# Instituto Tecnológico
# y de Estudios Superiores de Occidente

Recognition of official validity of higher education studies as set forth in ministerial agreement 15018, published in the Official Journal of the Federation on November 29, 1976.

Department of Electronics, Systems and Informatics

**Master's Degree in Electronic Design**




ITESO, Jesuit University of Guadalajara


## POST-SILICON FUNCTIONAL VALIDATION OF A DDR5 MEMORY CONTROLLER

**DEGREE-EARNING PROJECT** that, in order to earn the **DEGREE** of
**MASTER IN INDUSTRIAL ELECTRONIC**


Is submitted by: **FEDERICO J. HERNÁNDEZ REYES**

Tutor: **Dr. Omar Longoria**



Tlaquepaque, Jalisco. October de 2023.

# Acknowledgements

I want to thank God for allowing me to complete this project walking with me along all moments of my life being there and supporting me, and for the precious gift of life.

I want to acknowledge this work to my wife who encouraged me to come back and complete the process, and to my daughters Tania and Nicole, and my son Juan Pablo who complement my life making it great.

I want to acknowledge this work to my father who through his life and example taught me to persist and have faith, and to my mother who always believed in me and for her tender maternal love.

I want to heartly thank my tutor Dr. Omar Longoria for guiding me through the writing of this work, and to Intel for the support and opportunity to experience 17 years of post-silicon validation working through many successful server Xeon CPUs and Chipsets.

# Summary

*A case study of the application of a post-silicon functional validation methodology to the validation of a DDR5 memory controller is presented. The work is in the context of the post-silicon validation of the Next Generation Intel® Xeon Server CPU.*

*The post-silicon functional validation methodology is presented not as a novel approach to validation but to set the context for the description of how the different phases of the methodology were applied to the functional validation of a DDR5 memory controller which features technological advances in DDR5 frequencies up to 6400MT/s and Multiplexed Combined Rank DIMMs that achieve 30-40% more bandwidth than regular RDIMMs, new RAS characteristics and performance-driven architectural improvements. It also serves as guide for future applications of the methodology on the validation of different technologies or IPs.*

*The results of the validation of the DDR5 memory controller are presented and discussed for each of the phases of the methodology starting with the validation strategy centered in defining the HW and SW configurations aimed to cover all the functional features of the memory controller; the test content development and the debug strategies are also described. The utilization of Virtual Platforms and Emulation systems as a vehicle to verify the test content was ready and met its intent and the execution results in this stage are covered. Then, the results of the execution of the test plan in the actual post-silicon phases of power-on and volume validation execution are considered under the light of the number of silicon, architectural, BIOS and FW bugs found.*

*Finally, the evaluation of the project using number of bugs and time to market metrics is discussed. Areas of improvement for the post-silicon validation execution are identified and proposed as learnings to the next project.*

*Keywords: Post-Silicon, Functional Validation, Memory Controller, DDR5.*

.

# Content

# Introduction

With the increased complexity of general-purpose Central Processing Units (CPUs) it is not possible today to rely exclusively in good design techniques, and pre-silicon simulations and emulations to release a design to mass production. It is necessary to verify the design under real conditions before its mass production.

Post-Silicon Validation (PSiV) is the last stage of the design and development process of an Integrated Circuit (IC). Its goal is to ensure that the physical implementation of a design under real operating conditions conforms to the Architectural definition, the design specifications, and applicable industry standards. It also has the critical mission of avoiding a critical silicon bug escapes from the Lab to the customer.

A case study of the application of a post-silicon functional validation methodology to the validation of a DDR5 memory controller is presented. The work is in the context of the post-silicon validation of the DDR5 memory controller of the Next Generation Intel® Xeon Server CPU.

Chapter 1 – Problem Statement, describes the need of doing post-silicon validation and the criticality to find the bug that might cause a product recall. It also describes the motivation to document the case study of validating a DDR5 memory controller, and the general goal of the project.

Chapter 2 – Materials and Methods, provides a clear definition of all the phases of a post-silicon validation methodology covering from the definition of the validation strategy to the volume validation execution and debug of all failures.

Chapter 3 – PSiV methodology applied to a DDR5 Memory controller, documents the outcomes of each phase of the validation methodology applied to the post-silicon functional validation of the memory controller in the Next Generation Intel® Xeon CPU.

Chapter 4 – Discussion, reviews and analyzes the results presented in Chapter 3 from the perspective of the program execution, describing the main problems experienced during the post-silicon validation process.

# 1. Problem Statement

With the increased complexity of general-purpose Central Processing Units (CPUs) it is not possible today to rely exclusively in good design techniques, and pre-silicon simulations and emulations to deliver a design to production. It is necessary to verify the design under real conditions before its mass production.

Post-Silicon Validation (PSiV) is the last stage of the design and development process of an Integrated Circuit (IC). Its goal is to ensure that the physical implementation of a design under real operating conditions conforms to the Architectural definition, the design specifications, and applicable industry standards. It also has the critical mission of avoiding a critical silicon bug escapes from the Lab to the customer. It is well known that the cost of a bug grows exponentially as we go from pre-silicon to post-silicon and to the customer. As can be seen in Figure 1-1 the cost of a bug found by a customer can be >1000 times the cost of the bug if it had been caught during pre-silicon verification (Stecyk, 2016).

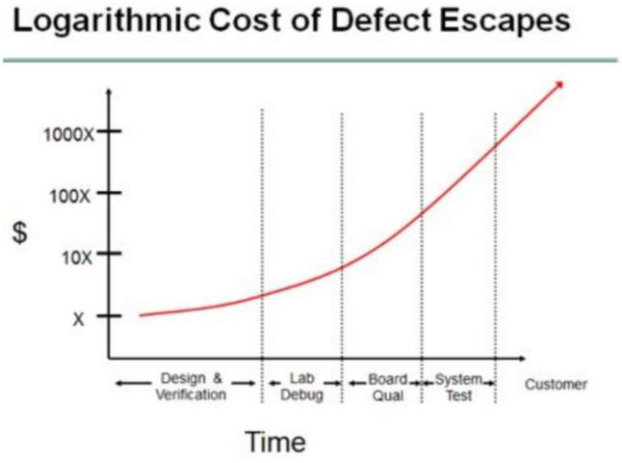


*Figure 1-1 Cost of a bug. Source: Intrinsix (Stecyk, 2016).*

As an example, Intel lost $1B USD back in 2011, $300M in direct revenue plus $700M in repair costs, due to a bug that caused the 3Gbps SATA controller of a chipset to degrade and lost connectivity to the SATA drive (Takashi, 2011).

PSiV is performed by a group of disciplines that cover all different aspects of the design, including Functional Validation (FV), Electrical Validation (EV), Power and Performance

Validation (PPV), Compatibility or Platform Validation (PV), and Volume Manufacturing Validation (VMV).

FV is focused on all functional features of the design, from power up and down sequences, loading and executing instructions and moving data from and to memory, connecting to an external Peripheral Component Interconnect Express (PCIe) or Compute Express Link (CXL) endpoint, or connect to other instances of the design.

EV is focused on the electrical characteristics of the design, from low-frequency signals and interfaces such as General-Purpose Input/Output pins (GPIOs), and reference clocks (100MHz), to high-speed interfaces such as PCIe, CX, or Dual Data Rate 5 (DDR5) that use electrical signaling ranging from 2800MHz in the case of DDR5 running at 5600MT/s (Mega Transfers per second) to 32GHz in the case of PCIe and CXL running at 32Gbps.

PPV is focused on the power consumption of the design under different workloads ensuring satisfactory performance indicators are maintained.

PV is focused on assuring that the design and the companion software stack (firmware, BIOS, OS device drivers) work with commercial devices and applications.

VMV is aimed to find all possible manufacturing defects by running a subset of FV test cases and content in a supervised set of cells that are used to test every single physical instance of the design (e.g., every single CPUs)

## 1.1. Motivation and Scope

In a personal computer as well as in a server, the CPU requires low latency, and always reliable access to instructions and data that are stored in the system memory. It is the memory controller's responsibility to provide this high bandwidth low latency and reliable access to the system memory.

Memory technology has evolved with the computer industry and achieved velocities in the range of 6400MT/s and capacities that go from 8GB to 256GB per memory module (SkHynix, 2023) (JEDEC, 2021).

Previous generation of memory controllers implemented DDR5 up to 4800MT/s in 1DPC (one DIMM per Channel) but fewer instances – 8, per CPU (Intel, 2023). The next generation is

designed to support DDR5 up to 6400MT/s in a single channel but with multiple instances -12 per CPU maintaining the support of DDR5 Sub-Channel architecture.

Every new generation of memory controllers adds a set of differentiating features over the competition (AMD, 2023) and improves performance over previous generations (Intel, 2023). Examples are changes like addition of Error Correction Code (ECC) protection in internal queues, multiple encryption keys, DDR IO circuit improvements to reach 6400MT/s or even higher speeds, and runtime soft per part repair, among others.

Pre-Silicon verification (PreSiV), although provides a high level of controllability on input parameters and stimulus to all internal Finite State Machines (FSMs), is not sufficient as it is always subjected to the quality of the test benches, checkers, and Bus Functional Models (BFMs). In System validation is still a must to ensure all new features supported by the new memory controller are tested under real system conditions to prove all features can be productized. In addition, multiple vendors of Dual In-line Memory Modules (DIMMs) are used to prove the design can be used at the required performance specifications in all Plan of Record (POR) customer configurations and workloads.

This work focuses on the post-silicon functional validation (also known as System Validation) of a new generation DDR5 memory controller, covering the different phases of the PSiV methodology for the A0 stepping of the design. The work is situated in the context of the post-silicon validation of the Next Generation Intel® Xeon Server.

## 1.2. General Objective

The general objective of this work is to review the application of a post-silicon functional validation methodology to a DDR5 Memory Controller exposing with sufficient details the results on each of the phases of the methodology, and to discuss the mayor problems and areas of improvement that can serve as learnings to upcoming post-silicon functional validation projects.

# 2. Materials and Methods

## 2.1. Phases of a PSiV project

A PSiV project is composed of several phases spanning two major stages of the product life cycle. These are a pre-silicon readiness and planning phase, and an execution phase. Figure 2-1 shows the design and post-silicon validation swim lanes running in parallel. The pre-silicon readiness and planning phase of the PSiV normally starts a little bit later after the architectural definition of the product has gained a minimum level of maturity and advances along with the micro-architecture definition of the various Intellectual Property (IP) blocks of the design. The PSiV execution phase starts with the first power on of the first samples of the design coming from the manufacturing plant and ends with the Product Release Qualification (PRQ) of the design. The project can have multiple planned major silicon releases; releases that require full resynthesis. Every new major version of the design has its own power-on and volume validation cycle. Minor versions or metal layer versions can also be created to fix design defects that do not require full resynthesis and are of interest to the validation and design teams – find bugs behind bugs or unblock functionality in the product. The minor stepping samples are seamlessly deployed to the validation lab without major implications. A sustaining phase starts after PRQ and lasts until the product line is discontinued; however, it is not part of the PSiV project.

