VIEW POINT

THE POWER OF INTEGRATION: UNLOCKING INNOVATION THROUGH SOFTWARE-DEFINED HARDWARE

Infosys®
Navigate your next

The boundaries between software and hardware have blurred, heralding the era of "Software-Defined Everything" (SDE). This paradigm shift places software at the helm, steering the functionality and evolution of hardware. From autonomous vehicles to medical devices and virtualized networks, the symbiosis between software and hardware is driving breakthrough innovations. Adopting a software-first approach in hardware integration isn't just a choice anymore—it's the competitive edge.

This article explores the indispensable role of software-first strategies, the advantages of layered architectures, the power of hypervisors, and the long-term value of proactive feature upgrade strategies. These practices, honed by industry leaders, serve as a compass for product designers and developers navigating this dynamic landscape.

Why the Software-First Approach Is the Future

Historically, a hardware-first methodology ruled the landscape—hardware was designed first, with software developed later to support existing capabilities. This model restricted innovation, locking products into fixed functionality. Enter the software-defined era, where software orchestrates hardware's capabilities free from physical constraints. The benefits are profound: greater flexibility, faster innovation cycles, and the ability to create products that can evolve post-deployment.

For example, software-defined networks (SDNs) now dynamically manage traffic by abstracting and optimizing hardware interconnectivity. Similarly, in the automotive sector, software-defined vehicles utilize centralized hardware waiting to be activated through software updates, such as Tesla's Autopilot or Acceleration Boost features. This paradigm prioritizes adaptability, future-proofing investments and delivering greater value to end-users.

Layered Software Architecture—The Blueprint for Seamless Integration

Powering this software-first approach is the application of layered software architecture. This time-tested model breaks the software stack into distinct, interdependent layers, ensuring modularity, scalability, and efficiency.

1. The Hardware Abstraction Layer (HAL)

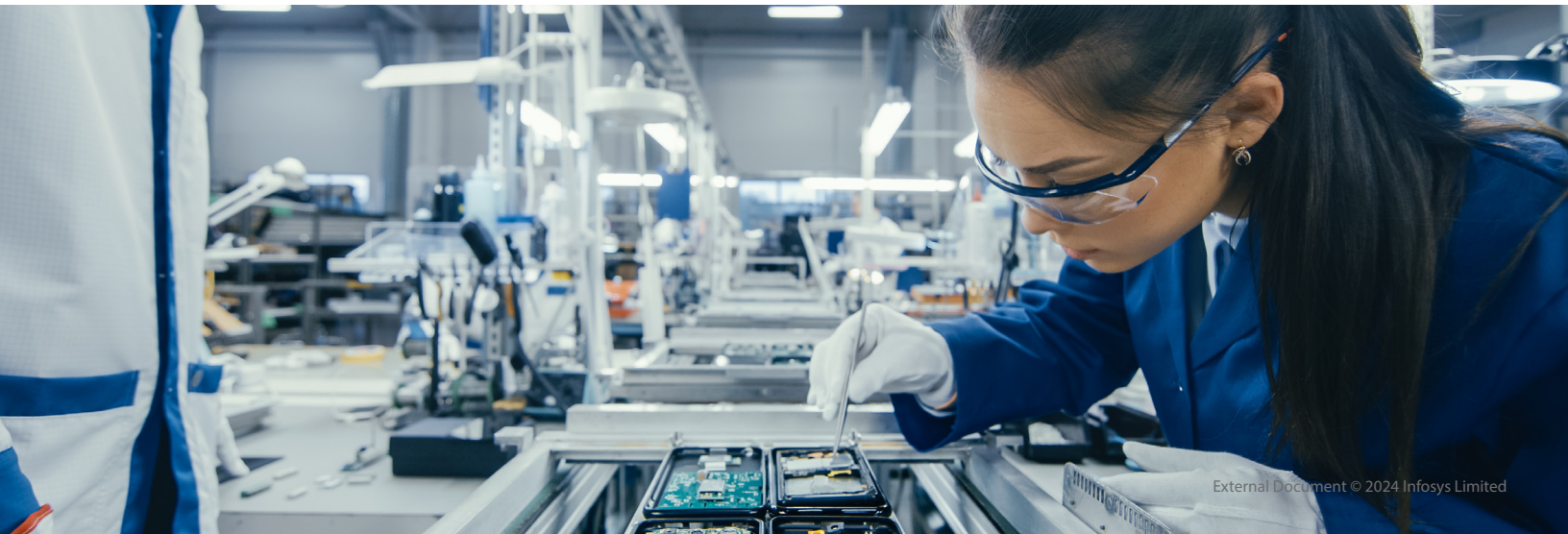
- The HAL serves as the bridge between hardware and software, abstracting hardware complexity and enabling reusability. HAL provides a generic interface consumed by software components above it, reducing dependency on specific hardware implementations. For optimal results, developers should embed industry standards like OpenMAX DL (for multimedia applications) or AUTOSAR's ECU Abstraction Layer (in the automotive industry). These standards streamline hardware-software integration while enhancing interoperability across diverse systems.
- Consider an accelerometer sensor as an example. By abstracting its memory addresses and control registers into readable, symbolic representations, HAL enables developers to seamlessly switch between hardware implementations or vendors without affecting higher-level software components.

