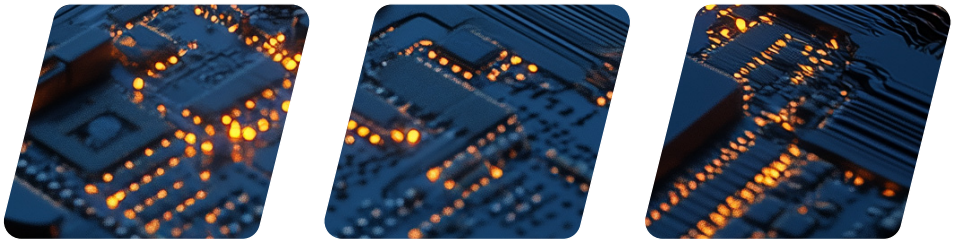


Modular Firmware and FPGA-Based Plug-and-Play: A Configuration-Driven Operating Model for Hyperscale Data Center Platforms



White Paper

Author:

Munir Ahmad, Distinguished Engineer, Lattice Semiconductor

DISCLAIMERS

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. Lattice products and services are not designed, manufactured, or tested for use in life or safety critical systems, hazardous environments, or any other environments requiring fail-safe performance, including any application in which the failure of the product or service could lead to death, personal injury, severe property damage or environmental harm (collectively, "high-risk uses"). Further, Buyer must take prudent steps to protect against product and service failures, including providing appropriate redundancies, fail-safe features, and/or shut-down mechanisms. Lattice expressly disclaims any express or implied warranty of fitness of the products or services for high-risk uses. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

INCLUSIVE LANGUAGE

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

ABSTRACT

Hyperscale operators have standardized the mechanical and electrical layers of the server through the Open Compute Project's Datacenter Modular Hardware System (DC-MHS), but firmware has not kept pace. This paper describes Modular Plug-and-Play (M-PnP), a configuration-driven approach that moves platform variation out of source code and into machine-readable descriptors, using a common boot firmware and baseboard management controller (BMC) image, self-describing hardware, and an on-module FPGA as the discovery authority. Across a representative set of ten platform variants, the model is projected to reduce total firmware engineering effort by roughly 66%. Security considerations for this model remain under evaluation.