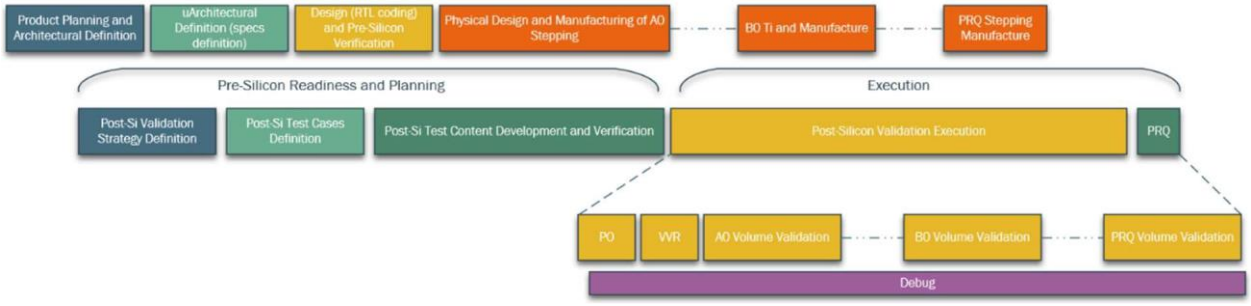


*Figure 2-1 Post-Silicon Validation Project Phases*

The following sections describe each of the PSiV pre-silicon readiness and planning, and the execution phases of the project.

# 2. MATERIALS AND METHODS

### 2.1.1 Validation strategy definition

The validation strategy is defined considering the program goals for time to market, cost, the highly visible customer differentiating features, and compliance with industry standards. During this phase the overall validation framework is defined. This includes the generation of validation platform requirements, Basic Input Output System (BIOS) requirements, validation software requirements, debugging requirements, head count, and staff training requirements.

### 2.1.2 Test case definition

In this phase all Test Cases are designed. Figure 2-2 shows the process followed by the validation engineers to design the test cases. The validation engineers read and analyze applicable specifications starting from the product architectural specifications, following with the uArch and design specifications, and any applicable industry standards; the test cases are designed to cover all functions of the product and demonstrate they work according to the specifications. During this process, the validation engineers provide feedback to the documentation and request clarification from product architects and design engineers.

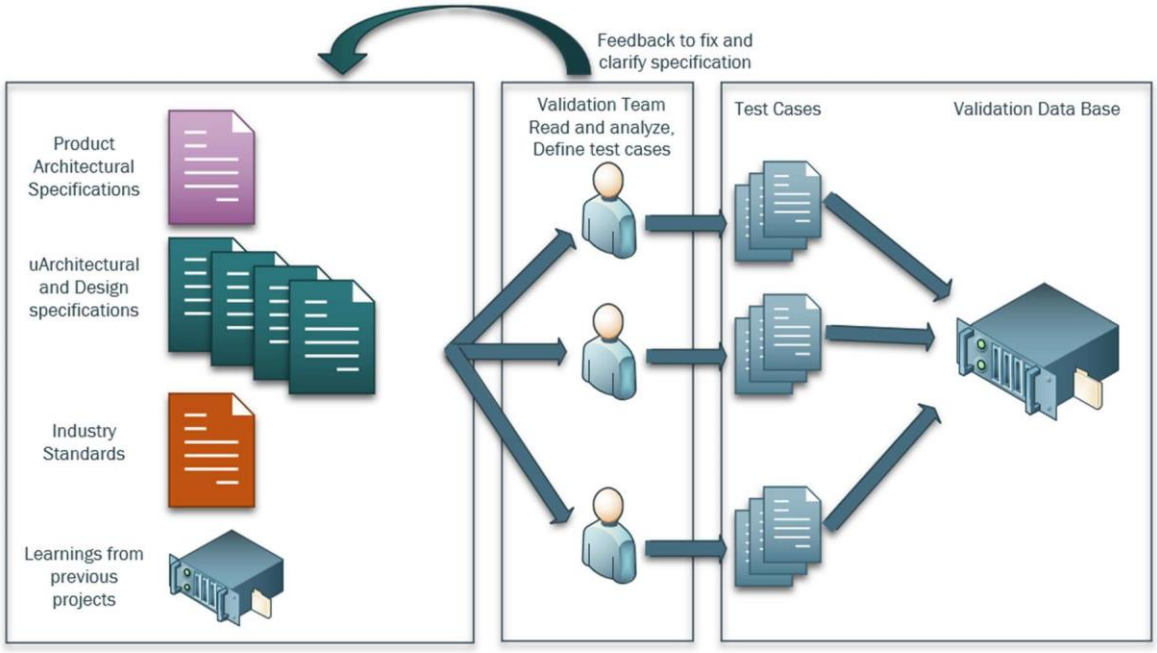


*Figure 2-2Test Case definition process*

The test cases can be classified based on their intent. An example of such classification is as follows:

a) Positive coverage

These are tests designed to ensure intended product functions are met. Positive coverage tests can further be classified into:

1. Feature enabling tests (also known as silicon checkout). These tests are basic in nature and are aimed to enable and demonstrate basic silicon functionality. These tests are usually executed during Power On (PO). Examples are to validate that the design was able to get out of reset and is ready to be configured by BIOS as its configuration registers are accessible now from the CPU.
2. Stress tests. These tests are designed to demonstrate that silicon functions can be performed under high traffic patterns and conditions that exercise the design in its limits. Examples are ensuring internal queues are full for long periods of time, error correction logic Finite State Machine (FSM) correctly navigates all states to correct errors and testing under high and low temperature and frequency conditions.
3. Cross products. These tests are designed to validate that functional features can be combined under stress conditions. Examples are validating that power saving features can work along with stress traffic and/or error detection and correction features.

b) Negative validation

These tests ensure the design gracefully handles inputs and conditions that are outside of normal system configuration. Examples are limiting credits between interfaces, attempting to exercise a functional feature that is disabled by fuse, and testing at voltage levels that are slightly outside of the design specification.

### 2.1.3 Test content creation

During this phase the test or validation content is written using the plan of record validation infrastructure which includes commercial or synthetic operative systems and drivers, scripting languages such as Python or bash, and the automation framework defined in the validation strategy.

## 2. MATERIALS AND METHODS

Figure 2-3 shows the basic process of content creation and the SW stack on which the content will be executed. The validation engineers write content for each of the test cases designed for each functional feature; one or more validation engineers can work on one or multiple functional features depending on the design complexity and the number of test cases designed for each. The validation content can take the form of Python scripts that are written to directly manipulate IP configuration registers, read status bits, and use IP logic designed to support validation; this logic is known as DFx (Design for x, where x is: debug, validation, manufacturability, etc.). The test content can also be a set of commercial or synthetic applications that run natively on the operative system and exercise the different IPs through traffic generation (transactions to/from memory, PCI-Express and CXL endpoints, accelerators) and register accesses.

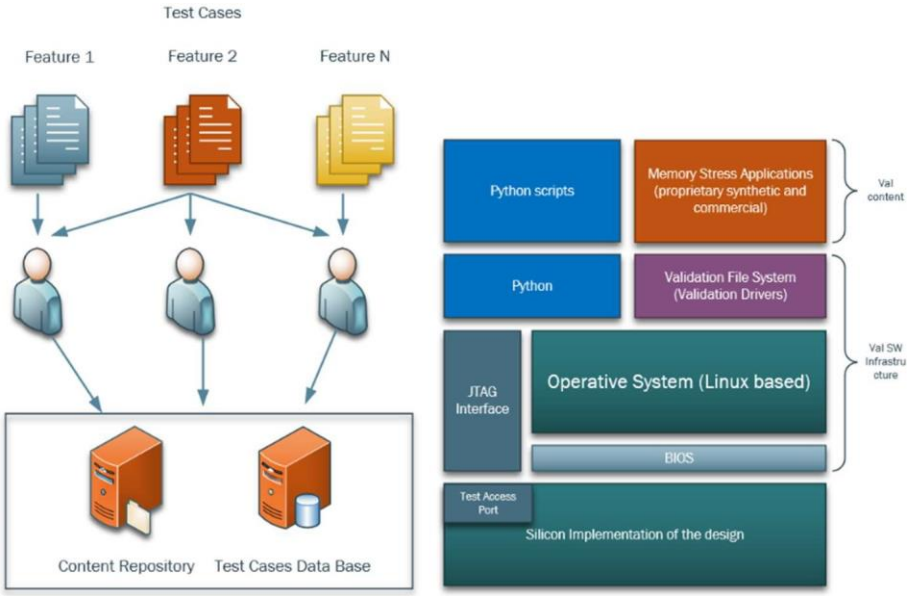


*Figure 2-3 Content creation using the plan of record validation SW stack.*

The validation SW stack is composed of the BIOS, a Linux based validation OS, a validation file system, and the validation content. The validation file system exposes through proprietary device drives the several IP's configuration and status registers to the validation content. This validation file system allows a Python virtual machine to gain access to the configuration and status registers when running on the validation OS.

A special set of Python scripts is called harassers. These scripts are intended to create extra stressful conditions for the IP under validation, as its name indicates, harassing a particular feature

while running stress traffic through the IP, for example injecting Error Correction Code (ECC) errors to memory to stress the memory controller's error correction logic.

The Python scripts can also run on a separate machine that has access to the Test Access Port (TAP) of the design using a JTAG interface. This Python scripts are mostly used to debug the design.

### 2.1.4 Content verification in pre-silicon models

During this phase, and in an iterative mode, the test content is verified for intent using pre-silicon models that can be virtual platforms, emulators running versions of the Register Transfer Level (RTL) models that will later become the silicon implementation, or previous generation silicon platforms that allow running portions of the content in a more realistic environment. Figure 2-4 shows a high-level view of the three main pre-silicon platforms. The Virtual Platform (VP) is a SW implementation of a behavioral representation of the design and can be implemented as a white box, black box, or a combination of both depending on the complexity of the design and the IP of interest. The Emulator can be a single Field Programmable Gate Array (FPGA) development board, or a farm of FPGAs that run a synthetized version of the RTL of the design. The Previous Program Platform (PPP) is a physical validation (mother) board using a previous product sample and HW collaterals (memory, test cards, Intel® In-Target Probe) and can run all or portions of the validation SW infrastructure (BIOS, OS, device drivers and applications).

The test content written for all new design functional features is first run in a VP to expose and fix all possible coding and algorithm bugs, then it is run in emulation platforms to verify intent and eventually find RTL bugs. The test content written or ported from the previous program validation content suite is generally run in the PPP where speed is significantly higher than in the other two platforms.

