"System and Platform Debug using the Ubiquitous* USB Type-C Connector"
(19th Sept. '24)

Sankaran Menon, Ph.D.
Chair, IEEE P2929 (Standard for System-Level State Extraction for Functional Validation & Debug)
(My Acknowledgements to Rolf Kuehnis for his collaboration and feedback)

"Learning is the only thing the mind never exhausts, never fears, and never regrets" – *Leonardo da Vinci*

"The beautiful thing about learning is that nobody can take it away from you " – *B.B. King*

*existing or being everywhere at the same

IEEE Central Texas Consultants' Network

IEEE CEDA
Council on Electronic Design Automation

Outline

1. Why Debug?
    1. Relation with Detectives and Medicine
2. What is System Debug and Platform Debug
    1. Hardware/Software/Firmware Debug done
3. What is Open Chassis and Closed Chassis Debug
4. Brief History of USB (Universal Serial Bus)
5. Connectors of Choice for Closed-Chassis and In-Field Debug
    1. Type-C Connector
6. Platform capability for Closed-Chassis and Infield Debug
    1. MIPI I3C®, USB Type-C®, etc.
7. Use for Infield verification and validation as well as for Life-Cycle Management
8. Future platforms (using wireless debug etc.)
9. Summary

Why Debug?

1. Hardware/Software/Firmware Debug:
    1. We will address Hardware debug first
2. Hardware Debug:
    1. First Silicon from Fab ➔ Faster debug can help with faster launch of products resulting in Cha-Ching (Cash for the company)
    2. HW debug of system failures
    3. Debug of Returned parts
3. Need to debug SW and FW bugs
4. First Silicon Debug resulting in Faults, Errors, Failures etc.
    1. Silicon can have inherent defects
    2. Defects can be due to impurities, mask issues, scratches, dust particles etc.
    3. Faults are incorrect state of hardware or software resulting from a physical defect
    4. Error is manifestation of a fault, bit in SRAM is a “0” instead of a “1” etc.
    5. Failure is system level effect of an error, e.g., system hangs, incorrect results etc.
    6. All these failures need to be debugged before releasing the product to customers

Defects ➡ Faults ➡ Errors ➡ Failures

IEEE CEDA
Council on Electronic Design Automation

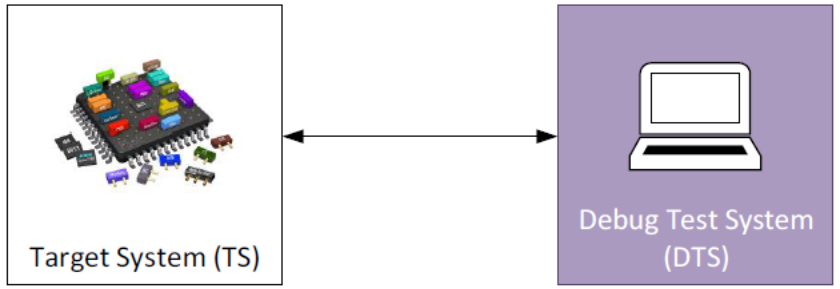
Debug and Parallels with Detectives and Medicine

1. Relation with other areas (e.g., Detectives and Medicine)
    1. Detectives look for clues to solve cases (Debug)
    2. In Medicine, Doctors look for clues by examining patients
2. Tools used in Medicine Vs. Chip Debug:
    1. Non-Intrusive vs. Intrusive
3. Non-Intrusive:
    1. Touch/feel, Stethoscope, BP Apparatus, Blood-Glucose tests, X-Ray imaging, MRI, PET Scans etc.
4. Intrusive:
    1. Surgery, Laparoscopy etc.
    2. Some of them are being done remotely using Da Vinci tools etc.
5. Lot of commonality between them

No.	Features	Medicine	Chip Debug
1	Non-Intrusive	Touch/Feel	Connect Debugger/Observe
2		Stethoscope	Connect Debugger/Control
3		X-Ray Imaging	Enable Scan Observability
4		MRI, PET Scan etc.	Enable Memory Observability
5		EKG, EEG, CGM, MRI, PET Scan etc.	Tracing for boot-flow, low-power, security, memory training etc.
6	Intrusive	Surgery, Laparascopic	Modifying Memory contents
7		Da Vinci Remote Surgery etc.	FIB (Focus ION Beam), Chip Micro-surgery

Process of Debugging

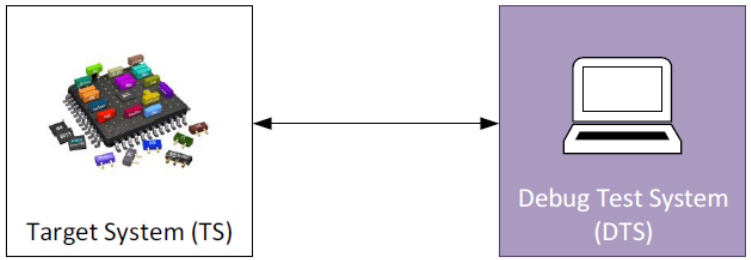
1. Following are the steps involved with Debugging:
    1. Reproduce the bug – recreating the conditions that caused the bug
    2. Locating the bug – finding out the location of the bug in HW, SW or FW
    3. Root Cause the bug – Identifying the bug in HW, SW or FW
    4. Find a fix for the bug – This could be adding a gate/connection, fixing the code etc.
    5. Test to make sure that the fix works – This is make sure that the bug has been fixed
    6. Documentation of the process
        1. SoCs assembled to form a system
    7. DUT or Target TS is mounted on a system board
    8. System board is connected to a Debugger via JTAG port