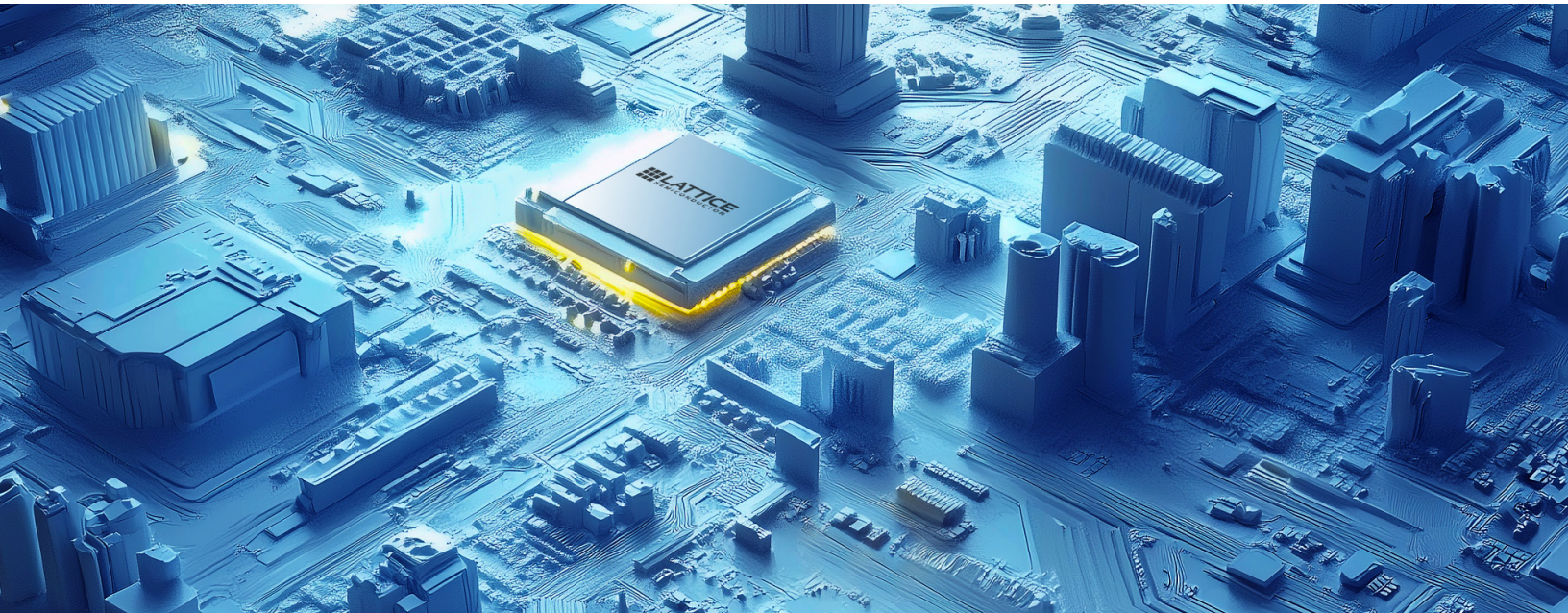


TABLE OF CONTENTS

Disclaimers 2
Inclusive Language 2
Abstract 2
Executive Summary 4
The Modular Data Center and Its Firmware Gap 4
The Scaling Challenge 4
The M-PnP Solution: Full System Modularity Including Firmware 5
Deployment Model: Multi-Generation, Multi-Platform 7
The Demonstration 8
Business Case and Value Proposition 8
Why It Matters: A New Firmware Operating Model at Hyperscale 9
Conclusion 10
Glossary 10
Productivity Model, Basis, and Assumptions 11

Executive Summary

Many hyperscale operators have adopted standardized mechanical and electrical layers of the server through the Open Compute Project's DC-MHS. Modules are designed to fit and power up across vendors. Firmware, however, has not kept pace. Each new host processor module, stock-keeping unit, or silicon generation typically triggers bespoke BIOS and BMC engineering, a fresh validation pass, and a new security qualification. The practical result is that plug-and-play stops at the connector: the hardware is modular, but firmware remains the bottleneck to scale.

This white paper describes Modular Plug-and-Play (M-PnP), an approach that implements the OCP DC-MHS MHS-V3 specification together with the OCP M-PnP Interoperability Design Specification so that platform variation moves out of source code and into configuration. A single qualified firmware base, consisting of common boot firmware paired with a common BMC image, is parameterized at runtime by self-describing hardware and machine-readable descriptors (field-replaceable unit (FRU) and JSON). An on-module FPGA acts as the discovery authority: it exposes topology and normalizes devices so the BMC and BIOS can bring up a platform without prior, hardcoded knowledge of it. AI-assisted orchestration generates the descriptors and manages their delivery across the fleet.

Across a representative set of ten platform variants, this model is projected to reduce total firmware engineering effort by roughly 66%, from about 210 to about 71 engineer-weeks, while cutting platform porting by 50–70%, accelerating validation and qualification by 30–50%, and shortening time-to-market by 60–85%. Beyond the engineering savings, a single qualified image spanning SKUs, generations, and silicon vendors can strengthen security posture and can provide a foundation for SaaS-style, fleet-scale firmware management. The sections that follow set out the challenge, the architecture, the deployment model, a planned demonstration, and the supporting business case. These are directional planning estimates, not commitments (see Table 3, Productivity Model, Basis, and Assumptions).

The Modular Data Center and Its Firmware Gap

The OCP DC-MHS family decomposes the server into standardized, interchangeable building blocks, most visibly the Host Processor Module (HPM), the Datacenter Secure Control Module (DC-SCM), and the defined mechanical and electrical interfaces that join them. This modularity has delivered real gains in supply-chain flexibility and mechanical reuse, allowing operators to mix compute modules and management modules with confidence that they will physically connect and power on.