2. The Middleware Layer

- Positioned above HAL, the middleware layer standardizes interactions across operating systems, creating uniformity while providing essential services like security, persistent storage, logging, and inter-service communication. Integration with protocols like OpenMAX IL or standardized middleware like AUTOSAR facilitates building robust, cross-platform solutions.
- For instance, Android's middleware enables fluid app development regardless of device hardware. By standardizing these functionalities, developers focus on innovation rather than battling OS and hardware disparities.

3. The Application Layer

- At the apex lies the application layer, where end-user functionality is realized through intuitive interfaces, business logic, and intelligent algorithms. Whether it's image processing in mobile devices or Artificial Intelligence-driven insights from hardware, this layer turns raw data into actionable outcomes, directly enhancing user experiences.
- By leveraging this layered approach, developers enhance adaptability and resilience, creating ecosystems where hardware upgrades, replacements, or enhancements don't disrupt the entire system.



Hypervisors—Pioneering a Multidimensional Approach

Today's systems, particularly in domains requiring versatile hardware stacks, demand multi-operating system support. Hypervisors rise to the challenge, enabling multiple virtual machines to coexist on a single hardware platform.

Take the automotive industry. The intelligent cockpit of a modern vehicle may demand diverse operating systems such as QNX for vehicle control, Android for in-car entertainment, and Linux for data processing. A hypervisor ensures these disparate environments operate seamlessly alongside each other, optimizing hardware utilization without conflicts.

By emulating multiple hardware configurations on the same device, hypervisors also future-proof product development. Teams can test, deploy, and support updates for isolated features without compromising the broader system.

Strategic Feature Upgrades—Maximizing Tomorrow's Potential

Long product lifecycles demand adaptability. A forward-thinking feature upgrade strategy isn't just about staying relevant—it's about building deliberate pathways for future functionality.

Some leading practices include:

1. Hardware Readiness for Future Needs

Many industry leaders pre-install advanced hardware in devices, enabling functionality to be activated later through software updates. Tesla's approach to releasing Autopilot features via software, long after users purchase cars, exemplifies this strategy.

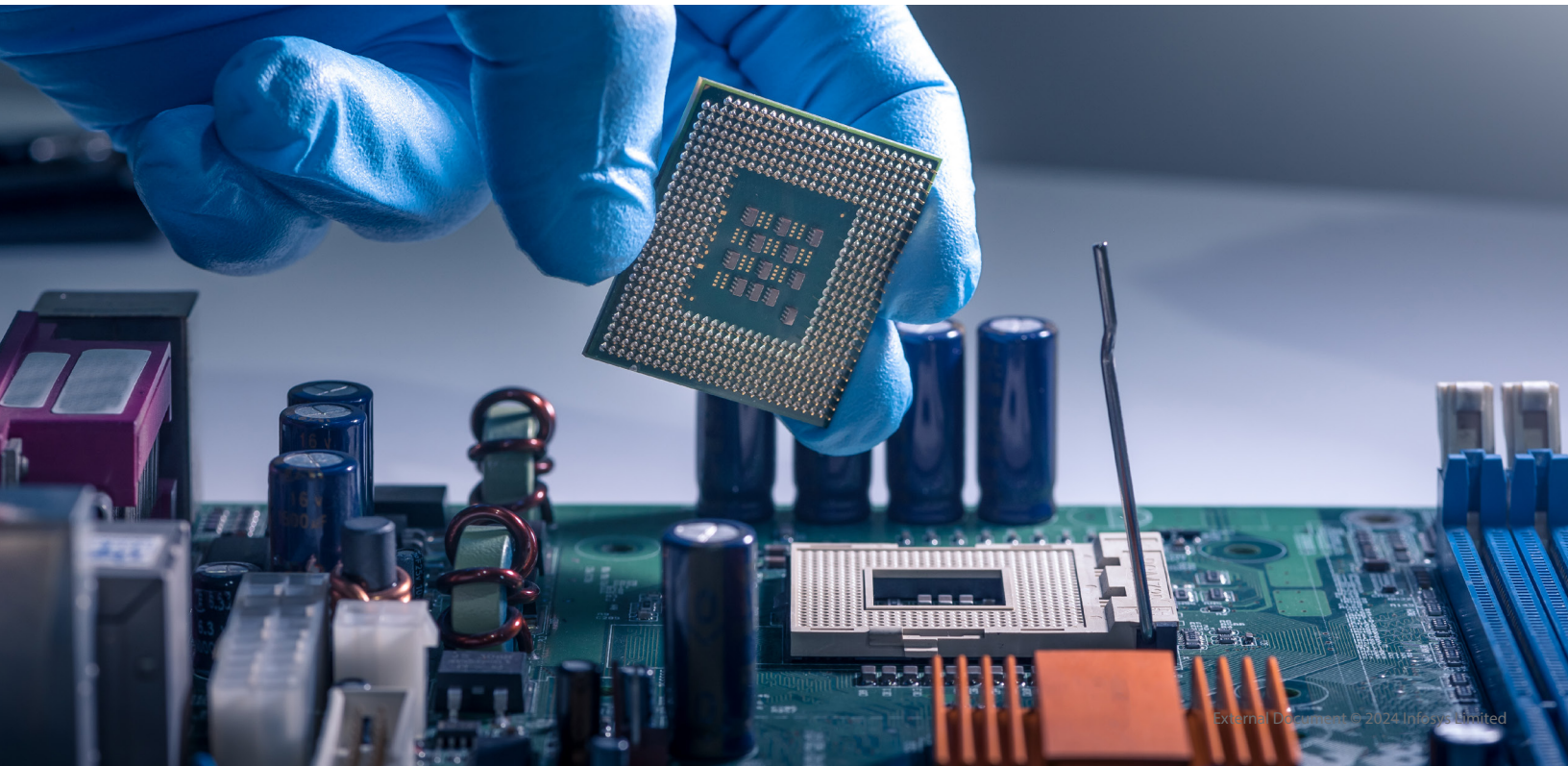
2. Software Modularity for Configured Delivery