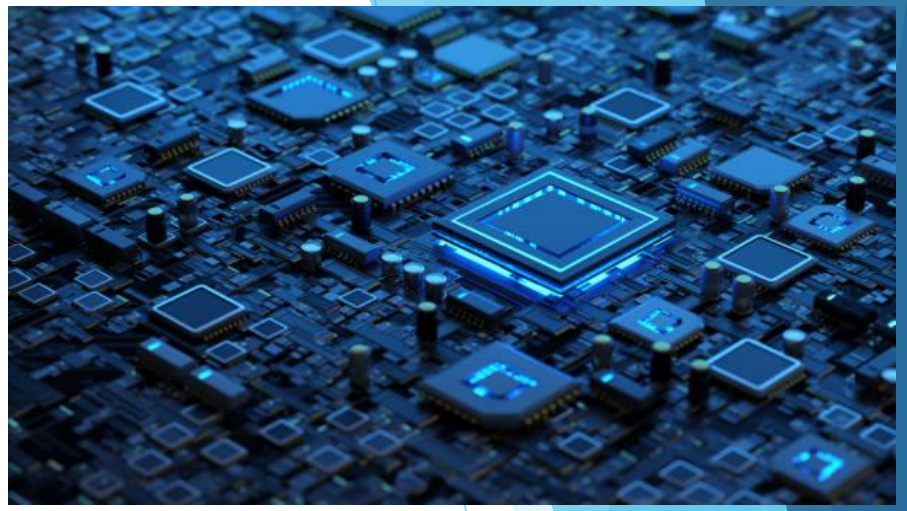


IEEE CEDA
Council on Electronic Design Automation

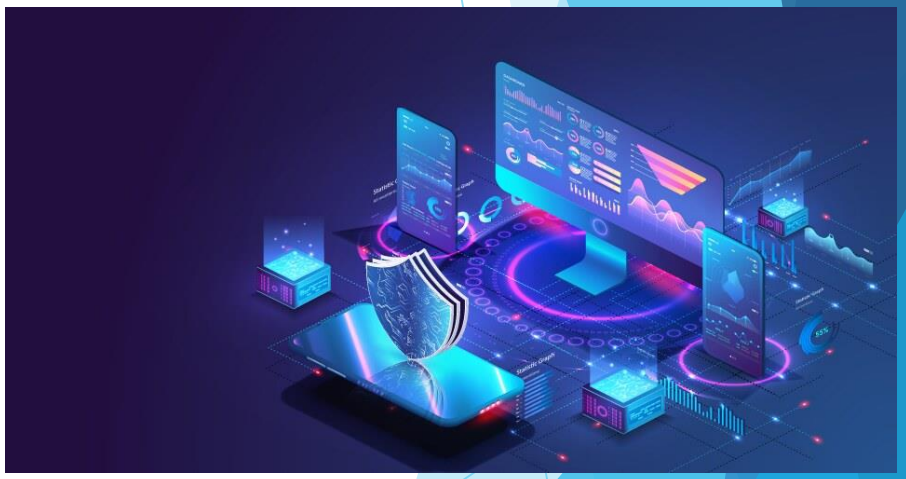
System and Platform Debug

1. System Debug:
    1. System in this context refers to a board with several Chips, Chiplets, etc.
        1. SoCs assembled to form a system
        2. Debugging Chiplets
        3. Systems with multiple SIPs/SoCs
    2. DUT or Target TS is mounted on a system board
    3. System board is connected to a Debugger via JTAG port
2. Platform Debug:
    1. Platform consists of several boards assembled together to form a Platform (HW/SW/FW) e.g., Laptop, Servers, Smartphones etc.
    2. Typically, debug of platforms are done by unscrewing the box and connecting JTAG at the board-level or at the system level
    3. Extremely time-consuming and we will talk about “Closed-Chassis Debug” which is a lot convenient and time-saving as well





System



Platform

Types of Debug

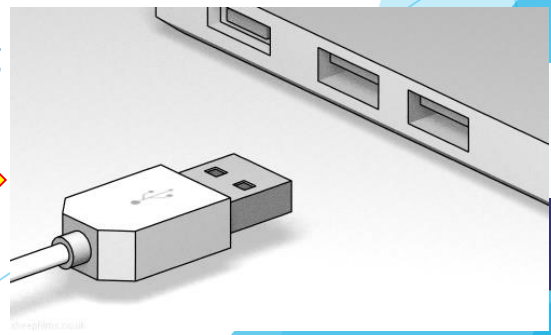
1. Types of bug differ in each of the categories given below:
    1. First Silicon from Fab:
        1. Types of failures can be: Dead Display to not being able to boot to DOA (Dead on Arrival)
        2. Even if first Si is DOA, needs to be diagnosed and root-caused to perform fixes on the next Rev
        3. DOA: Can be due to CPU/GPU not booting to anywhere from Clocks not propagating etc.
    2. HW debug of system failures or field failures:
        1. Marginal design degradation or transistor degradation etc.
        2. Aging related etc.
    3. Debug of Returned Parts (RMA):
        1. This category can have different types of failures
        2. Can be from marginal degradation to catastrophic failures due to temperature related etc.

Brief History of USB (Universal Serial Bus) (1 of 2)

1. USB has evolved over time, in 20 to 30-or-so years…
    1. 1994: Co-invented by Ajay Bhatt (Intel) and the USB-IF comprising Microsoft, LSI, Apple etc.
        1. USB1.0 in 1995 @ 12Mb/s, 1.1 @ 12Mb/s & 1.5Mb/s
            1. Hot Plug
        2. USB2.0 debuted in 2000 @ 480Mb/s
            1. Ease of use, self-powering capabilities & Plug and Play
            2. Backward compatibility of 12Mb/s & 1.5Mb/s
            3. USB OTG (On-The-Go)
        3. USB3.0 in 2008 with Super-Speed (SS) @ 5Gb/s
        4. USB3.1 in 2013 @10Gb/s, USB3.2 in 2017 @10Gb/s, USB4.0 in 2019 @20Gb/s,
        5. USB4 2.0 in 2022 @80Gb/s and asymmetric 120/40 Gb/s