The gap is what happens next. DC-MHS standardizes the mechanical and electrical domains, but the firmware that actually brings a platform to life (the BIOS/UEFI and the BMC stack) typically remains specific to each platform. Base specifications guarantee that parts fit and power up; they do not guarantee that a common firmware image will boot an arbitrary combination of modules all the way to a functional operating system. Today's platform development therefore tends to sit at one of two levels of interoperability maturity:

- **Plug-and-Code:** base specifications cover only electrical and mechanical interoperability. Everything above that boundary, including the firmware that makes components work together, is left to custom development.
- **Plug-and-Play 'Lite':** an interoperability framework is defined, but a significant amount of code is still required to stitch the framework pieces together and reach a baseline level of functionality, such as booting to an operating system.

The aspiration is full plug-and-play at the system firmware level: hardware that describes itself, firmware that adapts at runtime, and no rebuild required to absorb a change. Closing that gap is the purpose of M-PnP.

A note on scope. The primary objective is interoperability across multiple SKUs and platform or silicon generations. That interoperability is designed to be extensible across heterogeneous architectures (x86, Arm, RISC-V) and across silicon vendors such as Intel, AMD, and NVIDIA.

The Scaling Challenge

When firmware must encode platform knowledge in advance, three structural problems emerge that together cap how far a fleet can scale.

Extremely Prescriptive Hardware Design

Each system carries its own requirements and configuration variability. Because the firmware cannot discover those details on its own, they must be captured explicitly in code, and every deviation between platforms becomes a deviation the firmware has to account for by hand.

Custom Firmware Per Hardware SKU

The BMC needs prior knowledge of the system to function, and each BMC firmware image is typically compiled independently for its target platform. The boot firmware follows the same pattern. The number of images to build, sign, and maintain therefore typically grows in step with the number of SKUs.

Scalability at Risk

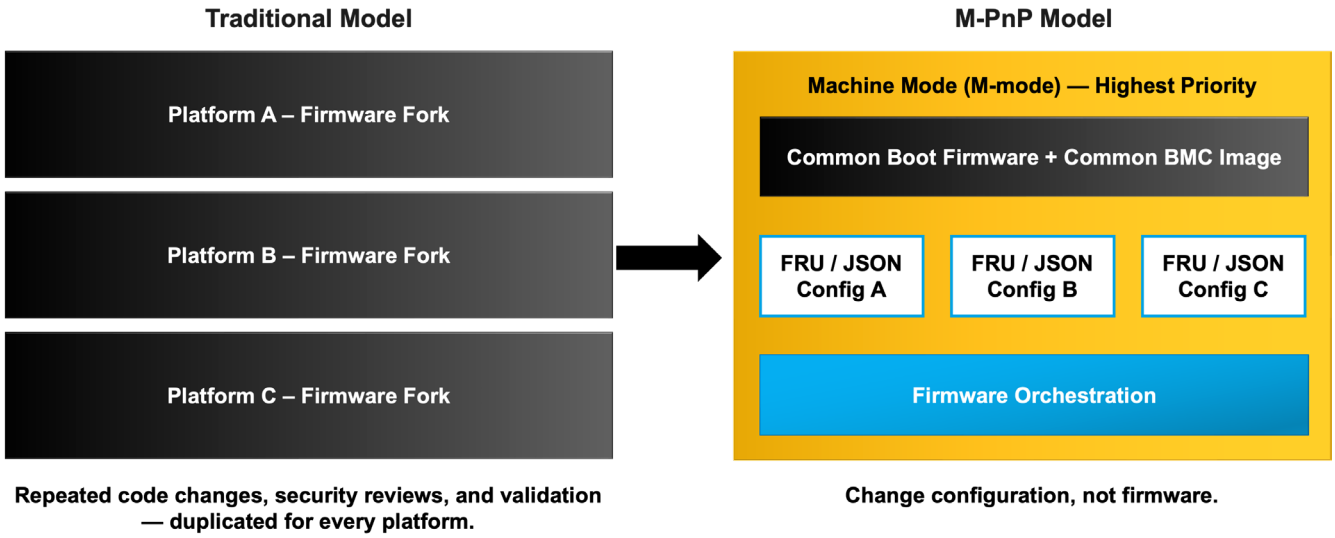
Without dynamic configuration and runtime system composition, the build-and-qualify operation simply does not scale. Every new module combination multiplies engineering, validation, and security work, and the build pipeline itself becomes difficult to operate at fleet volume.

