2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Accelerating Bare Metal Driver Development with Linux Drivers and System Verilog DPI-C

Suchir Gupta, Amit Sharma, Suneetha Suryadevara, Vishnu Vardhan Reddy

Synopsys Inc

Introduction

The Challenge

Bare metal driver development for pre-silicon IPs is time-consuming (6-8 months), requires extensive expertise, and involves limited code reuse.
Need to understand the software specification, develop full framework, and then verify IP driver against specifications.

The Opportunity with Mature Linux drivers

Linux drivers exist for virtually all peripheral hardware with years of development and testing.

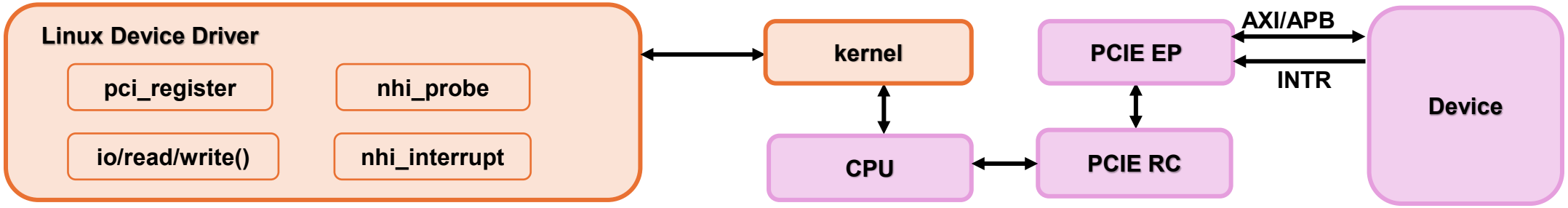
The Problem in using Mature Linux drivers

Linux drivers are tightly integrated with underlying operating system and can’t be used as is bare metal.

Devised a Methodology to convert Mature Linux drivers to Bare Metal Driver.

Tested the methodology for developing USB4 connection manager (Bare metal driver) to manage USB4 router (Bare metal) Zebu transactor.
Methodology applicable to convert any Linux driver to bare metal driver.

Typical Linux Driver Interaction : USB4



System Setup

- Linux kernel, CPU, and PCIE connect device driver with device.
- Kernel enumerates the PCIE device and invokes the device driver's `probe` function.

Memory Write/Read

- Driver issues ioread()/iowrite() : Kernel –> CPU –> PCIE RC –> PCIE EP –> Translator –> Device.

Interrupt

- Device triggers interrupt: line –> PCIE EP –> PCIE RC –> CPU –> Kernel –> INTR handler

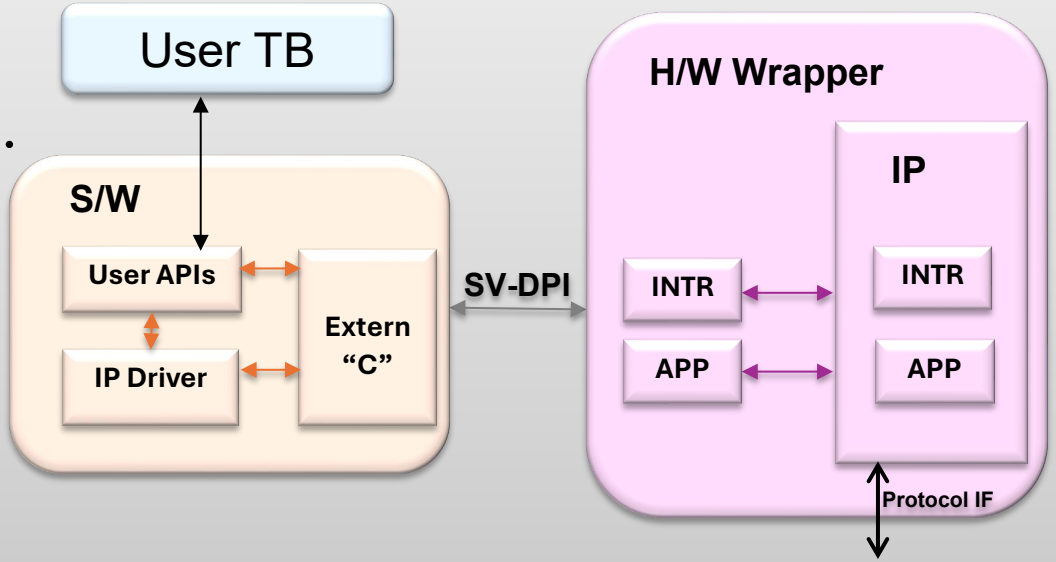
Transactor Overview for HW-SW communication