Connectors replaced by USB

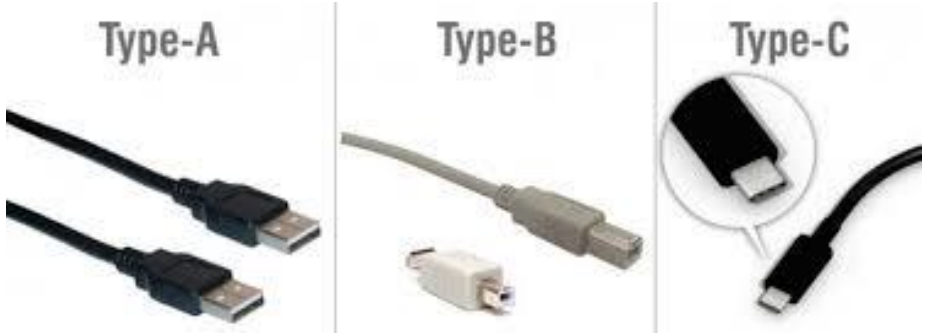
    2. Open Chassis debug is when the SoC is installed on a board, typically validation board, and debugging performed using Debug connector (JTAG, SWD etc.)
    3. The advantage of Open-Chassis is that most of the pins are accessible and observable/controllable
    4. Also, the thermal and RF effects are minimized as opposed to closed-chassis debug
2. USB connector connectivity:
    1. Odds of connecting correctly: <50%
    2. To connect correctly every time: USB Type-C



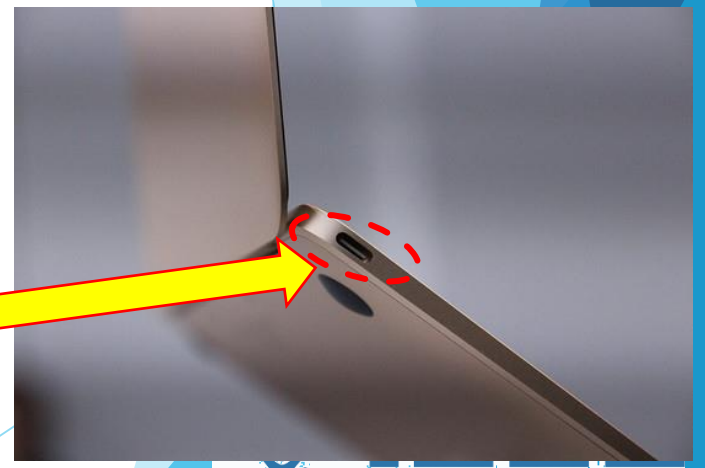
IEEE EDA
n Electronic Design Automation

Brief History of USB (Universal Serial Bus) (2 of 2)

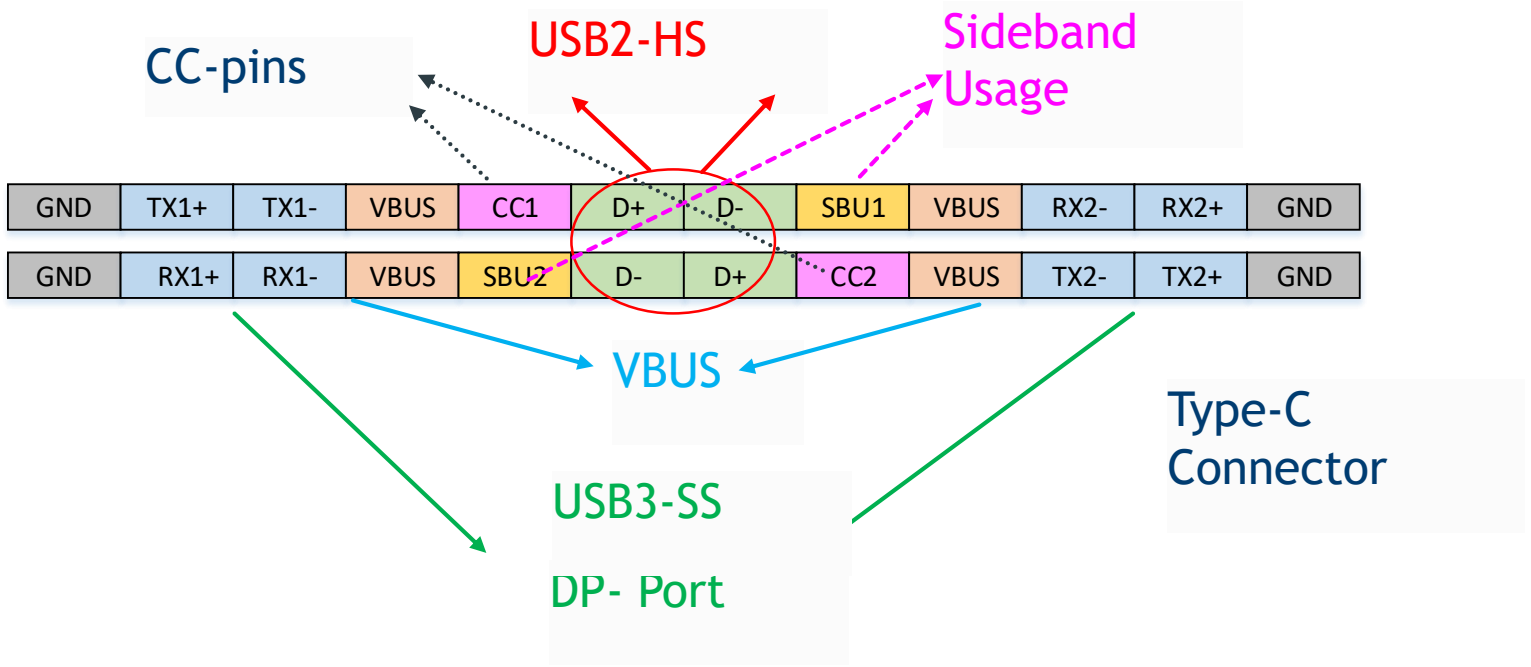
1. USB Connectors
    1. USB Type-A, Type-B
    2. USB Mini, Micro
    3. And now….., USB Type-C



1. USB Type-C Connector
    1. It's Never Upside-Down: Same on both sides
    2. USB3.1 PD: provides of up to 240W of power (48V, 5A)
        1. Type-A was at 5V
            1. USB 3.2 Gen 1 and Gen 2: Supports 5 Gbps and 10 Gbps connections
            2. USB 3.2 Gen 2×2: Supports 20 Gbps connections
            3. USB4: Supports 40 Gbps connections (Intel TBT)
            4. USB4 V2: Supports 80 Gbps (2 lanes) and up to 120Gbps (USB4, asymmetric)
            5. Thunderbolt 3 and 4: Use the USB-C protocol and offer up to 40 Gbps speeds
    3. It's Thin: 0.83x0.26 cm, compared to 1.4x0.65cm
    4. It's versatile: Transfer data, charge devices, video port, hook to external displays
    5. It's Bi-Directional: Can send power both ways, can charge mobile device with a laptop and vice-versa
    6. Only connector: Some just have ONE Connector