The cumulative effect is duplication. As shown in Figure 1, the traditional model maintains a separate firmware fork per platform, and each fork repeats the same code changes, the same security review, and the same validation cycle. As fleets span more SKUs, generations, and vendors, that duplication compounds, which can make the firmware operating model slow, expensive, and brittle.

The M-PnP Solution: Full System Modularity Including Firmware

The proposal is to implement the OCP DC-MHS MHS-V3 specification together with the OCP M-PnP Interoperability Design Specification, extending modularity from the hardware across the lifecycle. In concrete terms, this means implementing standardized configuration files, including OpenBMC configuration files and BIOS firmware builds, which describe the system, so that original design manufacturers (ODMs) and system integrators configure their system-specific implementation rather than re-developing firmware for it.

Figure 1: Replacing per-platform firmware forks with a single qualified image driven by FRU/JSON configuration



From Code to Configuration

The central move is to shift variation from source code into configuration. One common firmware codebase replaces the repeated, platform-specific ports that previously had to be carried across silicon vendors, HPMs, and BMC SoCs. Each new platform is described by configuration descriptors rather than by new code, so onboarding a platform can require less engineering, less validation, and less security review than it would under a per-platform firmware model. A qualified BMC image that can consume multiple FRU configurations shortens requalification and lets the fleet scale faster.

This establishes a baseline plug-and-play posture: minimal code and configuration work is needed to describe the baseline functionality of a platform and reach a basic level of operation, and integrators then supply only the configuration specific to their implementation.

Four Responsibilities, One Platform-Agnostic Image

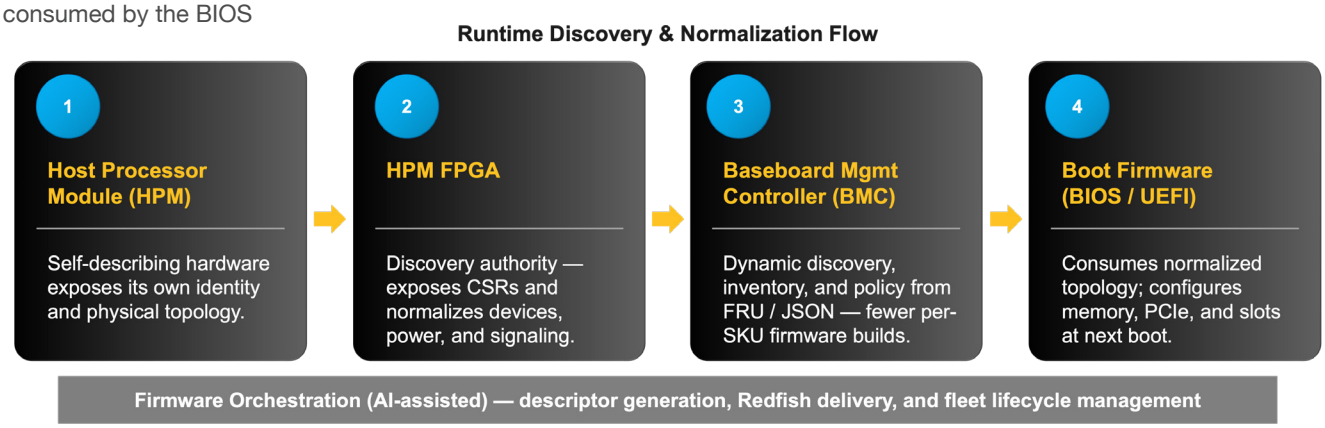
M-PnP works by distributing a small number of clear responsibilities across the platform so that firmware components need less hardcoded knowledge of the system. Table 1 summarizes that division of labor and the DC-MHS value each element delivers.

