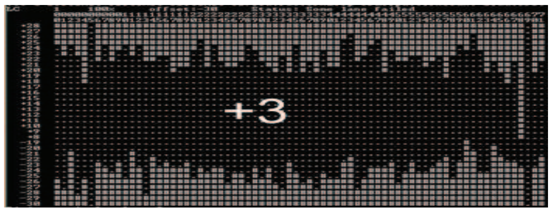


Figure 10: CPGC architecture silicon measurement of DDR IO eye diagram with VA stress pattern turned on.



```
LC      1    100%      offset:-30      Status: Some lane failed
        000000000011111111112222222222333333333344444444445555555555666666666677
        012345678901234567890123456789012345678901234567890123456789012345678901
```

Figure 11: CPGC BIOS measured eye height with all lanes enabled



```
LC      1    100%      offset:-30      Status: Some lane failed
```

Figure 12: CPGC BIOS measured eye height with bit 5 and ECC 5 disabled

2) *Software assisted CPGC test invocation* As mentioned earlier in this section, ad-hoc invocation of a CPGC based test can interfere with the performance requirements of the workload. Therefore we use statistical methods to evaluate the probability of failures based on memory related system events (Single-Bit-Errors etc.) to trigger this mechanism. These statistical methods seek to synthesize the temporal patterns of these failures using Markov chains. A system in this case can be modeled as a discrete Markov process to evaluate lifetime reliability and is described at any time to be in one of the states (S = Normal, Repair, Fail, etc). By employing the Markov modeling we can represent the failure detection, repair and maintenance process graphically. This provides us with a tool to evaluate system reliability measures and possible corrective measures to make it more reliable for the given cost. Details of this methodology are beyond the scope of this paper.

6 SIMULATION RESULT

Figure 8(a) illustrates CPGC IP interfacing mechanism with the memory controller via a standard interface using a request and grant-credit mechanism. As the initial three credits that were available for write are reduced to zero, the WrCreditValidDC returns a credit on each high pause, eventually restoring the credits to one, to two, and to three.

In order to find row-hammer problems, the CPGC IP generates a base address on the aggressor row, and generates an offset address on the victim row. Repeating base writes may result in row-hammer failures. As illustrated in Figure 8(b), RequestCD_Address_R stays on base row zero for a number of clocks and jumps to offset row one for a brief 1 or 2 clocks.

Checkerboard memory stress, as illustrated in Figure 8(c), writes all ones followed by all zeros to the DQ pins, which can be used to stress power supply on the memory IO as well as for memory cell retention.

LFSR is an important mechanism to generate pseudo random bit stream (PRBS). Illustrated in Figure 8(d), the 32 bit seed is fixed. As soon as enable is high, the Cpat_out, which generates 8-bits of output every clock starts to output random bit streams.

An aggressor/victim pattern is illustrated in Figure 7. Notice that while one lane acts as the victim, the remaining lanes act as aggressors, attacking the victim using a different pattern (LFSR1). Switching neighbors purposely introduces crosstalk on the victim bit, which will cause stress that leads to potential electrical failure.

7 CPGC RESULTS AND SILICON MEASUREMENTS

This section presents the CPGC results and silicon measurements.

CPGC was used in silicon validation to introduce the VA pattern rotation as shown in the simulation in Figure 7. Figure 9 demonstrates DDR margin without using VA pattern. Figure 10 shows DDR margin with aggressors turned on. As illustrated in Figure 9 compared to Figure 10, the CPGC controlled VA Pattern significantly reduced measured margin. This is a great tool for stressing the DDR IO link. The added VA rotation using CPGC reduced pattern reprogramming tests, and eliminated system reboots between the tests. This VA pattern rotation feature of the CPGC alone reduced the validation time by 3x.

CPGC is capable of debugging and isolating IO lanes bit by bit to see their contribution to crosstalk and result margin. Shown below in Figure 11 is the BIOS measured eye height for all 64 data lanes and 8 ECC lanes with all byte lanes enabled. Shown below in Figure 12 is the same eye height measurements with data lane 5 and ECC lane 5 disabled. Compared to Figure 11, there is 3 ticks of margin improvements. This shows that bit lane 5 alone contributes to 3 ticks of margin reduction.