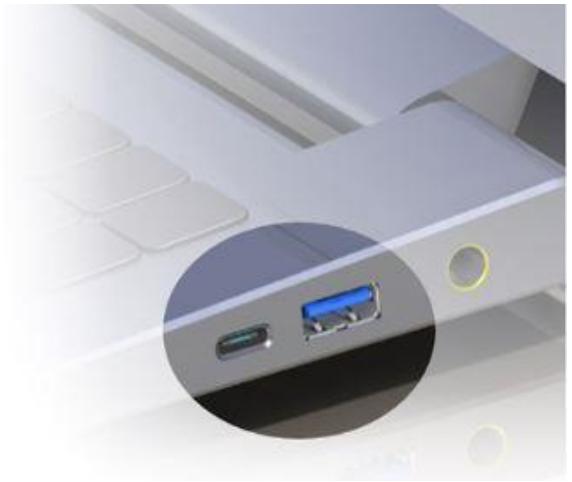
USB Type-C



1. USB Type-C is already becoming the ubiquitous connector in present-day and future devices from laptops to drones/IoTs, due to various advantages:
    1. Used for charging, power-out, connect to keyboard/mice, display, audio etc.
    2. We have extended this to debug as well
2. USB Type-C Pin-out of both plug and receptacle are shown

USB Type-C for Debug

1. As seen from the previous foil, USB Type-C is used for various purposes, such as:
    1. USB2, USB3, Display, Audio
    2. We use it for Debug as well
        1. Making this connector the most widely used connector for multipurpose, both Functional as well as Debug
        2. And “The most Ubiquitous Connector” (best thing since sliced bread)

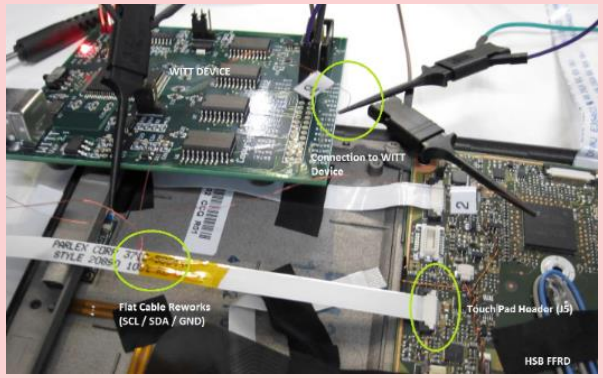


USB Type-C: Ubiquitous Connector
Best Thing since Sliced Bread☺

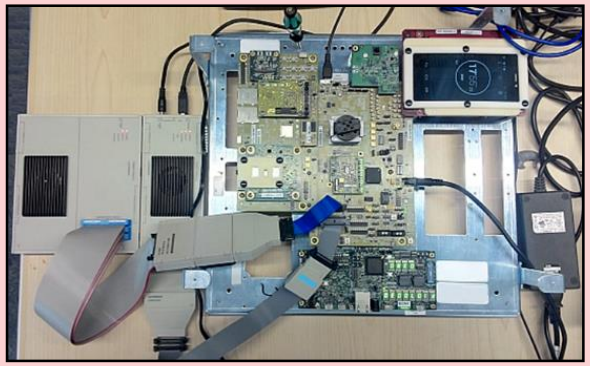
Open Chassis and Closed Chassis Debug

1. Open Chassis Debug:
    1. Open Chassis debug is when the SoC is installed on a board, typically validation board, and debugging performed using Debug connector
    2. The advantage of Open-Chassis is that most of the pins are accessible and observable/controllable
    3. Also, the thermal and RF effects are minimized as opposed to closed-chassis debug
2. Closed Chassis Debug:
    1. In this scenario of closed-chassis debug, the available "Functional" connector on the platform is reused for debug
    2. Closed-Chassis debug provides the more realistic temperature and RF scenarios for debug
    3. Will cover this in more detail

Open Chassis vs. Closed-Chassis



Open Chassis (e.g., RVP)



Closed Chassis (e.g., FFD)

Wing/ Debug Card

* Presented by Nokia in 2009 at ESC-panel

Acknowledgements: Rolf Kuehnis

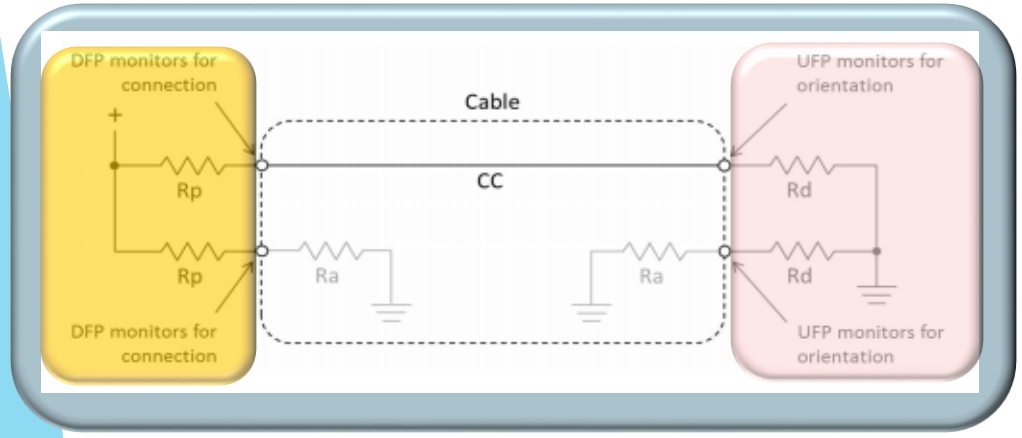
IEEE CEDA
Council on Electronic Design Automation

Open Chassis vs. Closed-Chassis (Cont'd)