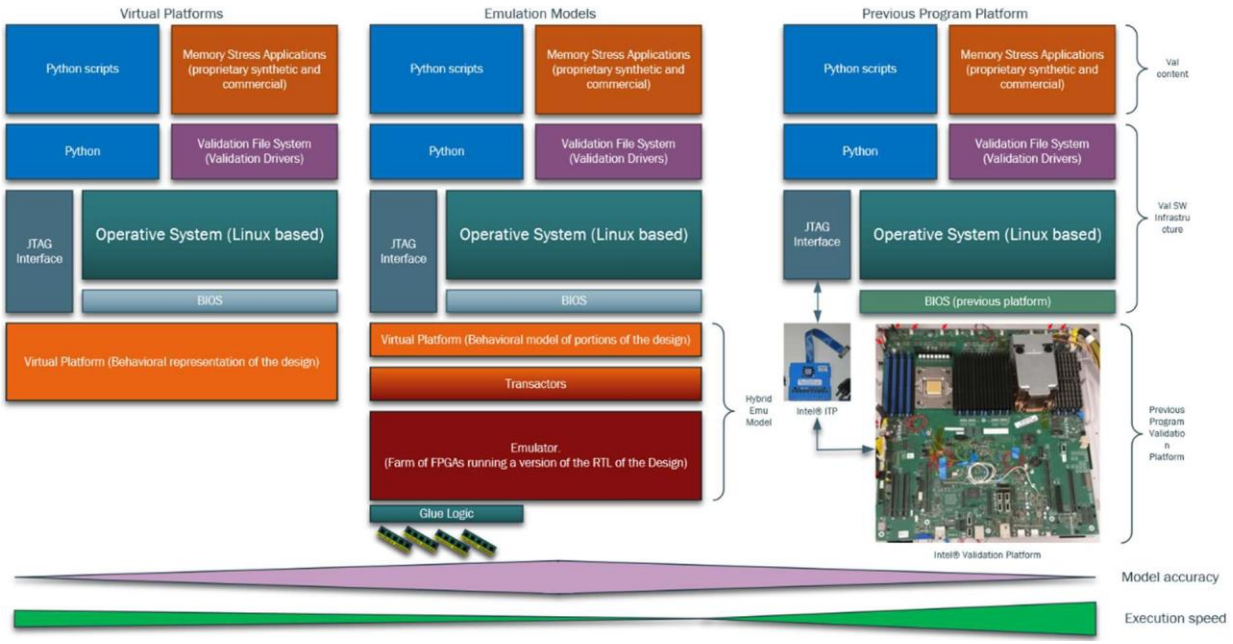


*Figure 2-4 Pre-Silicon validation platforms for test content verification*

### 2.1.5 Power-On (PO)

This is the phase in which the very first silicon samples of the product are powered up in a controlled environment. The power on activities and silicon checkout plans are defined to achieve the main objective of demonstrating that all critical features are functional, to enable validation of the product, and to enable debug features, equipment, and interfaces. It is important to mention that this phase is conducted by a large group of engineers, usually 100+ people working together in the lab to power up the silicon sample, boot to BIOS and the OS in the shortest time possible as this message reaches customers and build up confidence on the success of the project. This phase usually takes from 2 to 6 weeks depending on the complexity of the product.

Before the PO, there is a PO planning phase where all the validation teams define which tests and activities will be executed during the PO, who will be part of the first boot team, and what are the objectives of each PO sub-phase. **Phase 0** is aimed to get the design out of reset after applying power; entry to this phase requires that fuse recipes (hardcoded configuration settings applied to the IC) and critical debug scripts are tested first in emulation models; the exit of this phase is declared when the critical IPs of the design are effectively out of reset, for example in a

CPU the bootstrap core starts fetching BIOS instructions from the reset vector. **Phase 1** is focused on getting to the first BIOS boot, which in the case of a CPU encompasses fetching 1st instruction, bare minimal system configuration to enable memory training, PCI-express link training, and any other relevant IO are configured to allow the system memory map configuration including Memory Map IO (MMIO), and finally boot to BIOS shell. After BIOS has reached the BIOS shell the next step is to achieve the first OS boot. A predefined set of commercial OSes is usually attempted first, though any OS validation version sometimes is booted first as it provides a higher level of controllability and debuggability on the system. **Phase 2** is aimed to complete feature enabling and allow validation teams to do silicon checkout and complete debug tools enabling and debug stations setups. During Phase 2 the simplest silicon checkout test cases are executed first and as basic features are enabled, the validation teams continue with more complex test cases.

The PO phase is completed when all validation teams have enabled all critical features and all major problems are understood and resolved to enable the volume validation phase.

### 2.1.6 Volume Validation Readiness (VVR)

This phase is used to scale volume validation up. The different HW and SW configurations defined in the validation strategy under which the product is going to be tested are enabled by the validation teams. Depending on the size and complexity of the project, up to 100 to 150 validation stations are built and deployed to the VV lab ready to be used and controlled by the automation server. The validation team focuses on achieving automation environment robustness and validation platform stability to increase execution efficiency and reduce false failures from the validation environment.

The automation environment is said to be robust when: it can consistently cycle through the automation flow; detect and recover from platform and silicon errors; update SW collaterals; gather validation logs and debug data from the validation station; and finally uploads them to a file server.

A validation platform is said to be stable when it can be power cycled and rebooted consistently and reach OS 100% of the time.

The VVR phase not necessarily has to wait for the PO Exit declaration to start. The VV Team (VVT) which is composed by validation automation engineers also participates in the PO, although in the last phases of it, when the PO team has defined a recipe to boot the silicon to OS. The VVT starts testing the automation flow with the new silicon running dummy tests aimed to ensure the automation flow works with the new product, all platform checkers correctly detect system misconfigurations (e.g., wrong memory populated for a specific validation configuration) and correctly detect errors logged in the status registers of the silicon design. Once the automation flow is verified and the validation platforms are deployed to the VV Lab, the VV stage is started.

### 2.1.7 Volume Validation (VV)

This phase is where all silicon features are tested across multiple instances of the product in an automated environment. The validation coverage is gained through multiple hours of execution across the different HW and SW configurations. Pass tests are analyzed to verify coverage and intent of the test were met and to avoid false passes. Failures are debugged to understand the root cause and identify all design-related bugs, find workarounds, and provide feedback to the Design team to get fixes on the next stepping of the product. As is seen in Figure 2-1, there is a VV phase or VV test cycle for each of the planned steppings. All the test cases that are designed for VV are prioritized in such a way that all basic functional features are tested first – these are Priority 1 (P1) test cases, followed by stress tests and cross products – these are Priority 2 (P2), and finishing with long hour runs – these are Priority 3 (P3) test cases.

The VV environment is shared pool of 100 to 150 validation systems available for all validation teams to execute their tests. The VV Execution Capacity (EC) in the number of hours is determined as follows:

$$EC = NVS\,(\frac{24hrs}{day})\,(\frac{7days}{week})\,NW(ef)$$

*Equation 1 Execution Capacity*

Where *NVS* is the number of validation systems, *NW* is the number of weeks of execution for the VV cycle, and *ef* is an efficiency factor based on the perceived complexity and healthiness of the validation platform, expected up-time, and previous project experience.

The VV execution capacity is then distributed across validation teams considering factors such as test count, per test execution time, IP complexity and amount of change from the previous program. The VV execution capacity is not evenly distributed among validation teams. The VVT ensures that the execution capacity is sufficient to meet the needs of all validation teams, and tradeoffs and negotiations are made with the validation teams to do the correct planning under program constraints, as platform count and cost are fixed for the program. This is an exercise of bottom-up and up-down communication and planning.

Figure 2-5 shows an example of a volume validation environment in which the execution is controlled by an automation server that controls every validation station individually. The volume validation environment is said to be balanced when there is the same count of validation stations per HW config, and unbalanced when the validation stations are not evenly distributed across HW configs. Balanced vs. Unbalanced validation stations distribution is usually determined when the Validation Strategy is defined as it impacts HW collaterals inventory and project cost, however flexibility might be exercised as execution takes place and some rebalancing is needed to complete execution of lagged test cases. The test cases in the form of test lines are uploaded to the server automation. A test line is composed of pre-tests, test, and post-test steps. Every test line carries information about the HW and SW configuration it is intended to run on, the owner validation team, test time, overrides to default SW configuration in the form of BIOS overrides, OS kernel overrides, or modifiers to the standard execution flow. Once a test is executed the automation server collects test logs and uploads them to a server and updates the test case database with the result of the test (pass / fail).

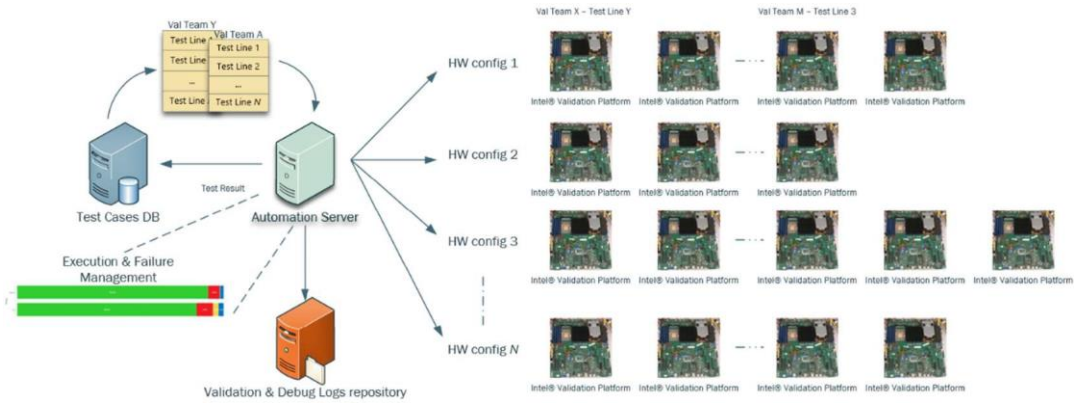


*Figure 2-5 Volume Validation and automation environment*

On every test the server automation runs a predefined set of steps to prepare the system for execution, this is known as the standard execution flow. The standard execution flow includes actions such as power cycling the system to start from a known good condition, clearing previous BIOS overrides and applying any new ones based on the test line, clearing any OS kernel overrides, checking for errors in the system, verifying all HW collaterals such as DIMMs and test cards are all up and detected. When the validation system is ready, then the automation server executes the test steps on the test line. When the test steps on the test line are completed, the automation server runs a Cleanup Flow where logs are collected and stored in the validation & Debug Logs repository. In the event the test line causes a system hang, the automation server detects the hang condition and runs an alternate cleanup flow called the Fail Flow. In the Fail Flow, the automation server collects debug information using Intel ® ITP tool and proprietary Python scripts to dump configuration and control registers, event counters, and internal FSM states. Then the automation server reboots the system and collects test logs for the validation engineer to analyze and debug the failure.

### 2.1.8 Debug

This phase is not serialized or separated from the rest of the silicon execution phases but runs along the execution. It is a necessary step in all post-silicon programs to analyze every single failure and root cause to identify all possible design bugs including HW and FW bugs.

# 3. PSiV methodology applied to a DDR5 Memory Controller

The PSiV methodology described in the previous chapter can be applied to any functional validation project and IP, however this work focuses on the validation of a DDR5 memory controller. The following sections will present how the methodology was applied.

## 3.1. DDR5 Memory Controller Validation Strategy

The DDR5 Memory Controller of the Next Generation Intel ® Xeon product line supports the features described in Table 1. Table 1 also presents a comparison to the previous generation memory controller.