These types of bit by bit isolation benefit root cause debug of board issues. Shown in Figure 14 is both a working board

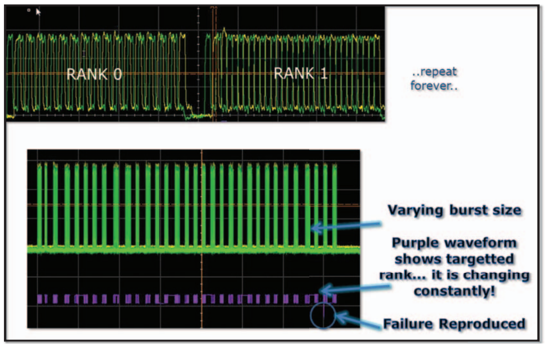


Figure 13: CPGC architecture silicon measurement of DDR IO initialization and training boot failure debug

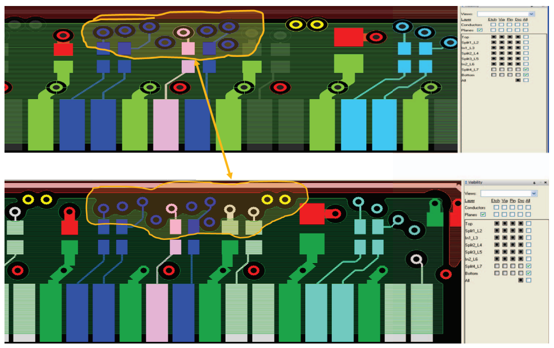


Figure 14: CPGC identified missing shielding on board layout

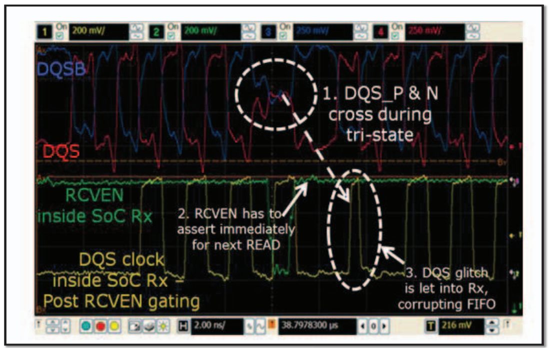
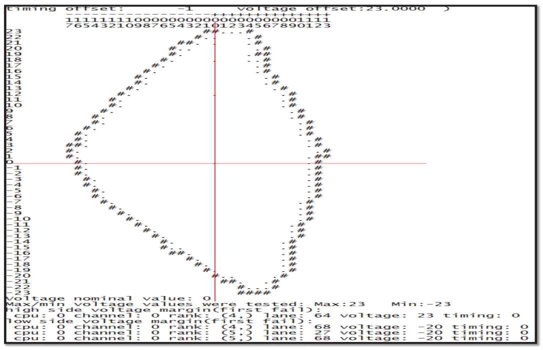


Figure 15: CPGC architecture silicon measurement of DDR IO DQS glitch corrupting Rx FIFO debug



```
timing offset:        -1      voltage offset:23.0000  )
       -----------------++++++++++++++
       1111111100000000000000000001111
       7654321098765432101234567890123
23                     ##...#
22                    #..   .#
21                  ##. .   .#
20                 #..       .#
19                 #.       .##
18                #.    .   ..#
17               #.          .#
16              #.           .#
15             #.             .#
14             #.             .#
13            #.               .#
12           #.         .       .#
11          #.                  .#
10          #.                  .#
9          #.                    .#
8         #.                     .#
7         #.                      .#
6        #.                       .#
5       #.                        .#
4       #.                         .#
3      ##.                         .#
2      #.                           .#
1      #.                          .##
0       #.              .          .#
-1      #.                         .#
-2      #.                         .#
-3       #.                        .#
-4       #.                        .#
-5        #.                       .#
-6        #.                       .#
-7         #.                     .#
-8          #.                    .#
-9           #.                   .#
-10           #.                  .#
-11            #.                .#
-12             #.               .#
-13              #.              .#
-14               #.            .#
-15               #..           .#
-16                ##.          .#
-17                  #.         .#
-18                   #.        .#
-19                    #.       .#
-20                     #..    .#
-21                      ##.  .#
-22                        #...#
-23                        #####
Voltage nominal value: 0
Max/min voltage values were tested: Max:23    Min:-23
high side voltage margin(first fail):
 cpu: 0 channel: 0 rank: (4,) lane: 64 voltage: 23 timing: 0
low side voltage margin(first fail):
 cpu: 0 channel: 0 rank: (4,) lane: 68 voltage: -20 timing: 0
 cpu: 0 channel: 0 rank: (5,) lane: 27 voltage: -20 timing: 0
 cpu: 0 channel: 0 rank: (5,) lane: 68 voltage: -20 timing: 0
```

Figure 16: CPGC architecture silicon measurement of DDR IO initialization and training BIOS mapped eye

that has low cross (top), where the victim bit in white is shielded from the 7 aggressors in blue and the non working board, where the shielding is missing between the victim in white and aggressors in blue.