1. Closed-Chassis Debug enables debug without opening the chassis
    1. Note that Security authorization is required to get any information from system
    2. Enables Hardware, Software and Firmware debug at the platform level
2. Enables early validation/debug on form-factor
    1. Avoids side-effects caused by RF, Power, Thermal as it mimics usage model
    2. Customers use mass-testing with 10s/100s unit simultaneously for stress test
    3. Helps to reduce DPM
3. Allows Infield debug to collect SoC/Platform parameters
    1. Reuses existing form-factor device interface for sending the information
    2. E.g., USB Type-A, Type-C, Display Port etc.
4. Connector of Choice for Closed-Chassis and Infield Debug
    1. USB Type-C Connector

USB Type-C (DFP vs. UFP) Vs. Type-A

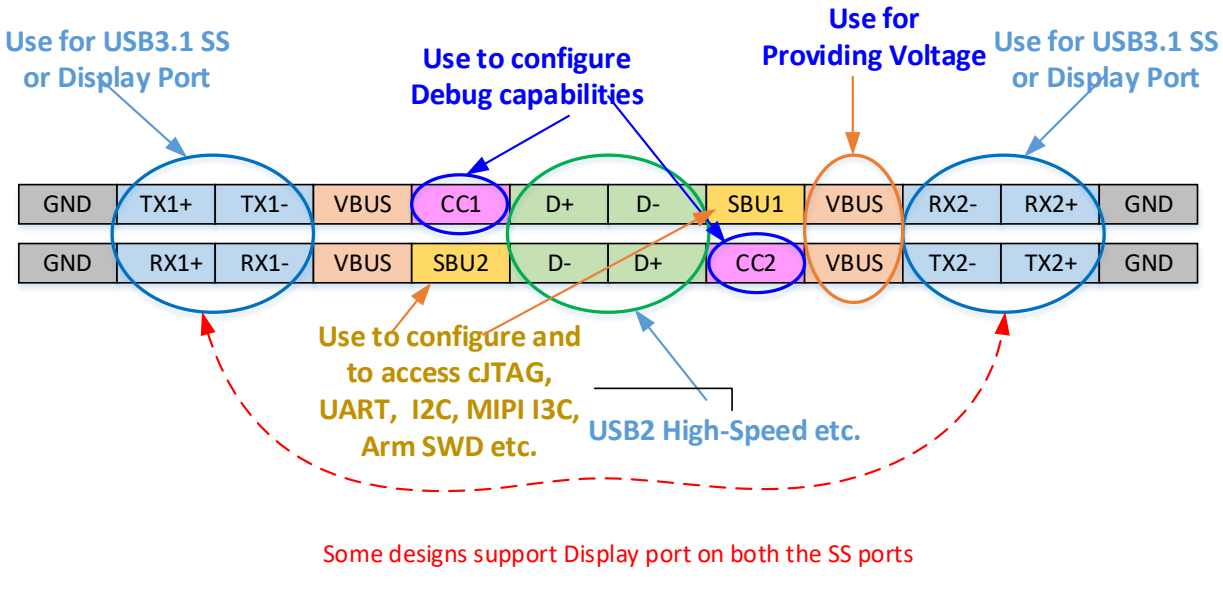
1. Type-A Connector
    1. Works as OTG (On The Go, can be Host or a Device, e.g., Laptop) – ID pin to identify role
    2. Works in Host Role (e.g., Desktop, Laptop etc.)
    3. Works in Device Role (e.g., Printer, Flash-drive etc.)
2. Type-C: Can be DFP (Host), UFP (Device), DRP (Both Host or Device)
    1. CC pins are used to identify the Host vs. Device roles
3. DFP (Downstream Facing Port or Host or Source) using Rp/Open or Open/Rp
4. UFP (Upstream Facing Port or Device or Sink) using Rd/Open or Open/Rd



#	Feature	CC1/CC2
1	DFP	Rp/Open or Open/Rp
2	UFP	Rd/Open or Open/Rd
3	DRP (Dual Role Port)	Toggles between DFP and UFP

USB Type-C for Debug

1. USB Type-C is ideal for debug of closed-chassis, form-factor devices:
    1. Areas ranging from TAP-level debug to high-level debug of SW apps
2. There are two ways of entering Type-C Debug Mode:
    1. Debug Alternate Mode: Via PD Messaging over a single CC line, using a std. Type-C cable
    2. Debug Accessory Mode: Via sensing of BOTH CC lines, using a direct connect or captive cable



Type-C Connector for Debug Modes

Platform Capability for Closed-Chassis and Infield Debug

- USB-C UFP/DFP/DRP CC Select:
  - DFP (Downstream Facing Port)
  - **Debug** Accessory Mode (Uses Rd/Rd or Rp_1/Rp_2* on CC1/CC2)
    - *Uses Asymmetrical Resistors for Orientation Detection
    - Uses captive cable or direct-connect

USB-C UFP/DFP/DRP CC Select Table

No.	Feature	CC1/CC2
1	DFP (Downstream Facing Port)	Rp/Open or Open/Rp
2	UFP (Upstream Facing Port)	Rd/Open or Open/Rd
3	DRP (Dual Role Port)	Toggles between DFP and UFP

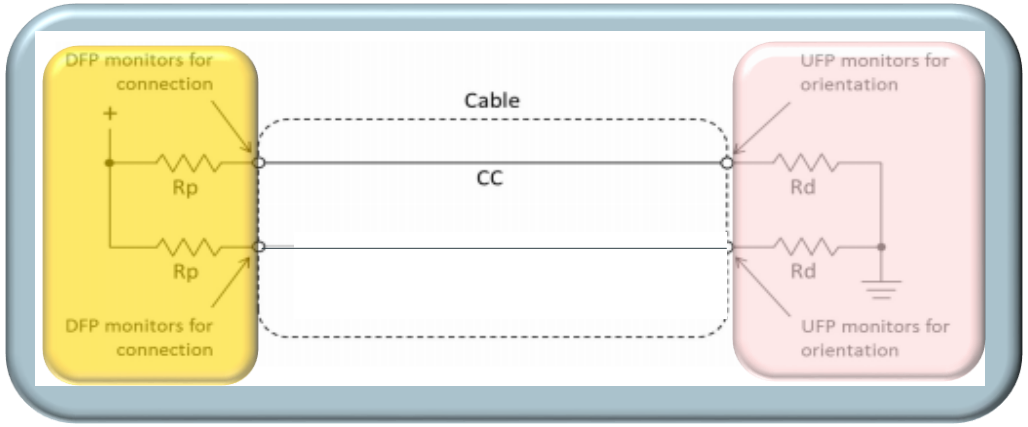
- USB DbC (USB Debug Class) Functions:
  - **Audio** Accessory Mode (Uses Ra/Ra on CC1/CC2)
  - **Debug** Accessory Mode (Uses Rd/Rd or Rp_1/Rp_2* on CC1/CC2)
    - *Uses Asymmetrical Resistors for Orientation Detection
    - Uses captive cable or direct-connect