Software developed with modularity ensures that optional features can be enabled as needed. For instance, cardiac implant programmers ship with standard configurations but can be tailored through activation approaches set during installation.

3. Remote Software Upgrades

Seamless, secure, and remote software updates have revolutionized post-deployment usability. Segregating core intellectual property (deployed via proprietary servers) from generic components (distributed through app stores) ensures security while streamlining user access.

Organizations employing these strategies maximize their products' longevity, enhance customer satisfaction, and open new revenue streams through subscription-based upgrades or add-ons.



Investing in Automation for Reliability

Software-hardware integration demands rigorous validation. Automated testing solutions like CppuTest accelerate error detection, especially for middleware and application layers, reducing reliance on physical hardware during early development stages. Custom tools simulating HAL interfaces further help developers uncover issues before hardware verification, shortening overall timelines. Such pragmatic approaches make automation an indispensable lever for achieving quality and efficiency.

The Road Ahead Integrating the Best

From software-defined networks enabling dynamic connectivity to automotive systems delivering self-driving capabilities, the marriage of software and hardware is not just transforming industries—it's reshaping expectations. These innovations are fueled by strategic architectures, robust integration frameworks, and an unrelenting commitment to adaptability.

Product designers and developers bear the responsibility of translating these best practices into tangible solutions.

By adopting software-first principles, leveraging layered architectures, integrating hypervisors, and enacting forward-looking upgrade strategies, they can drive technological evolution, unlocking products that stand the test of time.

At its core, the software-first approach is more than a method—it's a mindset. Through careful planning, seamless engineering, and a willingness to innovate, it empowers us to "Navigate our Next."

References:

1. Embedded. (n.d.). Why a software-defined approach is the future for embedded and IoT. Retrieved from https://www.embedded.com/why-a-software-defined-approach-is-the-future-for-embedded-and-iot/
2. Deloitte. (2021). Software defines vehicles. Retrieved from https://www2.deloitte.com/content/dam/Deloitte/cn/Documents/consumer-business/deloitte-cn-cb-software-defines-vehicles-en-210225.pdf
3. Qualcomm. (n.d.). Snapdragon 8+ Gen 1 mobile platform. Retrieved from https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-plus-gen-1-mobile-platform
4. AUTOSAR. (n.d.). Standards – Adaptive Platform. Retrieved from https://www.autosar.org/standards/adaptive-platform
5. Android Developers. (n.d.). Platform architecture overview. Retrieved from https://developer.android.com/guide/platform
6. Khronos Group. (n.d.). OpenMAX IL (Integration Layer). Retrieved from https://www.khronos.org/openmaxil
7. Khronos Group. (n.d.). OpenMAX AL (Application Layer). Retrieved from https://www.khronos.org/openmaxal
8. CppuTest. (n.d.). CppuTest manual. Retrieved from https://cpputest.github.io/manual.html
9. AUTOSAR. (n.d.). Standards – Classic Platform. Retrieved from https://www.autosar.org/standards/classic-platform
10. ITU-T. (n.d.). Software-defined networking (SDN). Retrieved from https://www.itu.int/en/ITU-T/sdn/Pages/default.aspx
11. U.S. Food and Drug Administration (FDA). (n.d.). Regulatory information. Retrieved from https://www.fda.gov/regulatory-information
12. Khronos Group. (n.d.). OpenMAX DL (Development Layer). Retrieved from https://www.khronos.org/openmax/dl
13. AUTOSAR Today. (n.d.). Guide to the Classic AUTOSAR architecture. Retrieved from https://www.autosartoday.com/posts/guide_to_the_classic_autosar_architecture
14. Khronos Group. (n.d.). OpenCL Overview. Retrieved from https://www.khronos.org/api/opencl

About the Author

Balaji Lakshmi Narasimhan

Senior Principal Technology Architect, Engineering Services, Infosys

For more information, contact askus@infosys.com

Infosys®
Navigate your next