Table 1: Distribution of responsibilities under the M-PnP architecture

Architecture element	Primary responsibility	DC-MHS value delivered
Host Processor Module (HPM)	Exposes self-describing hardware via FRU, FPGA, and JSON	Vendor-agnostic, swappable compute modules
HPM FPGA	Discovery authority; CSR exposure; power and device normalization	Reduced firmware complexity and consistent hardware abstraction
Baseboard Management Controller (BMC)	Dynamic discovery, inventory, sensor management, and policy application	No per-SKU rebuilds; late binding of hardware configuration
Boot Firmware (BIOS / UEFI)	Consumes normalized topology and configures the system at boot	Runtime adaptation without hardcoded platform knowledge

These responsibilities chain together into a single discovery-to-boot flow, illustrated in Figure 2. In the M-PnP design, hardware reports itself, the FPGA normalizes it, the BMC applies configuration and policy, and the BIOS configures the system at boot, all from a single qualified image.

Figure 2: The discovery-to-boot flow: self-describing hardware is normalized by the FPGA, applied by the BMC, and consumed by the BIOS



Self-Describing Hardware and the FPGA Discovery Layer

The keystone of the model is hardware that can describe its identity, inventory, and topology for firmware to consume. The HPM exposes that information through FRU data, platform descriptors, and an on-module FPGA that serves as the discovery authority. The FPGA exposes platform control and status registers and normalizes device, power, and signaling differences into a consistent abstraction that the BMC and BIOS can use regardless of the underlying processor, board implementation, or vendor-specific wiring. By absorbing platform-specific differences in reconfigurable logic instead of firmware source, the FPGA enables configuration to be bound at deployment or boot rather than at compile time.

Configuration Descriptors: FRU and JSON

Platform variation is carried by machine-readable descriptors. FRU and HPM/platform JSON files describe inventory, identity, and topology. The BMC consumes these descriptors to discover inventory and apply policy, while the BIOS consumes the normalized topology to configure memory, PCIe® lanes, and slots at boot. The effect is that a platform change becomes a JSON-defined configuration rather than a new firmware port. AI-generated FRU JSON is intended to compress onboarding further by automating the inventory description and reducing manual errors.

A Modular Firmware Stack

The firmware stack pairs a modular UEFI BIOS (MUEFI) with OpenBMC, both built to be parameterized by descriptors rather than recompiled per platform. In the M-PnP design, a single MUEFI BIOS can serve one SoC across several different SKUs, and a single qualified BMC image can boot multiple FRU-defined platforms, reducing the number of independently compiled images that must be maintained.

AI-Assisted Orchestration

Orchestration ties the model together. AI tooling is intended to generate and update platform descriptor files faster than manual authoring and with fewer syntax and inventory mistakes; the orchestration layer then delivers configuration to the platform, for example over Redfish, and manages firmware lifecycle across the fleet. This is where descriptor generation, distribution, and fleet management are coordinated, turning a library of qualified components plus per-platform descriptors into running systems.

Security Considerations

The M-PnP architecture extends modularity into the firmware layer by enabling platform configuration through self-describing hardware, FPGA-based discovery, and machine-readable descriptors. While this approach reduces firmware image proliferation and simplifies platform interoperability, it also introduces new security considerations that require further study and architectural definition.

Key areas under evaluation include:

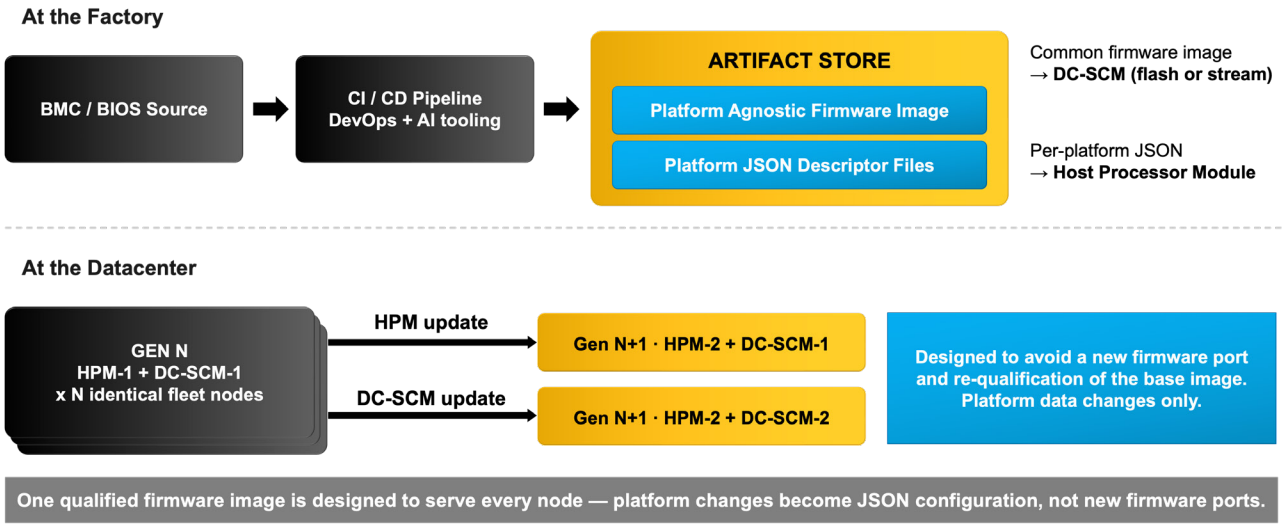
- Protection, authentication, and integrity validation of FRU and JSON platform descriptors.
- Establishing trust in the FPGA discovery layer, including assurance of FPGA configuration and topology reporting.
- Secure binding between hardware identity and platform configuration data.
- Preservation of Root of Trust (RoT) and Secure Boot requirements across modular HPM and DC-SCM combinations.
- Measurement and attestation of dynamically discovered platform configurations.
- Secure provisioning, distribution, and lifecycle management of configuration descriptors.

These considerations are recognized as important aspects of a configuration-driven firmware model. The current work focuses on demonstrating interoperability, modularity, and operational scalability. Security architecture, trust boundaries, attestation mechanisms, and descriptor protection strategies are active areas of investigation.

Deployment Model: Multi-Generation, Multi-Platform

M-PnP uses HPM JSON and FRU JSON to configure a common boot and management firmware image across qualified HPM and DC-SCM combinations. The outcome is a single firmware codebase spanning many platform deployments.

Figure 3: From factory to fleet: one platform-agnostic image plus per-platform JSON descriptors adapt to new modules without new firmware ports.



At the Factory

BMC and BIOS sources flow through a CI/CD pipeline, for example, Azure DevOps, augmented with AI tooling, producing two artifacts in an artifact store: a platform-agnostic firmware image and a set of platform JSON descriptor files. The common firmware is flashed, or streamed, to the DC-SCM, while per-platform JSON is delivered to the HPM. Because the boot firmware can be streamed at deployment, firmware-less shipment becomes possible: the same firmware serves multiple SKUs, and the platform-agnostic firmware flashed on DC-SCMs simply reads its JSON configuration from the HPM.

At the Data Center

In the field, the same qualified image is designed to adapt to change without new ports. Two common transitions illustrate the point:

- **HPM Update (Gen N → Gen N+1)**: a new HPM arrives preloaded with its platform JSON. The platform-agnostic DC-SCM firmware reads the configuration, and a new DC-SCM should not be required to support the next generation.
- **DC-SCM Update:** BMC SoC updates are supported with a single DC-SCM image. The HPM carries the same configuration, and the BMC firmware can be loaded at the factory or streamed at deployment, while the boot firmware remains the same or is likewise streamed.

The net effect is that multi-generation platforms can run on the same firmware, and because there is a single image to reason about, security management and audits can be simplified.

The Demonstration

To make the model concrete, Lattice and AMI, a Lattice Company, are preparing a demonstration that shows plug-and-play behavior at the system firmware level on a DC-MHS platform. The sequence is as follows:

- Establish a baseline boot on a DC-MHS system.
- Inject a new HPM JSON descriptor through the orchestration layer, delivered via Redfish.
- The BMC dynamically discovers and reconfigures the hardware from the new descriptor.
- The FPGA exposes the updated topology through its control and status registers.
- The BMC reads the CSR and manages the devices behind the FPGA on the HPM.
- At the next boot, the BIOS adapts memory, PCIe, and slot configuration to match.
- New devices are expected to appear and be brought into service, with no firmware rebuilds.