DbC Functions

No.	Name	Shorthand	Usage	Sub-usage
1	General Purpose 1 (GP1)	DbC.GP1	Software	Kernel SW debug
2	General Purpose 2 (GP2)	DbC.GP2	Platform	DMA to/from system/memory
3	Dbg	DbC.Dbg	Platform	In/Out from FIFOs
4	Trace	DbC.Trace	Platform	Streaming from Aggregator

USB Type-C (Accessory Mode, Alternate Mode)

1. USB Type-C Accessory Modes:
    1. Audio Accessory Mode (Uses Ra/Ra on CC1/CC2)
    2. Debug Accessory Mode (Uses Rd/Rd or Rp1/Rp2* on CC1/CC2)
        1. *Uses Asymmetrical Resistors for Orientation Detection
        2. Uses captive cable or direct-connect

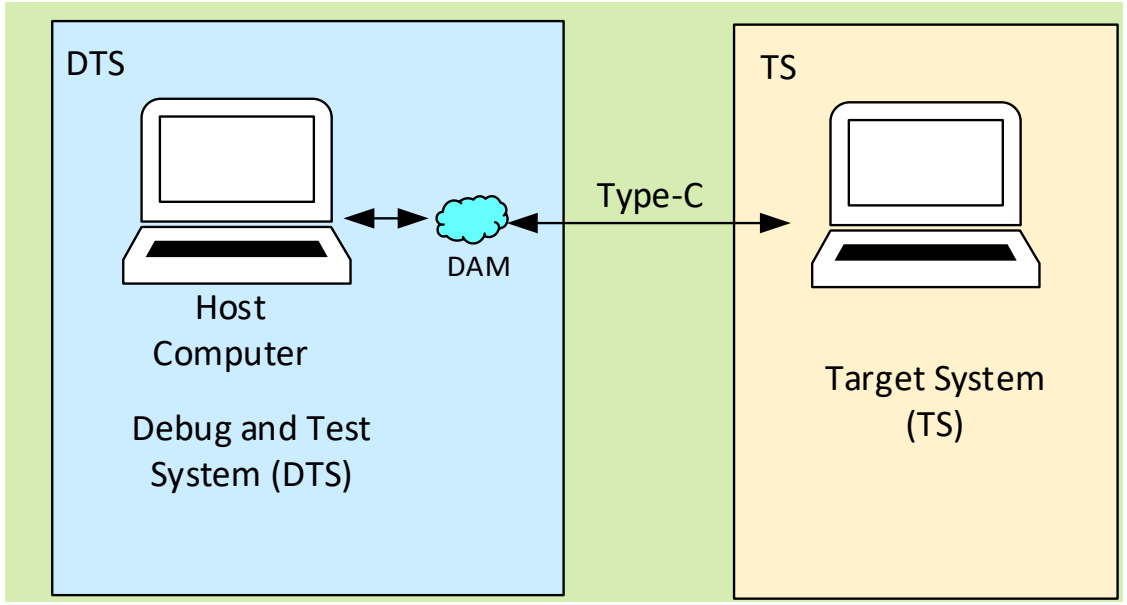


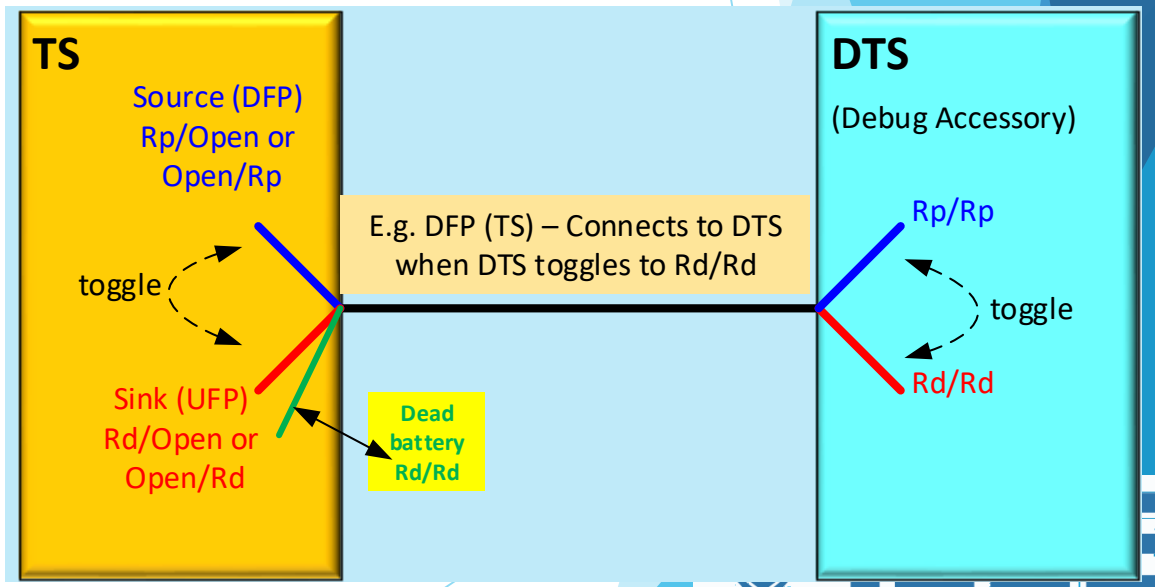
#	Feature	CC1/CC2
1	Audio Accessory Mode	Ra/Ra
2	Debug Accessory Mode	Rd/Rd or Rp1/Rp2*