HW

- Consists of SV-DPI methods, and IP.
- IP consists of protocol, APP, and INTR interface.

SW

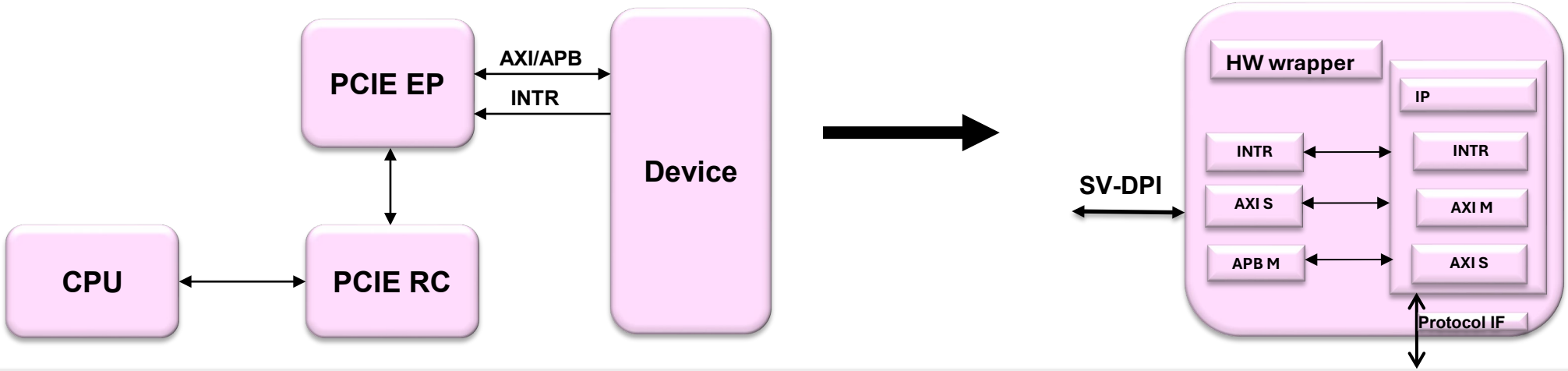
- IP driver implements SW IP specification.
- User APIs let user control transactor
- extern “C” functions interact with HW.



The Linux driver has full specifications implemented but is tightly integrated with underlying OS.

Requirement : From a USB4 standpoint, for the transactor to be leveraged for SOC verfication, the USB4 device driver needs to interact with USB4 router through extern “C” functions without any dependency on Linux, CPU, and PCIE components.

HW methodology to convert Linux driver to bare metal driver.



Develop HW wrapper -- ***Replaces CPU, PCIE RC, and PCIE EP components***

- Encapsulate IP
- Develop Application interface counterparts in HW wrapper.
- Develop interrupt handler
- Develop SV-DPI methods to transport application + interrupt data SW.