*Table 1 Memory controller main features compared to previous generation design.*

| Feature | Next generation Intel® Xeon | Previous generation |
|---|---|---|
| DDR5 Frequency | Up to 6400MT/s in 1DPC RDIMM<br>Up to 5200MT/s in 2DPC RDIMM<br>Up to 8800MT/s in 1DPC MCR | Up to 4800MT/s in 1DPC<br>Up to 4000MT/s in 2DPC<br>No MCR support |
| DDR5 DRAM densities | 16Gb, 24Gb | Same |
| DDR5 DRAM width | x4, x8 | Same |
| DDR5 Memory capacities | 16GB, 24GB, 32GB, 48GB, 64GB, 96GB, 128GB, 256GB | Same |
| DIMM configurations | 2Rx4, 1Rx4, 2Rx8, 1Rx8, 2H, 4H, 4Rx8 (MCR) | 2Rx4, 1Rx4, 2Rx8, 1Rx8, 2H, 4H |
| Number of Channels | 12 | 8 |
| Memory form factor | RDIMM, 3DS RDIMM | Same |
| System memory bandwidth | Up to 51.2GB/s per channel | Up to 38.4GB/s per channel |

| | | |
|---|---|---|
| Security memory encryption | Total Memory Encryption (TME) and Multi-key Total Memory Encryption (MKTME) | Same |
| ECC protection | 128b, 125b, 96b | 125b, 96b |
| Patrol Scrub | Yes | Same |
| Error Correction and Scrub (ECS) | Yes | Yes |
| Command and Address Parity (CAP) | Yes | Yes |
| Runtime Soft Post-Package Repair (sPPR) | Yes | No |
| Adaptive Double Device Data Correction (ADDDC) | Yes | Yes |
| Mirroring | Full and Partial | Same |
| 3-way rank interleaving | Yes | No |
| Error signaling | MCA, SMI, ERR | Same |
| Refresh | All bank, same bank, fine grain, RFM, 2x Refresh | Same |
| APD/PPD | Yes | Yes |
| PkgC | Yes | Yes |
| Closed Loop Thermal Throttling | Yes | Yes |
| PM Maintenance | RCOMP, ZQCal, DQs retraining | Same |
| Scrambler | Yes | Yes |
| On-Die Tracer | Yes (Debug in Lab only) | No |

### 3.1.1 Validation SW strategy

As can be seen in Table 1, the new generation memory controller mainly differentiates from the previous program on the DDR5 supported frequencies, number of channels, support of Multiplexed Command Ranks (MCR) DIMMs, ECC modes, and sPPR. Because of this, a good level of the previous program content can be reused although porting is required. This defines the validation content strategy for the program to follow a reuse approach developing Python harassers to cover new features and extending existing Python scripts to cover for deltas from the previous design. Table 2 shows the different components of the validation content and whether they are new developments or port from previous programs.

*Table 2 Validation SW strategy*

| Validation Content | | Description | Port / New development |
|---|---|---|---|
| Validation OS | | Synthetic Linux based Validation OS | Port |
| Memory targets | | Validation filesystem drivers that expose blocks of memory (Targets) to the validation OS applications. These blocks of memory map the actual physical DRAM behind each memory channel. | Port |
| Memory traffic generator | | Validation OS native application that generates stress traffic (Memory Reads and Writes) to the memory targets.<br>It supports multiple algorithms to stress memory address and data lanes in the DDR5 bus and the DRAM. | Port |
| Python harassers | ECC | Python script focused on ECC injection, detection, logging, and correction | Port and extend |
| | ADDDC | Python script focused on ADDDC flow testing | Port |

| | | | |
|---|---|---|---|
| | Patrol Scrub | Python script focused on scrub interval testing along with error injection, detection, logging, and correction. | Port |
| | CAP | Python script focused on command and address parity error injection, detection, logging, and correction | Port |
| | sPPR | Python script focused on soft post package repair flow stressing | New development |
| | Mirror | Python script focused on full mirroring and address range mirroring flow testing | Port |
| | Refresh | Python script focused on memory refresh cycles and refresh management | Port |
| | Cke | Python script focused on APD/PPD low power states testing | Port |
| | Thermal | Python script focused on Closed Loop Thermal Throttling | Port |

### 3.1.2 Validation HW strategy

The number of combinations of Dual In-line Memory Modules (DIMMs) frequencies, densities, widths, configurations, and capacities is large and covering all of them in VV represents a high cost because VV usually uses large number of systems per HW configuration. To overcome this problem, the most common customer memory configurations are selected for the VV environment, while the rest of the configurations are covered in a separate set of validation platforms called Memory Matrix (MMX). There are two market segments that will be addressed with different product configurations. Well refer them as Platform 1 that supports 2DPC memory topologies and 8 memory controllers per CPU, and the other one as Platform 2 that supports 1DPC memory topologies but 12 memory channels per CPU.

# 3. PSIV METHODOLOGY APPLIED TO A DDR5 MEMORY CONTROLLER

Examples of VV HW configurations and MMX HW configurations are depicted in Figure 3-1 and Figure 3-2. Both pictures show the different DIMMs configurations per product following a DIMM nomenclature formed by [raw card][capacity]_[ranks]x[width]. Failover configurations are meant to validate scenarios in which a DIMM could go bad and system software isolates it then mapping out other DIMMs to create a performance balanced configuration.

| Platform 1<br>MEMOY - DD5 | VV Config. 1<br>256GB<br>1DPC | VV Config. 2<br>256GB<br>1DPC / Fail Over | VV Config. 3<br>1024GB<br>2DPC | VV Config. 4<br>256GB<br>1DPC | VV Config. 5<br>768GB<br>1DPC | MMX Config. 1<br>192GB<br>1DPC / Fail Over | MMX Config. 2<br>1024GB<br>1DPC | MMX Config. 3<br>128GB<br>1DPC | MMX Config. 4<br>192GB<br>1DPC | MMX Config. 5<br>512GB<br>2DPC / Fail Over |
|---|---|---|---|---|---|---|---|---|---|---|
| MC0-S0 | C32GB_1x4 | A64GB_2x4 | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | | 128GB_3DS | D16_1x8 | C48_1x4 | A64GB_2x4 |
| MC0-S1 | | | A64GB_2x4 | | | | | | | A64GB_2x4 |
| MC1-S0 | C32GB_1x4 | | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | D24_1x8 | 128GB_3DS | D16_1x8 | | |
| MC1-S1 | | | A64GB_2x4 | | | | | | | |
| MC2-S0 | C32GB_1x4 | A64GB_2x4 | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | | 128GB_3DS | D16_1x8 | C48_1x4 | A64GB_2x4 |
| MC2-S1 | | | A64GB_2x4 | | | | | | | A64GB_2x4 |
| MC3-S0 | C32GB_1x4 | | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | D24_1x8 | 128GB_3DS | D16_1x8 | | |
| MC3-S1 | | | A64GB_2x4 | | | | | | | |
| MC4-S0 | C32GB_1x4 | A64GB_2x4 | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | | 128GB_3DS | D16_1x8 | C48_1x4 | A64GB_2x4 |
| MC4-S1 | | | A64GB_2x4 | | | | | | | A64GB_2x4 |
| MC5-S0 | C32GB_1x4 | | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | D24_1x8 | 128GB_3DS | D16_1x8 | | |
| MC5-S1 | | | A64GB_2x4 | | | | | | | |
| MC6-S0 | C32GB_1x4 | A64GB_2x4 | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | | 128GB_3DS | D16_1x8 | C48_1x4 | A64GB_2x4 |
| MC6-S1 | | | A64GB_2x4 | | | | | | | A64GB_2x4 |
| MC7-S0 | C32GB_1x4 | | A64GB_2x4 | E32GB_2x8 | A96GB_2x4 | D24_1x8 | 128GB_3DS | D16_1x8 | | |
| MC7-S1 | | | A64GB_2x4 | | | | | | | |

*Figure 3-1 Memory Platform 1 VV and MMX HW configurations*

| Platform 2<br>MEMOY - DD5 | VV Config. 1<br>256GB<br>1DPC / Fail Ove | VV Config. 2<br>384GB<br>1DPC | VV Config. 3<br>576GB<br>1DPC | VV Config. 4<br>384GB<br>1DPC / Fail Ove | VV Config. 5<br>768GB<br>1DPC | MMX Config. 1<br>384GB<br>1DPC | MMX Config. 2<br>384GB<br>1DPC | MMX Config. 3<br>768GB<br>1DPC | MMX Config. 4<br>3072GB<br>1DPC | MMX Config. 5<br>1152GB<br>1DPC |
|---|---|---|---|---|---|---|---|---|---|---|
| MC0-S0 | | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC1-S0 | | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC2-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC3-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC4-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC5-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC6-S0 | | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC7-S0 | | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC8-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC9-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC10-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | C48_1x4 | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |
| MC11-S0 | E32GB_2x8 | C32GB_1x4 | E48_2x8 | | A64GB_2x4 | E32GB_2x8 | F32GB_1x4 | B64GB_2x4 | 256_3DS | A96GB_2x4 |

*Figure 3-2 Memory Platform 2 VV and MMX HW configurations*

Along with each HW configuration there are SW configurations made of different BIOS settings that define system behavior. The most relevant SW configurations for the memory controller are defined by the Page Policy (Adaptive or Closed Page), Mirroring (full mirroring or address range mirroring), and ADDDC (enable or disabled) settings; all other settings such as refresh mode, scrub rate, low power modes are set to their default values and modified through harassers. All the test cases run in VV are directed to all or some of the SW configurations on each HW configuration providing this way the required coverage for all features.

### 3.1.3 Debug Strategy

It is required to have memory bus observability to debug the most complicated memory problems in post-silicon. The current off-the-shelf solutions support only up to 4800MT/s in 1DPC. This Logic Analyzer uses a passive interposer and thus is limited to that speed.

The debug strategy for this product defined two DDR observability solutions: A new active interposer for the Logic Analyzer that was developed to support up to 5600MT/s in 1DPC population mode, and an internal On-Die Tracer. The active interposer, which has an RCD, requires to be trained along with the DIMM; this imposes challenges to the BIOS algorithms and the signal integrity in the bus. In addition to these observability solutions, Intel® developed an On-Die Tracer observability solution that works for frequencies at or above 5600MT/s. This On-Die Tracer observability solution is very well applicable for debugging protocol problems as being internal does not provide visibility between the actual DDRIO pins and the DIMMs.

The DDR observability solutions are complemented with visibility to some critical internal signals in the memory controller. These signals are part of the Visualization of Internal Signals Architecture (VISA) (Intel, 2021). Both the On-Die Tracer and the VISA observability solutions are available only when putting the system in a special debug mode that requires secure authentication to unlock the system. The following debug capabilities are available in all Intel® silicon and were used in this project: Hardware-based execution run control for basic execution control of the CPU, TAP access to registers that allow basic read and write operations on control and status registers of the memory controller via IEEE 1149.1 using an Intel® ITP box, Scandump and Array Freeze and Dump that provides access to FSMs in the memory controller and all the design.

## 3.2. Test Plan and Test Case Definition

The memory controller test plan was structured into three different sub-plans: Memory Base (MemBase), Memory Features (MemFeat), and Memory Cross Products (MemXProducts).