1. USB Type-C Alternate Modes:
    1. Display Alternate Mode
    2. Debug Alternate Mode
        1. Alternate Mode uses PD Controller and PD (VDM) Messaging to enter/change modes

Debug Accessory Mode

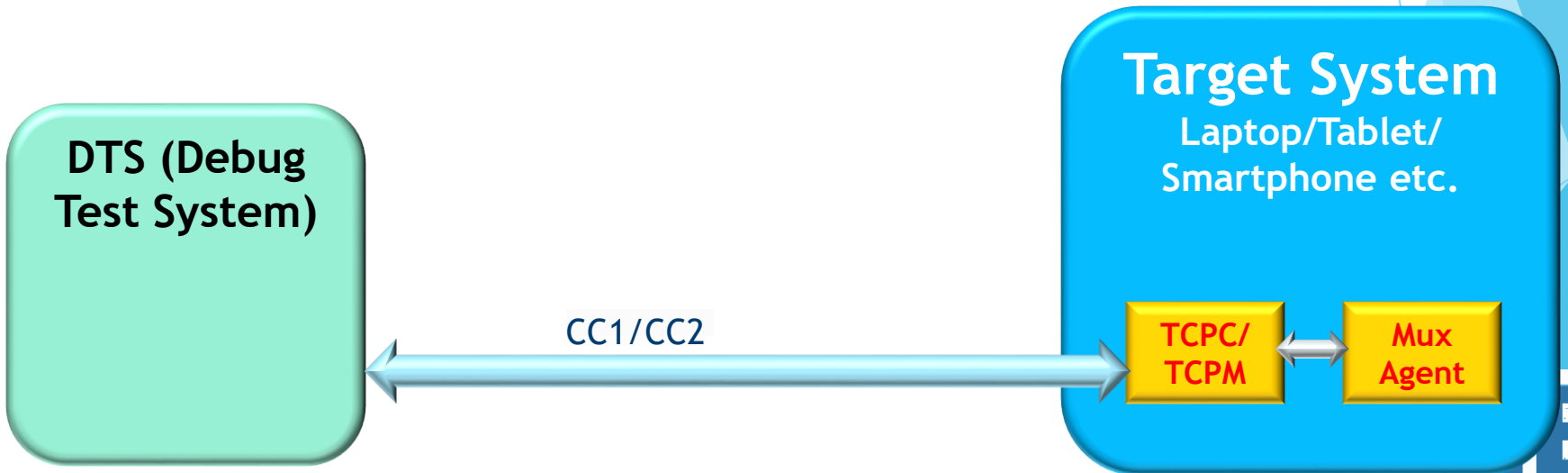
1. Debug Accessory Modes:
    1. Debug Accessory Mode is defined when a Target System under Test presents:
        1. Rp1/Rp2 (Source or DFP – Downstream Facing Port) or
        2. Rd/Rd (Sink or UFP – Upstream Facing Port)
        3. DRP (Toggles until the logic settles to one of the states)





Debug Alternate Modes

1. Debug Alternate Modes:
    1. Debug Alternate Modes are specific redefinition of Type-C pins for debug:
        1. Allows simultaneous debug and functional modes (e.g., USB, Display, TBT etc.)
        2. Uses Structured VDM as defined by the USB Power Delivery Spec
    2. MIPI NIDnT Alternate Modes are defined by MIPI Alliance
        1. Uses NIDnT overlays