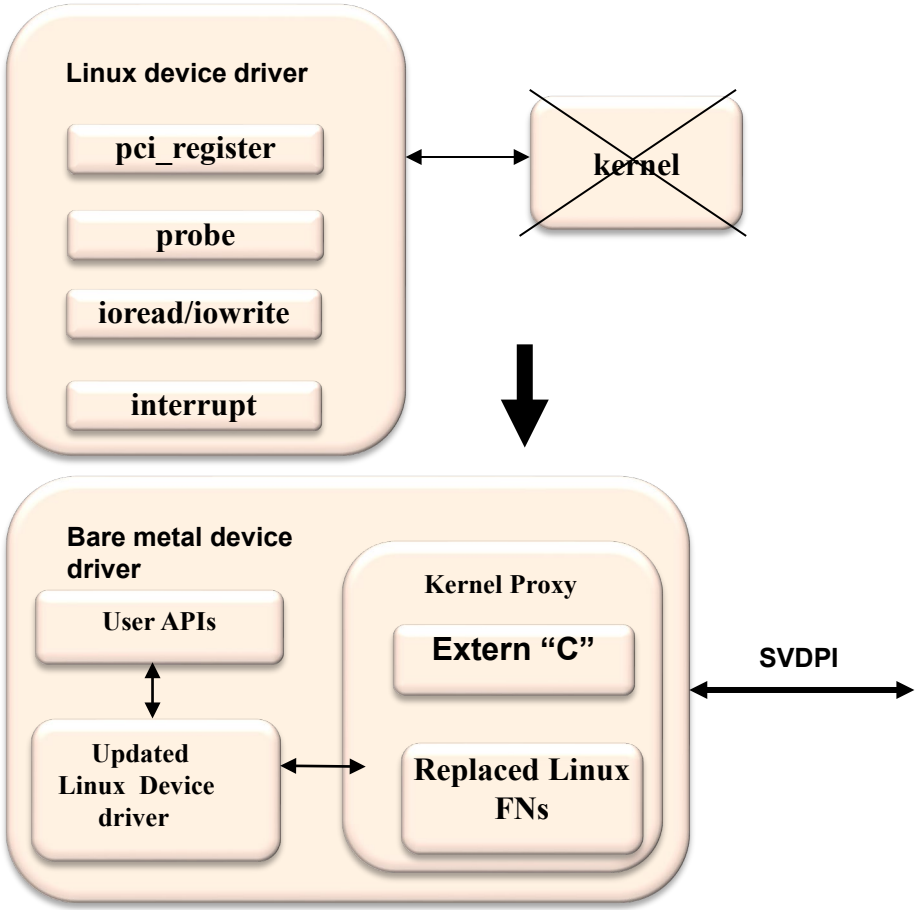
SW methodology to convert Linux driver to bare metal driver.

Analyze Linux Device Driver

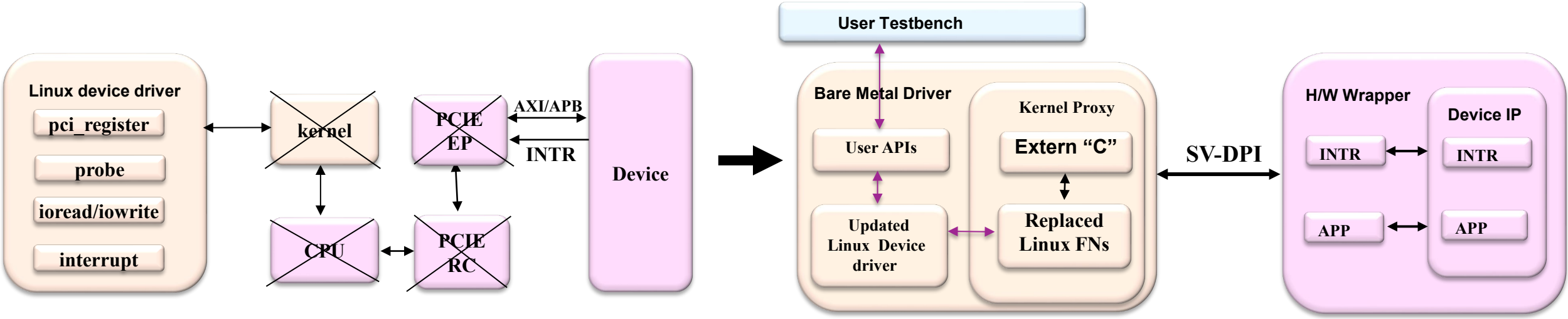
- Download Linux Kernel source
- Identify Kernel dependencies.
- Classify each dependency: re-implement, stub, or remove.