### 3.2.1 Memory Base (MemBase)

The MemBase test plan is focused on enabling and testing all basic features of the memory controller including DDR5 maintenance commands, internal memory transaction scheduling, memory topologies, DRAM refresh modes, address translation, clustering modes, and stress traffic patterns in the DDR5 bus. Table 3 describes each of the MemBase test plan sections and the number of test cases created for each one.

*Table 3 MemBase test cases description*

| Feature | Description | Number of Test Cases |
|---|---|---|
| DDR5 Maintenance commands | Validate DDR5 periodic maintenance command for RCOMP, ZQcal, DQs and their corresponding triggering flows | 7 |
| Transaction scheduling | Validate memory controller's ability to schedule transactions with supported page policies, stress of major mode switching, scheduling pre-emption, internal pending queues, dependent reads, and other performance features. | 12 |
| Memory Stress | Validate memory controller's ability to sustain traffic to the DDR5 bus using multiple address and data pattern algorithms | 32 |
| Memory Topologies | Validate the memory controller can control and use the different memory types (RDIMM, 3DS), configurations (1Rx4, 2Rx4, 2Rx8, 1Rx8), and densities (16Gb, 24Gb) from all defined memory vendors. Basic tests are run during PO. Full coverage is achieved during VV. | 36 |
| DRAM Refresh modes | Validate the memory controller can periodically refresh the DRAM using All Bank, Same Bank, and Refresh Management as supported by the different | 18 |

| | | |
|---|---|---|
| | DIMMs, as well as staggered and panic refresh modes. | |
| Address Translation | Validate the memory controller can effectively address the entire DIMM capacity behind the memory channel including memory interleaving in power of 2 and non-power of 2 topologies, along with reverse address translation for error logging. | 11 |
| Clustering Modes | Validate the uniform memory addressing and sub-Numa clustering through system configuration and DIMM populations. Covered through VV and MMX configurations. | 7 |

### 3.2.2 Memory Features (MemFeat)

The MemFeat test plan is focused on stressing memory Security, memory Power Management (PM) and memory reliability availability serviceability (RAS) features individually demonstrating they can be used by the customers of the product. The MemFeat test plan is composed of the PM and the RAS sub-test plans. These tests are run before any cross-products can be exercised. Table 4 shows a description of the PM sub-test plan features and the number of test cases created for each one.

*Table 4 Memory PM test cases description*

| Feature | Description | Number of Test Cases |
|---|---|---|
| CKE | Validates DDR5 power-down modes across RDIMM and 3DS DIMM topologies | 3 |
| I3C | Validates I3C controller used to interact through SMBus with the DIMM's EEPROM, PMIC, RCD, and Thermal sensor on-DIMM devices | 5 |

| | | |
|---|---|---|
| Package C | Validates Self-refresh entry/exit flow in the DDR5 bus when the system is doing core low-power states | 1 |
| Thermal Throttling | Validates memory controller support for traffic throttling when the system is reaching high temperature | 2 |

Table 5 shows a description of the RAS sub-test plan features and the number of test cases defined for each one.

*Table 5 Memory RAS test cases description*

| Feature | Description | Number of Test Cases |
|---|---|---|
| ADDDC | Validates ADDDC feature doing Bank forward and reverse sparing, Rank forward and reverse Sparing, and sparing flow under error injection conditions. | 14 |
| Mirroring | Validates memory controller can effectively participate in a Mirroring (full and partial) configuration where either half or a reduced range of memory is mirrored to another memory channel. | 15 |
| Patrol Scrub | Validates memory controller's ability to scrub the entire memory capacity on the DIMMs within a specified period detecting and correcting correctable ECC errors and properly logging and escalating uncorrectable errors | 19 |
| Runtime soft per part repair | Validates the memory controller's ability to spare copy data from a failing DRAM row to a spare DRAM row. | 7 |
| ECC | Validates the memory controller's ability to work in supported ECC bit modes, correctly detecting and correcting correctable ECC errors and properly logging and escalating uncorrectable errors. | 65 |

| | | |
|---|---|---|
| CAP | Validates the memory controller's ability to detect and correct command and address parity errors in the DDR5 bus | 4 |
| Error Correction and Scrub | Validates the memory controller's ability to instruct the DIMM to enable/disable Error Correction and Scrub. | 5 |

Table 6 shows a description of the memory Security sub-test plan features and the number of test cases defined for each.

*Table 6 Memory Security test cases description*

| Feature | Description | Number of Test Cases |
|---|---|---|
| Memory Integrity | Validates the memory controller's ability to detect and log errors in its internal SRAMs holding encryption keys while Integrity is enabled. | 4 |
| Multikey Total Memory Encryption | Validates the memory controller's ability to perform inline encryption under stress traffic using multiple keys and memory regions. | 7 |

### 3.2.3 Memory cross products (MemXProducts)

The MemxProducts sub-test plan is focused on functionally crossing all memory PM, RAS, and Security features while running stress traffic to demonstrate the memory features can work together. The cross-products test cases are the most complex testing scenarios and are derived from a cross-functional matrix. The post-silicon validation team discussed and agreed with the design team on what cross-products are more relevant to the product. Figure 3-3 shows an example of the validation cross-product matrix committed to the program; cells labeled xP are explicit test cases to be executed, and green cells are implicit test cases, this means that the crossing features will be covered while running other test cases as features are turned on by default (e.g. ADDDC x Patrol,

patrol scrub is always turned on by BIOS and configured with a scrub rate of 24hrs, when running ADDDC test cases in the MemFeat RAS test plan the crossing will be implicitly covered); the cells in blue are crossings that the memory controller does not support or there is no interest from customers to use.

| MemxProcuts | ADDDC | CAP | ECC | CKE | i3c | Mirroring | MKTME | Patrol/Scrub | runtime SPPR | PkgC | Refresh | Throttling | ECS (MRR/MRW) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADDDC | | | | | | | | | | | | | |
| CAP | xP | | | | | | | | | | | | |
| ECC | xP | xP | | | | | | | | | | | |
| CKE | | xP | | | | | | | | | | | |
| i3c | | | | | | | | | | | | | |
| Mirroring | | xP | xP | | | | | | | | | | |
| MKTME | | xP | xP | | | | | | | | | | |
| Patrol/Scrub | | xP | xP | | | | xP | | | | | | |
| Runtime SPPR | | xP | xP | xP | | | | | | | | | |
| PkgC | xP | xP | xP | | xP | | xP | | | | | | |
| Refresh | xP | xP | xP | xP | | xP | | xP | xP | xP | | | |
| Throttling (CLTT) | | xP | xP | | xP | | | | | xP | xP | | |
| ECS (MRR/MRW) | | xP | xP | xP | | | | | | | xP | xP | |
| Warm Reset | | xP | xP | | | | | | | | xP | xP | |

| | |
|---|---|
| | Implicit coverage |
| xP | Explicit coverage |
| | Not Interesting / NA |

*Figure 3-3 Memory cross-products matrix*

### 3.2.4 VV execution budget

The test plan described in the previous sections greatly expands on VV as most of the test cases are run in all planned HW and SW configurations. The actual VV execution capacity required by memory was calculated by 350 * 10 HW * 3 SW * 2hrs = 21000hrs of execution.

Considering an efficiency factor of 40%, the total VV execution capacity was 90,720 hours of execution to complete the plan in the program indicated timeframe. Since that time must consider test reruns and the fact that it is distributed across validation teams, the memory team had to select which tests were critical to run in all three SW configurations and which we would run in one or two of them. After that analysis the team concluded to run only 1300 test lines total consuming 5200 hours of the total execution budget.

## 3.3. Test Content Verification

During the content verification phase, we used the pre-silicon validation platforms described in Section 2.1.4. to ensure content was ready for PO, and as a byproduct of running the content in Emulation platforms, expose any possible RTL and/or FW bugs before the A0 tape-in.

We used three main indicators, one for content readiness which accounted for content development for each test case (e.g., porting or writing new test and debug scripts, defining traffic generator command lines), one for content enabling which accounted for the individual pieces of content (e.g., Python scripts, traffic generator command lines) tested in a VP or a PPP, and one for intent verification in RTL which accounted for the individual pieces of content tested in emulation platforms. Figure 3-4 shows the content readiness indicator used to track content development progress per week as can be seen by 1p0 milestone we were able to reach 97% of content development of the 350 test cases planned, only 2 of those cases were blocked due to a dependency on the On Die trace debug software and 7 more were still in development at that time. Considering only actual Python scripts and OS readiness the team was able to complete what was necessary to start PO.

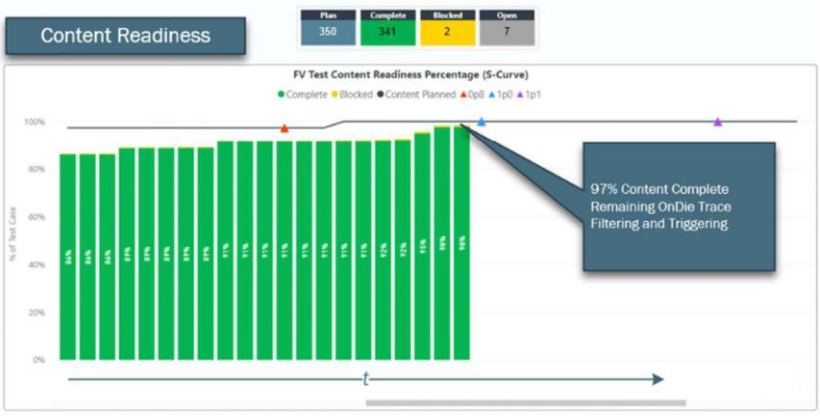


*Figure 3-4 Content Readiness indicator used to track content development. Source Intel*

Figure 3-5 shows the content enabling indicator used to track the basic quality of the content by running in VP or PPP platforms. We planned to run in VP and PPP platforms the scripts and tools necessary to run the post-silicon test cases. As can be seen the plan was made of 55 test cases that covered all major pieces of test content from which we passed 39 (71%) in VP by the

1p0 milestone ensuring coding and basic algorithmic problems were resolved. 13 test cases were blocked due to limitations in the VP on error injection capabilities forcing us to move that content to emulation platforms. 3 test cases were marked as failed which were later fixed.

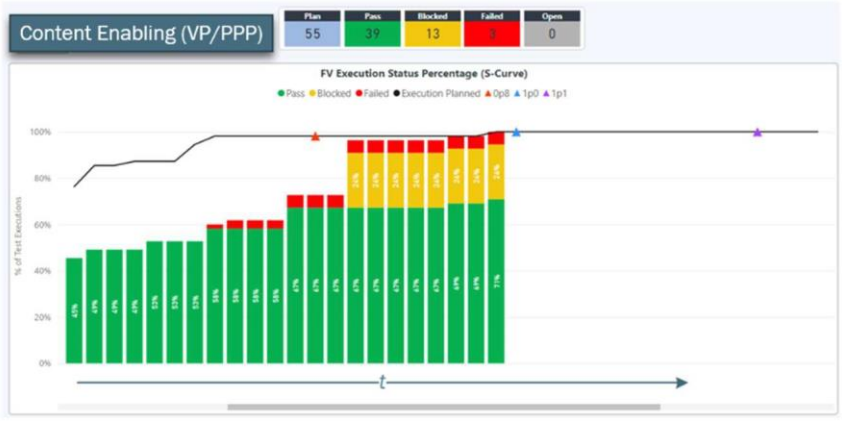


*Figure 3-5 Content Enabling indicator used to track content execution in VP/PPP. Source Intel*

Figure 3-6 shows the intent verification indicator used to track execution in emulation platforms. Given that emulation platforms were not only used to verify content intent but to also do bug hunting as an effort to increase the quality of the design by A0 TI, the type and quantity of the test cases selected to run in emulation differs from those that run in VP. As shown in Figure 3-6 we selected 75 test cases, however we were able to successfully run only 31, having 30 (40%) passes and 1 failure. Different factors affected execution throughput during this phase being the most important ones: emulation boxes availability, model stability, and maturity of FW ingredients which were under development at the same time we were running our test cases.