The intended takeaway is that the same qualified image can absorb a hardware change as a configuration event rather than as an engineering project, the defining characteristic of plug-and-play at the firmware level.

Business Case and Value Proposition

Customer Value

For the operator, the model converts firmware from a per-platform cost center into a scalable, shared asset. The headline outcomes are summarized in Table 2.

Table 2: Customer value proposition

Outcome	What It Can Deliver
Projected 60–70% less deployment friction	Fewer hand-offs and less platform-specific glue work to bring a system online
Projected up to 3X faster platform readiness	New platforms reach a functional state in a fraction of the time
Projected 50–70% fewer firmware updates	A shared base means fewer divergent images to patch and re-release
Potentially stronger security posture	One qualified base can narrow the attack surface and simplify audits and patching
SaaS-ready fleet management	Configuration-driven firmware can enable continuous, fleet-scale management delivered as a service

Productivity Gain Drivers

Five mechanisms drive productivity improvement:

- Port once, configure many: a single codebase across silicon vendors, HPMs, and BMC SoCs can remove repeated board-porting effort.
- AI-generated platform files: tooling creates and updates descriptor files faster than manual authoring, with fewer syntax and inventory mistakes.
- Qualified BMC image: one qualified image boots multiple FRU-defined platforms, reducing requalification cycles.
- Less security fallout: fewer divergent code branches can mean fewer new vulnerabilities, easier patching, and a more stable Root of Trust (RoT) implementation.
- Descriptor-driven test automation: qualification tests are automated from the modified platform files, enabling faster porting cycles.

Quantitative Productivity Model

Table 3 models the engineering effort for a representative program of ten platform variants spanning multiple silicon vendors and HPM combinations. The logic is straightforward: a common codebase eliminates repeated source modifications, so only the descriptors change per platform.

Table 3: Directional engineering-effort model (basis: 10 platform variants). Figures are planning estimates for discussion, not commitments.

Workstream	Traditional	M-PnP	Improvement
BIOS / BMC platform porting	100 wks.	30 wks.	up to 70%
Validation & regression	60 wks.	30 wks.	up to 50%
Security validation	30 wks.	8 wks.	up to 73%
Platform configuration updates	20 wks.	3 wks.	up to 85%
Total engineering effort	**210 wks.**	**71 wks.**	**up to 66%**

Strategic Position: Time-to-market, ROI, and TCO

Taken together, these drivers can contribute to faster time-to-market, improved return on investment, and lower total cost of ownership compared with a per-platform firmware model. Shifting platform development “left” can reduce development cost up front, while plug-and-play and interoperability can drive long-term TCO benefits across fleet services. A common firmware base also lays the groundwork to manage firmware continuously, at fleet scale, as a service.

Why It Matters: A New Firmware Operating Model at Hyperscale

Up to 50–70%
less platform porting effort

Up to 30–50%
faster validation and qualification

Up to 60–85%
faster time-to-market

The significance of M-PnP is that one qualified BMC/BIOS image is designed to scale across SKUs, generations, and vendors. At hyperscale, the projected impact is substantial: roughly 50–70% less platform porting effort, 30–50% faster validation and qualification, and 60–85% faster time-to-market. These are the levers that determine how quickly an operator can field new silicon and how cheaply it can sustain the fleet that results.

The benefits compound beyond raw engineering hours. A common firmware base can narrow the attack surface and simplify audits and patching, helping strengthen security posture as the fleet grows rather than weakening it. Because the firmware is configuration-driven, it can become a natural foundation for continuous, SaaS-style fleet firmware management. And the same model can optionally extend across heterogeneous architectures and silicon vendors, giving operators room to diversify supply without re-architecting their firmware.

The shift is that firmware becomes configuration-driven rather than board-driven. Hardware describes itself, firmware adapts at runtime, and no rebuilds are required.