On an Intel SOC (early stepping), CPGC facilitated to debug and isolate a memory controller speed-path. CPGC was able to isolate the bug because it provided an alternative path to the DDR physical layer. When the debugger noticed that the functional memory subsystem did not work at certain frequencies, but using CPGC to bypass the functional memory controller, the DDR physical layer was working at the specified frequency. The debugger was able to quickly focus the attention on the memory controller and find the speed-path.

7.1 Debugging boot failure using CPGC LFSR addressing

CPGC was proven to be extremely useful in debugging memory initialization and boot failures. At first, the boot failure was not deterministic, which was hard to reproduce, and would have taken a lot of debug time. Illustrated near the top of the Figure 13, typical memory traffic writes to Rank 0 followed by Rank 1 when not using CPGC.

Illustrated near the bottom of Figure 13, by varying burst size using CPGC and LFSR addressing, the targeted rank ends up changing constantly (purple waveform), which led to a quick failure reproduction. This helped to reduce boot failure debug time down to a few minutes.

Illustrated in Figure 15, the DQS_P and DQS_N pin crossed during tri-state. The Receive Enable (RCVEN) inside SOC Rx was asserted immediately for the next READ command. The crossing triggered the DQS clock inside the SOC Rx post RCVEN to glitch, which corrupts the Rx FIFO. Once again, CPGC provides precise address and data pattern control, which enables the debugger to fine tune the pattern and dial into the glitch quickly. This again saved several days of debug labor compared to that without the CPGC reusable BIST engine.

In DDR initialization and training, CPGC's CADB pattern and deselect feature provided an ISI and crosstalk rich pattern during the non-functional select de-asserted state. This helped the BIOS and firmware to quickly center the command and address pins and ensure these pins are protected against functional ISI noise.

Figure 16 illustrates the measured eye from BIOS based memory training and eye centering using CPGC based LFSR pattern. LFSR provides a quick pseudo random pattern. The pattern does not have to be the worst case stress, but how quickly the pattern can be applied is important from a BIOS boot time point of view. CPGC delivers this speedy pattern, which is ideal for memory training. The vertical axis represents the voltage measurements, and the horizontal axis represents the timing measurements. Both voltage and timing offset can be applied to measure the boundary of failure. A contour of the eye is mapped out by the BIOS before the voltage and timing is centered in the middle of this mapped eye.

CPGC can also be used in conjunction with circuit tuning controls such as reference voltage and phase interpolators to intentionally offset the data strobe to measure I/O jitter and DDR skew.

8 Summary

To summarize, the CPGC architecture offers significant time savings for validation, test, and debug for DDR, TSV, and 3DS WIO as shown in Table 6. The victim and aggressor pattern rotation automation of the CPGC reusable BIST engine alone decreased validation time by 3x. The CPGC software assisted repair technology reduced test time from minutes down to seconds. The CPGC reusable BIST engine enabled debug of various memory and IO failures including memory IO not booting. The CPGC reusable BIST engine reproduced indeterministic boot failures in minutes, and consistently reproduced bugs that are hard to find, which enabled a debug time savings of 5X. An overview of the CPGC reusable BIST

	Savings	Comments
Validation	3X	VA pattern rotation automates victim lane selection
Test	Minutes to seconds	Auto-Repair reduced loading and unloading of failure signature
Debug	5X	Reproduced in-deterministic boot failures in minutes

Table 6: CPGC architecture savings to Validation, Test, and debug Time

engine architecture on Intel 14nm SOC was presented in this paper. The CPGC BIST engine was designed as a reusable IP that supports standard interfaces to enable quick turn-key SOC solution. CPGC is also used in memory training and initialization, which makes it a critical part of Intel SOC.