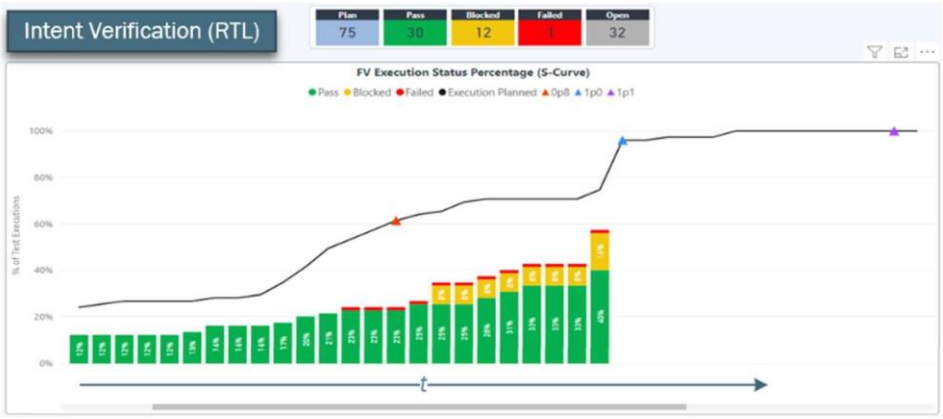


*Figure 3-6 Intent verification (RTL) indicator used to track execution in emulation models. Source Intel*

Executing some of our test cases in emulation platforms allowed us to gain deeper knowledge of the memory controller, and to put into practice processes that will later be used in post-silicon such as the pre-sighting process and debug management.

## 3.4. DDR5 Memory Controller enabling and silicon checkout during PO

The PO of the product started with raw unfused parts. The tools team developed a script to allow taking the part out of reset and start fetching BIOS instructions from an SPI flash in the board. The boot script applies a set of fuse values which configure the product to work with its basic configuration. Internal PLLs in the memory controller and DDRIO circuits get their configuration from this set of fuse values allowing them to lock at frequencies necessary for the memory controller and the DDRIO to operate correctly and train the DDR5 memory.

The CPU executes the BIOS code for memory training called Memory Reference Code (MRC); there is a specialized team in charge of achieving DDR5 memory training. This process uses specific logic in the memory controller to initiate transactions in the DDR5 bus to tune sampling timings for all DDR5 signals including DQs, CS, DM, Command/Address, DQ, etc. (JEDEC, 2021). The first boot was achieved at 4800MT/s and later the MRC team achieved 5600MT/s and 6400MT/s.

Once the first boot to OS was achieved, the team started working on functional validation of the memory controller. 215 test items were planned for PO, 100% were executed, 94% passed, 4% failed, and 2% remained blocked due to content issues. One silicon bug was found, and a workaround identified to allow volume validation until fix arrived in B0 stepping. The PO was completed 2 weeks ahead of plan with no blockers to VV.

## 3.5. DDR5 Memory Controller VVR and VV

During VVR phase, which was embedded in between PO and VV, the team focused on demonstrating all DIMM types from the 3 selected memory vendors successfully trained to all

supported speeds and pass stress traffic for at least 1hr without logging errors. This task was important to detect any possible MRC issues and avoid VV from being blocked. Figure 3-7 shows an example of a training heat map that served the purpose of tracking the work and communicating upstream the memory enabling status. This information was used by upper management to understand the main problems and what help was needed, also it was used by the office responsible for interfacing with the memory vendors and the ecosystem to drive vendor involvement strategies. The training heat map shows the DIMM configurations enabled per Vendor at different DDR5 frequencies, green cells indicate the link training was successful and stress traffic encountered no errors, white cells indicate the DIMM has not been tested at a particular DDR5 frequency, gray cells indicate that despite the DIMM type is POR we hadn't received samples from the Vendor, and finally black cells indicate that the DIMM type and configuration is not POR for the product. The memory configurations POR evolved along the project removing DIMM types and configurations the customers are not interested in consequently reducing validation effort, time, and cost.

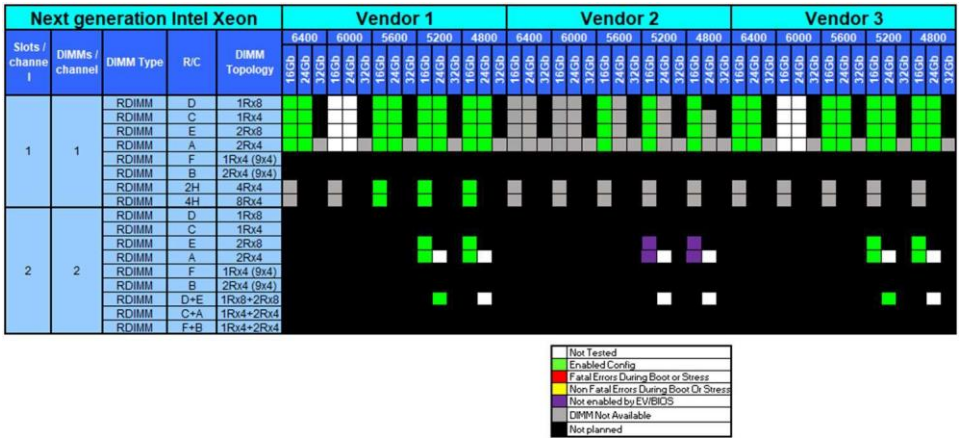


*Figure 3-7 Memory training heat map from functional validation. Source Intel*

The VV phase was executed in two validation cycles being the first one focused on a chopped version of the product that featured only 4 memory controllers and a reduced number of CPU cores, the second one focused on the validation of the full product featuring 12 memory controllers and the full count of CPU cores. For the first validation cycle, the team uploaded to the automation server 1300 test lines to run in the VV systems and 800 test lines to run in MMX systems. The first validation cycle was run for 18 weeks with a goal to reach 70% of the tests passing; Figure 3-8 shows the execution progress (stacked bards of the pass, fail and blocked test

lines) versus plan (black curve), a pass plan (green curve) and the pass rate (blue curve) defined as the ratio of pass test lines to actual executed test lines per week.

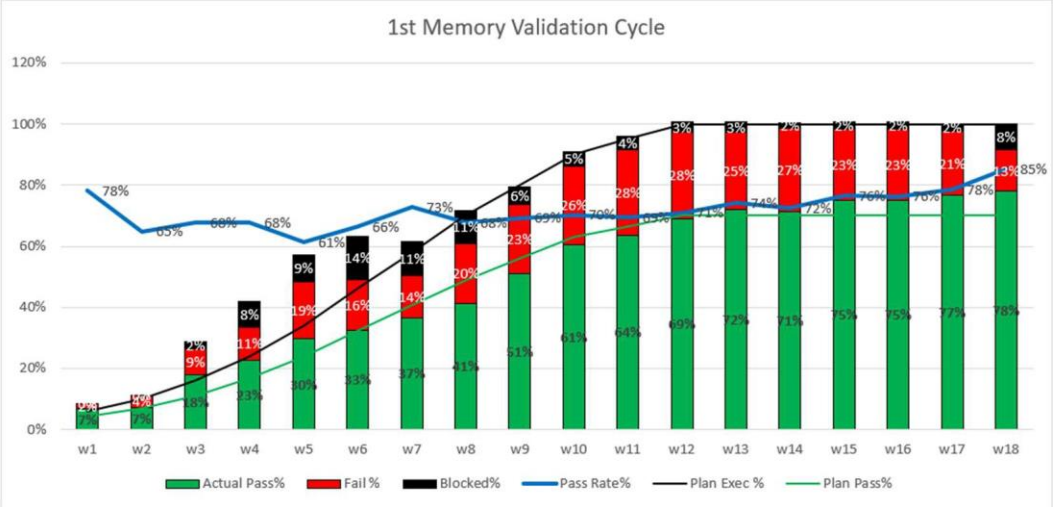


*Figure 3-8 1st Memory VV cycle. Source Intel*

At the beginning of the validation cycle the MemBase tests were executed demonstrating basic features were healthy enough to progress to MemFeat and MemXProducts test cases execution. The team was able to reach 70% pass 13 weeks after the start of the validation cycle, however test failures quickly built up ranging from 20-28% of the total test count. Handling this high number of test failures became a challenge as the engineering team needed to quickly discriminate between automation environment noise, content issues, BIOS and FW bugs, and real silicon bugs. Content updates were continuously needed as we advanced our understanding of several features' functional behavior, architectural limitations, BIOS updates and Python collaterals changes. To reduce the failures backlog we added four more engineers to the team of ten working on the project. The functional features with the greatest number of failures were ADDDC, runtime SPPR, and Mirroring. These three features required a deep dive discussion with the design team to ensure our content was following the expected configuration and testing flows to enable them in VV. Several sections of the content were rewritten and retested again in silicon.

The second validation cycle started three weeks before the first validation cycle concluded. This is explained by the silicon ramp-up in the Lab and the velocity to upgrade the validation systems with new samples of the fully featured product. For this cycle, the team ran a regression of the first validation cycle focused on feature scalability and reruns of the tests that previously

failed in the first validation cycle, the team uploaded to the automation server 524 test lines to run in the VV systems, and 590 tests to run in MMX systems. For this cycle the goal was set to 100% execution and 80% pass by week 32. The team met the goal and reached 94% pass before B0 tape-in. As can be seen in Figure 3-9 between weeks 23 and 29 the failure count averaged 27% of the executed test lines facing similar problems as in the first validation cycle however aggravated by a problem induced by a change in BIOS that caused the system to run slowly limiting the ability to fully stress the memory controller while running MemXProduct test cases. After a month of running a task force the slowness issue was resolved and the team was able to successfully execute the most complex test cases in ADDDC, runtime SPPR and Mirroring, exposing three silicon bugs and two architectural bugs in these areas.