Develop kernel proxy - **Replaces Linux Kernel**

- Redefine initialization (probe), memory (read/write), and interrupt APIs.
- Implement extern “C” functions to create SV-DPI bridge. The bridge interacts with H/W wrapper.
- Provide user level initialization APIs to control Device Driver.



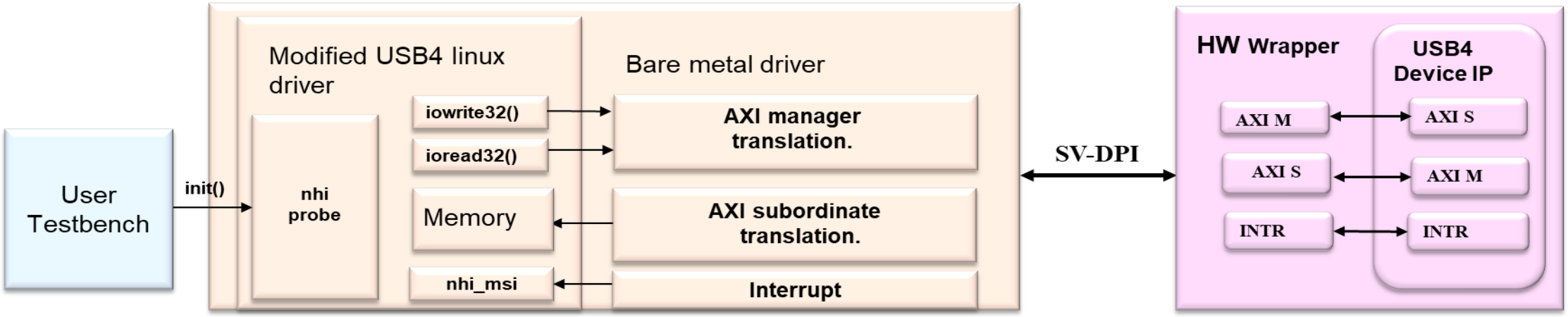
Methodology Summary



Overall flow describing the bare metal driver development using Linux drivers.

1. User APIs invoke **probe** rather than Kernel proxy.
2. IO operations: Updated Linux driver > Replaced Linux functions > extern "C" > DPI > Device IP.
3. Interrupt: IP > DPI > extern "C" > Replaced Linux Fn > Updated Linux driver

Case Study: USB4



USB4 bare metal driver development using Thunderbolt Linux driver SV-DPI

Driver Analysis	Kernel Proxy	H/W Wrapper
• Downloaded Thunderbolt driver • Removed ~150 Linux-specific functions • Redefined key library functions	• Developed init() to call `nhi_probe ()` • Called `nhi_msi() for` USB4 interrupt handler. • Developed AXI translators for `iomemwrite()/iomemread()` • Developed extern “c” functions to connect with H/W wrapper.	• Encapsulated USB4 router. • Developed interrupt, AXI interfaces to interact with USB4 router. • Developed SVDPI-C functions to interact with kernel proxy.

USB4 Case Study Results and Summary

Development Time
6-8 person months to 1 person month (80% reduction).

Code Reused
39K lines of code.

Functional Equivalence to protocol features.

Comprehensive Validation

VCS Simulation
Host & Device Router enumeration, NVM operations tested

Zebu Emulation
Tested NVM Read and Writes at 4Gb/s data rates

Broad Applicability — Multiple Use Cases, methodology applicable to ANY Linux driver – Strategic advantage for hardware verification

HW/SW Co-Verification Integration testing	**Pre-Silicon Debug** Early validation	**SoC Bring-Up** System integration
Emulation Large-scale testing	**Virtual Prototyping** Software development	**System Validation** Complete flows