A modular foundation that lowers cost, risk, and friction as platforms scale.

Conclusion

DC-MHS solved the mechanical and electrical problem of the modular data center; M-PnP closes the firmware gap that remained. By moving variation from code to configuration, anchoring discovery in an on-module FPGA, and orchestrating descriptors with AI assistance, operators can run a single qualified firmware image across a diverse, multi-generation fleet. The payoff is faster time-to-market, materially lower engineering effort, a potentially stronger security posture, and a credible path to SaaS-style fleet firmware management.

The building blocks, MHS-V3, the M-PnP Interoperability Design Specification, OpenBMC and MUEFI configuration, FPGA-based discovery, and AI-assisted orchestration exist today. What changes is the operating model: hardware that describes itself, and firmware that adapts to it. That is the difference between plug-and-play that stops at the connector and plug-and-play that reaches all the way to the running system.

Glossary

Term	Definition
BIOS / UEFI	Boot firmware that initializes platform hardware and hands control to the operating system
BMC	Baseboard Management Controller, the service processor that manages and monitors a server independently of the host operating system
CSR	Control and Status Register, an FPGA register that reports and controls device state
DC-MHS	Datacenter Modular Hardware System, the OCP family of specifications standardizing modular server building blocks
DC-SCM	Datacenter Secure Control Module, the standardized module carrying the BMC and security / root-of-trust functions
FPGA	Field-Programmable Gate Array, reconfigurable logic used here as the on-module discovery authority and hardware-abstraction layer
FRU	Field-Replaceable Unit data, standardized inventory and identity information describing a hardware component
HPM	Host Processor Module, the standardized compute module hosting the CPU/SoC and memory
JSON descriptor	Machine-readable HPM / platform configuration file that parameterizes firmware for a specific platform
M-PnP	Modular Plug-and-Play, the configuration-driven interoperability model described in this paper
MHS-V3	The version of the OCP Modular System specification targeted by this implementation
MUEFI	Modular UEFI BIOS designed to be parameterized by descriptors rather than recompiled per platform
OCP	Open Compute Project, the industry body defining open data center hardware standards
OpenBMC	Open-source BMC firmware stack, configured here through standardized configuration files
Redfish	Standard RESTful interface used to deliver configuration and manage platforms
RoT	Root of Trust, the hardware-anchored basis for firmware integrity and secure boot
SKU	Stock-Keeping Unit, a specific, orderable hardware configuration variant
TTM / ROI / TCO	Time-to-Market / Return on Investment / Total Cost of Ownership

Productivity Model, Basis, and Assumptions

The figures in the Quantitative Productivity Model are directional planning estimates intended to support proposal discussion; they are not commitments. They rest on the following basis:

- Scope of ten platform variants spanning multiple silicon vendors and HPM combinations
- A single common firmware codebase, so that platform onboarding changes descriptors rather than source
- A worked example for the BIOS layer of one SoC supporting five different SKUs from one MUEFI build
- Improvement percentages compare the traditional per-platform model against the M-PnP configuration-driven model on identical scope

Actual results will vary with program scope, the maturity of the descriptor tooling, and the diversity of the silicon and module portfolio. The model is offered as a planning aid, and the assumptions above should be revisited against each operator's specific portfolio.



READY TO LEARN MORE?

To learn more about Lattice low power FPGA-based solutions for compute, communications, industrial, and embedded applications, visit www.latticesemi.com or contact us at www.latticesemi.com/contact or www.latticesemi.com/buy.

TECHNICAL SUPPORT ASSISTANCE

Submit a technical support case through **www.latticesemi.com/techsupport**.
For frequently asked questions, please refer to the Lattice Answer Database at **www.latticesemi.com/Support/AnswerDatabase.**

© 2026 Lattice Semiconductor Corporation and affiliates. All rights reserved. Lattice Semiconductor, the Lattice Semiconductor logo, Lattice Nexus, and Lattice Avant are trademarks and/or registered trademarks of Lattice Semiconductor and affiliates in the U.S. and other countries. Other company and product names may be trademarks of the respective owners with which they are associated.

V.1 WP0060