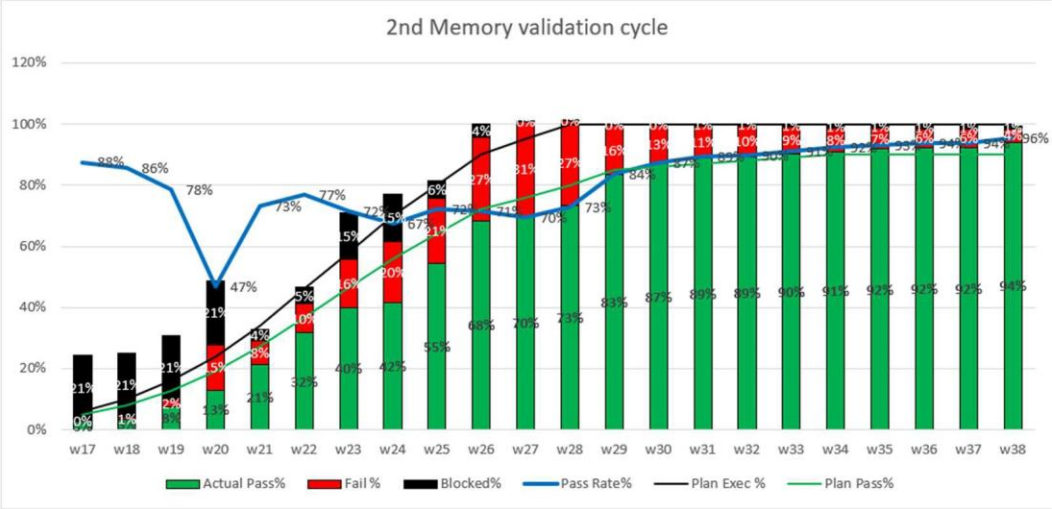


*Figure 3-9 2nd Memory VV cycle. Source Intel*

An important part of memory validation is the memory cross-products testing as it is quite helpful in exposing both silicon and FW bugs. The MemXproducts test execution was tracked more closely from a coverage and pass rate perspective using the cross-products matrix shown in section 3.2.3. During program execution it served to review progress on content enabling in VV and MMX and silicon health on feature crossings using a color coding based on pass rates. Figure 3-10 shows an example of an execution snapshot with areas with low pass rate (<50%) highlighted in yellow and red; those areas required deep dives to analyze the risk for the B0 TI and respond to a simple but complicated question: "*Is there a silicon bug behind?*". The same question can be asked for the areas with 100% pass rate as validation is always at the mercy of the quality of the

test and validation content, thus test plan reviews, paranoia checks, and a mentality of *"what else can we do to break the design?"* are critical to exercise along the project in order to improve the quality of the validation and therefore the quality of the product.

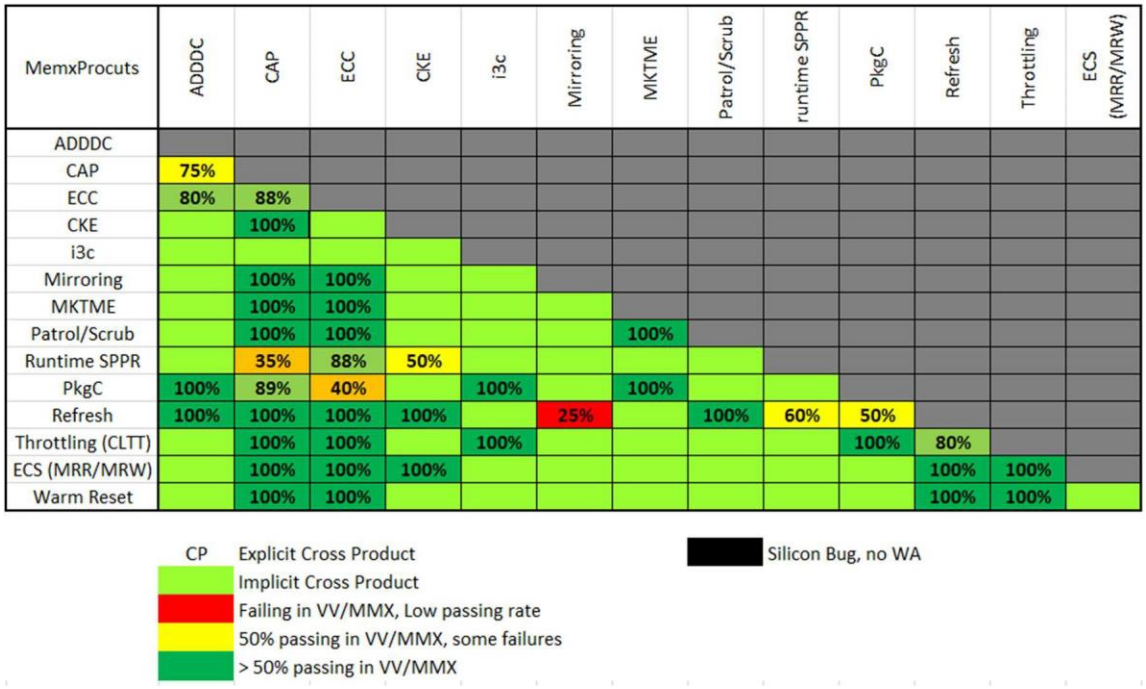


*Figure 3-10 Memory cross products matrix by B0 TI. Source Intel*

## 3.6. DDR5 Memory Controller Post-Si Debug

Debugging is a critical task and the success of the validation project and therefore the quality of the product depends on it. The pace and effectiveness of the debug tasks and experiments, the clarity to describe what is and what is not the observed problem (Agans, 2002) have a direct impact on the schedule of the program, for this reason assigning the correct resources in quantity and experience is fundamental to resolve issues during the execution of a validation project. Usually, the most senior engineers work on the most complex problems while the junior engineers perform failure triage and run experiments guided by a senior engineer or technical lead. Initially, I organized the team in a way that everyone would be responsible for a particular feature or features and would do failure triage, pre-sighting, and sighting debugging. This pretended to allow junior engineers to gain experience as we executed the program, however when complex

issues started to be discovered and discussed with the Design team I had to modify and reorganize such that critical sighting debug could make effective progress to root cause the problems.

As it was discussed before, all test failures, either generated by environment issues, BIOS and FW bugs, or real silicon bugs, all need to be understood and resolved. To do so we followed a systematic process depicted in Figure 3-11, of reviewing test logs of each test run, associating the failure to a pre-sighting, and debugging the pre-sighting to a point in which we could determine if the problem was content or environment issue, or it had a high probability (>50%) to be a BIOS/FW or silicon bug or we needed further help from the Debug Forum composed of design, BIOS, FW, and other validation experts. If a failure was caused by content we fixed it, tested the content fix locally, and requested an update to the validation environment to rerun the test. If the failure was caused by an environment problem (e.g., board, DIMM, automation flow, etc.) we worked together with the VV Team to resolve the problem and rerun the test. When a pre-sighting was promoted to a sighting, the problem was discussed in the Debug Forum where further experiments were defined and results were analyzed. In some cases, pre-silicon simulations were required to confirm the problem was a silicon bug. It is worth noting that the Debug Forum also worked on sightings sourced by other teams including EV, PnP, Platform Validation, and others.

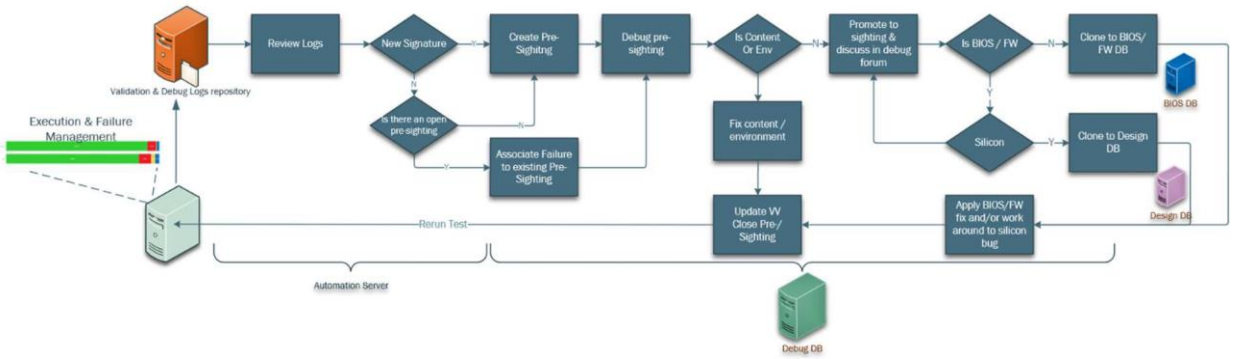


*Figure 3-11 Failure and system debug process*

During the validation of the memory controller many pre-sightings were filed reaching by B0 TI the considerable amount of 2186. Figure 3-12 shows the distribution of the closing conclusions for all of them and as can be seen environmental problems (e.g., automation, execution, and test content) represent 34% of failures associated to pre-sightings; in the same lines rejected/merged represent another 34% of the pre-sightings filed which consume engineering bandwidth to analyze and dispose although a large amount of them are quick to dispose of; 11%

of the pre-sightings were concluded to be BIOS issues, Board, HW, FW, and Doc add up to 4% of the issues and these along with BIOS were promoted to sightings for proper resolution. An area of opportunity to reduce the program cost and time to market is to reduce the number of failures caused by environment issues including test content which represented 34% of the total environment bugs.

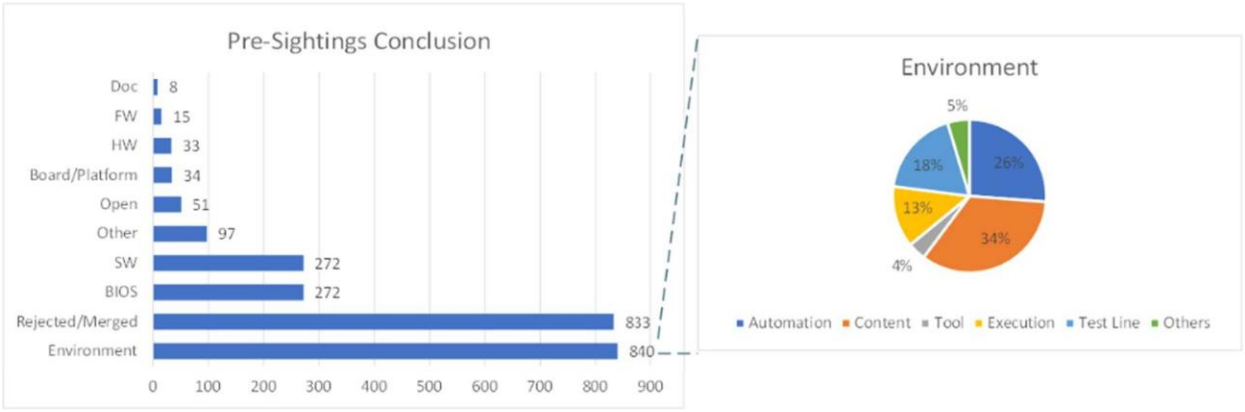


*Figure 3-12 A0 Validation pre-sightings by conclusion. Source Intel*

The Debug Forum set a very high bar to the quality of the sightings that were filed forcing the entire organization to increase discipline and meet a minimum standard on the amount and quality of the information provided in the sightings. This was done to increase the effectiveness of the Debug Forum in root causing the problems, the better the information the faster an issue can be resolved. Figure 3-13 shows the distribution of sightings per conclusion for the 145 sightings filed since the PO to the B0 TI having Rejected / Not a Bug representing 39% of the total sightings; SW bugs (OS, applications, content) were 28%, and HW (silicon, arch, tunning) added up to 23% of the total sightings. HW silicon bugs were 12% of the total sightings; finding them during A0 assured a better B0 stepping quality and a better product for our customers.