Acknowledgments

The authors would like to acknowledge Thilak K Poriyani House Raju for design, simulation, and verification of the CPGC IP. The authors would like to thank Adam Taylor for his knowledge and data contribution to this paper. The authors would like to acknowledge Anne Meixner for reviews, technical advice, and encouragement.

9 References

[1] S. Sarangi, S. Narayanasamy, B. Carneal, A. Tiwari, B. Calder, and J. Torrellas, "Patching Processor Design Errors with Programmable Hardware," Micro, IEEE, vol. 27, pp. 12-25, 2007.

[2] R. Venkatesh, S. Kumar, J. Philip, and S. Shukla, "A fault modeling technique to test memory BIST algorithms," in Memory Technology, Design and Testing, 2002. (MTDT 2002). Proceedings of the 2002 IEEE International Workshop on, 2002, pp. 109-116.

[3] Y. Jen-Chieh, W. Chi-Feng, C. Kuo-Liang, C. Yung-Fa, H. Chih-Tsun, and W. Cheng-Wen, "Flash memory built-in self-test using March-like algorithms," in Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on, 2002, pp. 137-141.

[4] A. J. Van de Goor, S. Hamdioui, and H. Kukner, "Generic, orthogonal and low-cost March Element based memory BIST," in Test Conference (ITC), 2011 IEEE International, 2011, pp. 1-10.

[5] S. Sheng-Chih, H. Hung-Ming, C. Yi-Wei, and L. Kuen-Jong, "A high speed BIST architecture for DDR-SDRAM testing," in Memory Technology, Design, and Testing, 2005. MTDT 2005. 2005 IEEE International Workshop on, 2005, pp. 52-57.

[6] G. Mrugalski, A. Pogiel, N. Mukherjee, J. Rajski, J. Tyszer, and P. Urbanek, "Fault Diagnosis in Memory BIST Environment with Non-march Tests," in Test Symposium (ATS), 2011 20th Asian, 2011, pp. 419-424.

[7] 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 Test Symposium (ATS), 2013 22nd Asian, 2013, pp. 227-232.

[8] P. Bernardi, M. Grosso, M. S. Reorda, and Y. Zhang, "A programmable BIST for DRAM testing and diagnosis," in Test Conference (ITC), 2010 IEEE International, 2010, pp. 1-10.

[9] Y. Dongkyu, K. Taehyung, and P. Sungju, "A microcode-based memory BIST implementing modified march algorithm," in Test Symposium, 2001. Proceedings. 10th Asian, 2001, pp. 391-395.

[10] B. Schroeder, E. Pinheiro, and W. Weber. "DRAM Errors in the Wild: A Large-Scale Field Study", SIGMETRICS/Performance '09, June 15-19, 2009, Seattle, WA, USA.

[11] D. Blankenbeckler, A. Norman, M. Shepherd. "Comparison of off-chip interconnect validation to field failures", VALID, 2011, Barcelona, Spain.

[12] Nejedlo, J.J., "IBIST (interconnect built-in-self-test) architecture and methodology for PCI Express," Test Conference, 2003. Proceedings. ITC 2003

[13] Nejedlo, J.J., "TRIBuTE board and platform test methodology," Test Conference, 2003. Proceedings. ITC 2003

[14] Nejedlo, J.J., "IBISTTM (interconnect built-in self-test) architecture and methodology for pci express: intel's next-generation test and validation methodology for performance IO," Test Conference, 2003

[15] Nejedlo, J.; Khanna, R., "Intel IBIST, the full vision realized," Test Conference, 2009. ITC 2009. International , vol., no., pp.1,11, 1-6 Nov.

[16] http://www.memcon.com/pdfs/proceedings2013/track1/Tuning_DDR4_for_Power_and_Performance.pdf

[17] http://www.memcon.com/pdfs/proceedings2013/track1/Tuning_DDR4_for_Power_and_Performance.pdf

[18] http://www.google.com/patents/WO2014004748A1?cl=en

[19] http://www.youtube.com/watch?v=i3-gQSnBcdo

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

[21] Querbach, B. "Comparison of Hardware Based and Software Based Stress Testing of Memory Io Interface." IEEE MWSCAS 2013

[22] "'http://en.wikipedia.org/wiki/Transistor_count"'

[23] "'http://www.extremetech.com/computing/152979-intel-announces-full-set-of-new-atom-and-xeon-server-processors-to-fend-off-arm"'