# 3. PSiV METHODOLOGY APPLIED TO A DDR5 MEMORY CONTROLLER

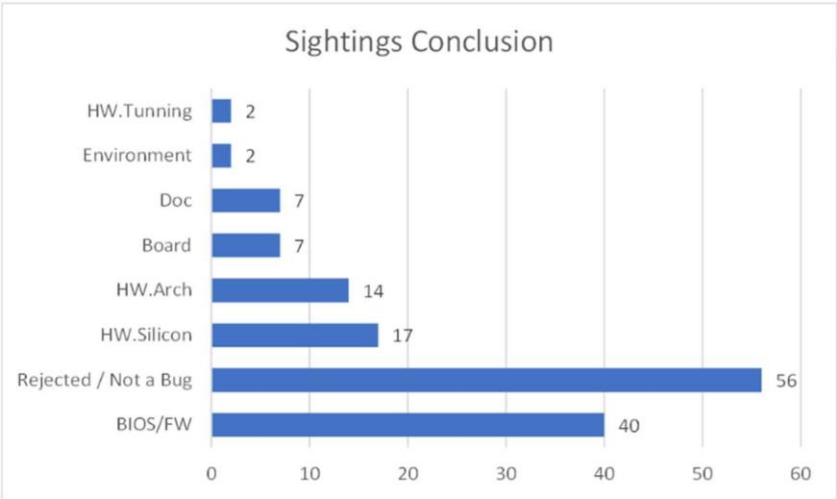


*Figure 3-13 Sightings by conclusion. Source Intel*

As a final note in post-silicon debug, Figure 3-14 shows the turnaround ratio per week since PO to the B0 TI. In simple terms, the graph represents the ratio of open and closed sightings per week and tracks the number of high-priority sightings per week. As can be seen, the higher number of open sightings took place at 3 times in the program: during PO, at the transition to the SKU with 12 memory controllers, and when we got closer to the B0 TI when it was critical to analyze every failure and determine nothing was left behind so the next stepping could have the best quality possible. During those picks engineering resources were also redirected to debug and root cause sightings.


*Figure 3-14 Sightings turnaround ratio per week. Source Intel*

# 4. Discussion

## 4.1. A successful project

The post-silicon validation methodology was applied to the memory controller of the Next Generation Intel® Xeon Server CPU as described in Chapter 3.

The validation project completed PO in 6 weeks – 2 weeks ahead of plan, executed over 1300 test lines along the duration of the two validation cycles in A0 stepping, reaching a 94% pass in VV by B0 tape-in, and finding 17 silicon bugs, 14 architectural bugs, and 8 documentation bugs, along with 38 BIOS and FW bugs.

We can evaluate the success of the project using two main metrics, one is Time-to-Market measured in terms of number of days ahead of committed schedule where a product consistently meeting the milestones on time gets a positive note (e.g., declaring B0 tape-in on projected drive-to schedule) while a product missing or delaying milestones get a negative grade; it is worth noting that declaring a milestone is most of the times a project management decision that takes input from different stakeholders including Design, Manufacturing, and Post-Silicon Validation. The other metric is the number of bugs found during the project and the managed risk of remaining fail tests at the time of the milestone decision. In the case of the Next Generation Intel® Xeon CPU the projected number of bugs from pre-silicon to post-silicon was around 36 bugs in the Memory Controller, although this number may be perceived as arbitrary it is derived from the number of pre-silicon bugs found during design verification, and the ratio at which the pre-silicon bugs were found. Evaluating the DDR5 Memory controller validation project against both metrics we can conclude that it was a successful project as validation did not delay B0 tape-in and completed PO 2 weeks ahead of plan, and the number of bugs found was 39 excluding BIOS and FW bugs.

## 4.2. The cost of post-silicon validation

We can ask ourselves: *what is the cost of finding those 17 silicon bugs?* To answer this question, we must consider multiple factors such as time, effort, head count, a fix execution cost composed of the number of validation platforms, memory DIMMs, CPU samples, Lab space, electricity, cooling systems, Lab host computers, debug equipment, and the criticality of those bugs. Can we deliver to our customers the product with those bugs? Will we lose a competitive advantage? Certainly, the customer is the one that ultimately defines quality and what are the most important features for them to succeed. Putting everything together, every bug found in the A0 stepping is far less expensive today than finding it later (e.g., during B0) as it would cost another stepping plus an extra validation cycle, schedule delays which impact customers' ability to launch their products and services, and our own ability to produce profits, or even worse, requiring a product recall if the bug is found by our customers, which by far would damage the company's reputation, stock market value, and the trust of our customers.

## 4.3. Areas of improvement

Nevertheless, there are areas of the validation execution that need to be improved. As described in Section 3.6, an important area to address is the high number of pre-sightings derived from execution failures caused by content and automation issues that altogether represented 34% of the total number of pre-sightings the team had to deal with.

Despite we reached 97% of content development completion and were able to verify major pieces of it in VP reaching 71% passing, we struggled during the PO and the early VV days, mainly because of three factors: one being the limitations encountered in the VP models such as the lack of error injection support which impaired us to test the content correctness to control the error injection logic of the memory controller. The second one being the fact that PO started with unfused chopped samples and all the content was developed considering the full design; multiple assumptions on system behavior were not realized in the first units of the chopped samples thus forcing us to add multiple hacks to make the content to work; such hacks need to be removed once

the full chip samples arrived. The third one was a degree of inexperience in most of the team members who had little bit more than one year in the company, leading to several sections of the content to be ported not fully addressed and therefore not matching with silicon changes from the previous generation CPU. Addressing these factors will certainly improve the content readiness in the next project and therefore will reduce the number of failures derived from it.

To reduce the number of failures coming from the automation environment the VVT in conjunction with the functional validation teams need to work in improving both the quality of the platform checkers and the objective of having those checkers in place. A multitude of failures came from system misconfigurations originated by wrong system assembly; others came from changes in BIOS due to new features being enabled, and other from BIOS bugs; the checkers were able to catch those bugs, but the execution team perceived them as checkers failure.

## 4.4. Debug acceleration

Analyzing failures and associating them to pre-sightings can be a tedious task. It requires downloading multiple log files from the file server, looking for the particular step of the automation flow that reported a failure, and understanding the print messages in the concerning log files, then manually creating a pre-sighting with a clear description of what the failure is, what was the system configuration when the failure occurred, and what the possible trend of the failure is, if it is pointing to a silicon problem, BIOS/FW problem, or content/automation problem. Although a good level of knowledge on the memory controller is required to correctly triage a failure, this activity can be accelerated with the implementation of an automatic system to analyze the logs.

During the program it was put in test an AI driven system that pretended to accelerate the initial analysis of the logs and propose pre-sightings, however the system was not trained before it was used therefore requiring the engineers to create signatures to help the AI system, making it a little bit of a burden. Nevertheless, AI triaging acceleration is an area of interest for future developments.

# Conclusions

A post-silicon validation methodology for functional validation was presented with sufficient details to explain each of its phases from pre-silicon planning and readiness to post-silicon execution.

During the pre-silicon planning and readiness, it covered the definition of the validation strategy which encompasses the strategic decisions about what HW, SW, Debug equipment, and automation infrastructure will be used to validate the product being designed. In addition, it covers the test plan design, content development and content verification.

During the post-silicon execution, it defines the phases for power-on, volume validation readiness, and volume validation across the different test cycles and steppings of the product until the production release qualification is declared.

The results of its application to the validation of a DDR5 Memory Controller which is part of the Next Generation Intel® Xeon CPU are documented. Details on what DIMM populations and DIMM types were validated are provided as part of the validation strategy phase. In a similar manner, the overall definition and structure of the Test plan and number of test cases is documented. The verification of the test content in pre-silicon platforms is discussed.

The post-silicon execution results are presented, highlighting the fact that 17 silicon, 14 architectural, 8 documentation, and 38 BIOS/FW bugs were discovered during the validation of the A0 stepping. The debug process and metrics that show the distribution of the conclusions of the pre-sightings and sightings are documented. These metrics are key to understand what were the major areas that sourced problems.

Beyond showing the execution results during PO and VV, this work provides an analysis of the results and identifies areas of improvement and future development works on the application of AI to accelerate failure triage and pre-sighting and sighting debug.

This case study demonstrated the value of post-silicon validation.

# Bibliography


Agans, D. J. (2002). *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems Illustrated Edition.* New York: AMACOM.

AMD. (2023). *AMD EPYC™ 9354P*. (AMD) Retrieved from https://www.amd.com/en/product/12611

Bailey, B. (2018, July 26). *When Bugs Escape*. (Semiconductor Enginering) Retrieved October 7, 2023, from https://semiengineering.com/: https://semiengineering.com/when-bugs-escape/

Intel. (2021, 7 28). *Intel Debug Technology*. (Intel) Retrieved 9 14, 2023, from https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/intel-debug-technology.html

Intel. (2023, August 30). *hot-chips-2023-bhs-and-granite-rapids-xeon-architecture-presentation.* Retrieved October 7, 2023, from www.intel.com: https://www.intel.com/content/www/us/en/content-details/787386/hot-chips-2023-bhs-and-granite-rapids-xeon-architecture-presentation.html?wapkw=granite%20rapids

Intel. (2023, August 30). *hot-chips-2023-granite-rapids-and-sierra-forest-xeon-press-briefing-presentation.* Retrieved October 7, 2023, from www.intel.com: https://www.intel.com/content/www/us/en/content-details/787432/hot-chips-2023-granite-rapids-and-sierra-forest-xeon-press-briefing-presentation.html?wapkw=granite%20rapids

Intel. (2023, Jan 01). *Intel® In-Target Probe (Intel® ITP) eXtended Debug Port (XDP)*. Retrieved Sep 4, 2023, from https://designintools.intel.com/intel-in-target-probe-intel-itp-extended-debug-port-xdp.html

Intel. (2023, Jan). *Intel® Xeon® Platinum 8452Y Processor*. (Intel) Retrieved from https://www.intel.com/content/www/us/en/products/sku/231761/intel-xeon-platinum-8452y-processor-67-5m-cache-2-00-ghz/specifications.html

JEDEC. (2021, August). *DDR5 SDRAM.* JEDEC Global Standards for the Microelectronics Industry. Retrieved Oct 3, 2023, from JEDEC: https://www.jedec.org/standards-documents/docs/jesd79-5a

Samsung. (2023). *RDIMM.* Retrieved Oct 3, 2023, from https://semiconductor.samsung.com/us/dram/module/rdimm/

SkHynix. (2023). *RDIMM.* Retrieved Sep 2, 2023, from https://product.skhynix.com/products/dram/module.go

Stecyk, S. (2016, October 21). *https://www.intrinsix.com*. (Intrinsix) Retrieved October 7, 2023, from https://www.intrinsix.com/blog/bug-tracking-for-soc-asic-design-verification-projects

Takashi, D. (2011, February 1). *Intel's billion-dollar mistake: Why chip flaws are so hard to fix*. (Reuters) Retrieved October 7, 2023, from reuters.com: https://www.reuters.com/article/idUS370181011120110201