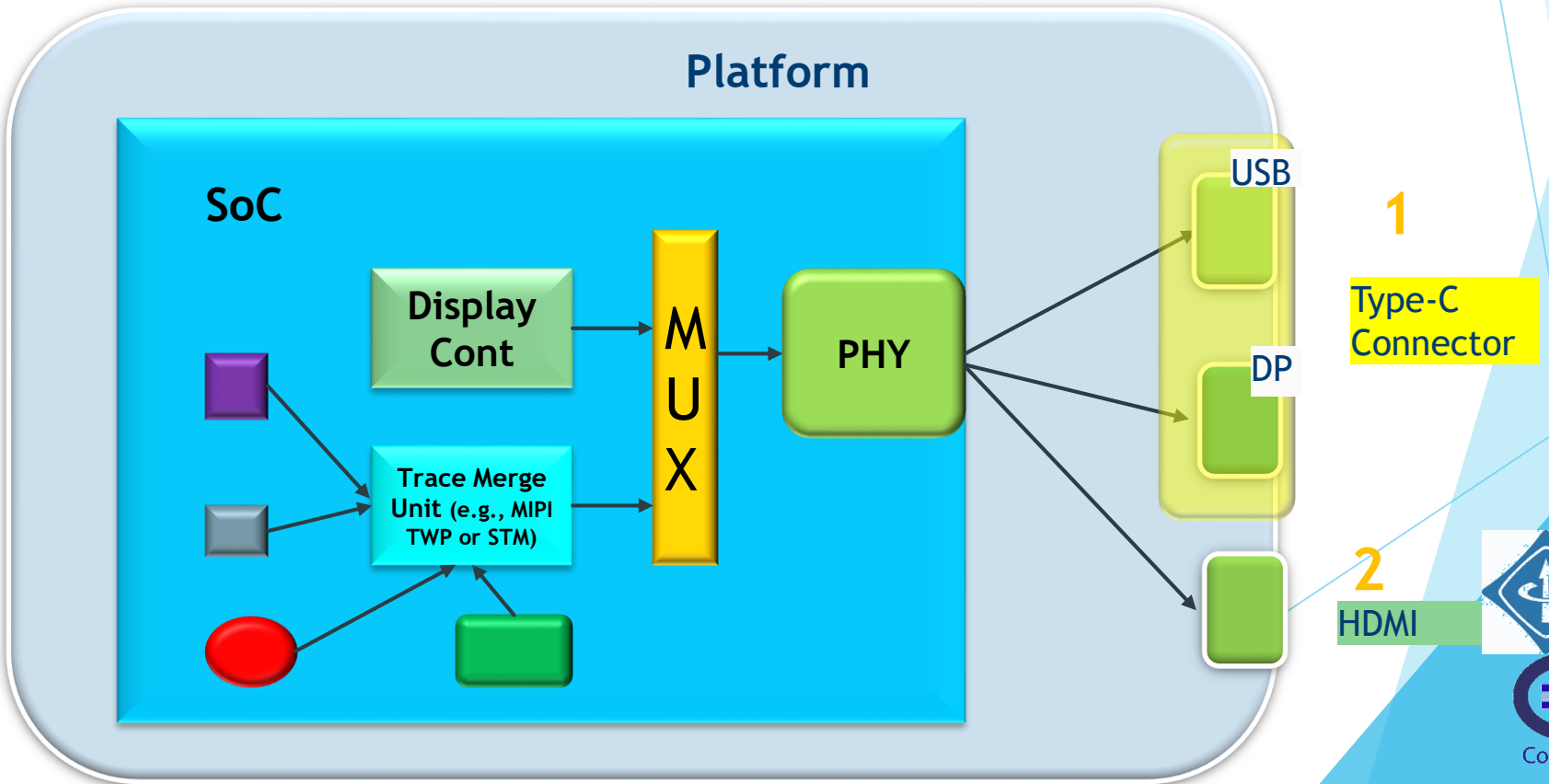


USB Debug Class (USB2 DbC and USB3 DbC)

1. USB Debug Class Debug capabilities (Requires DbC Controller powered up)
    1. Does not required OS/SW support
    2. DbC.DfX (Provides JTAG access)
    3. DbC.GP1 (Provides Kernel Mode Debug Capability)
    4. DbC.GP2 (Provides DMA Capability)
    5. DbC.Trace (Provides Trace Capability)
2. Intel DCI-OOB (Direct Connect I/F - Out of Band) provides a bypass path from the USB2/3 PHYs
    1. Via Intel EXI (DFX eXtendable Interface) to provide JTAG Access
    2. Does not require USB controllers to be powered up
    3. Useful for JTAG
3. DbCs are available:
    1. Over USB2 & USB3.x Interfaces
    2. Speeds (at 480Mb/s for USB2.0, 5Gb/s for USB3.0
    3. Up to 80Gb/s or 120Gb/s (asymmetric mode) for USB4 V2

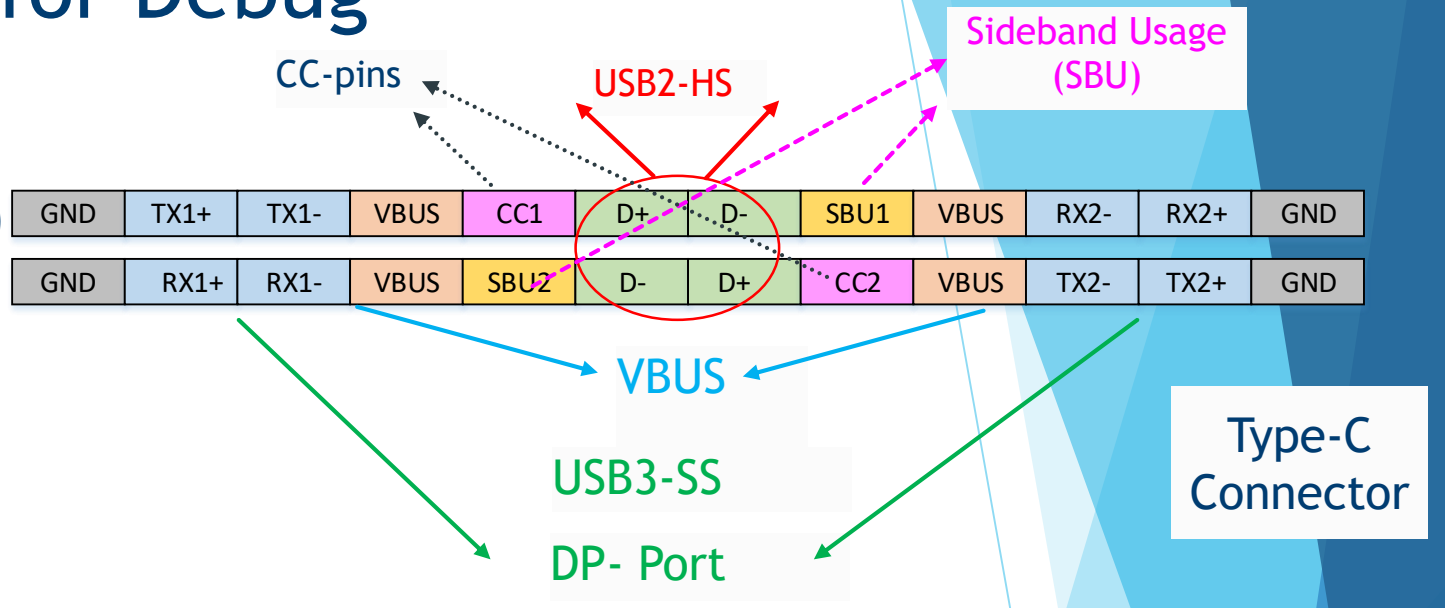
High Speed Trace Interface (MIPI-HTI) implementation

1. HTI used for Hi-Speed Tracing
    1. Provides Very High BW, uses Aurora Protocol
    2. Display and Trace Muxed via Display Port (DP-PHY), different PHYs used for Display and Trace
    3. Available over Type-C receptacles
    4. Useful for Closed-Chassis Debug



Summary of USB Type-C for Debug

1. SBU (Side Band Usage) Pins:
   1. Used for Bare-Metal access
   2. For MIPI-I3C, cJTAG*, Arm SWD, UART, I2C etc.)
2. USB2.DbC (USB2 Debug Class)
   1. Used for other Debug
   2. USB2 for Functional data transferDbC.GP1
3. USB3.DbC (USB3 Debug Class)
   1. USB3.SS (Super Speed)
   2. TBT or MIPI-HTI
4. Electricals and PHY
   1. SBU are CMOS/GPIO type
   2. USB3 SS are SERDES
   3. USB2 are something in between



Type-C Cable

Debugger
1. DCI-OOB
2. DbC
3. HTI

Example of Closed-Chassis Debug of a Smartphone/Tablet

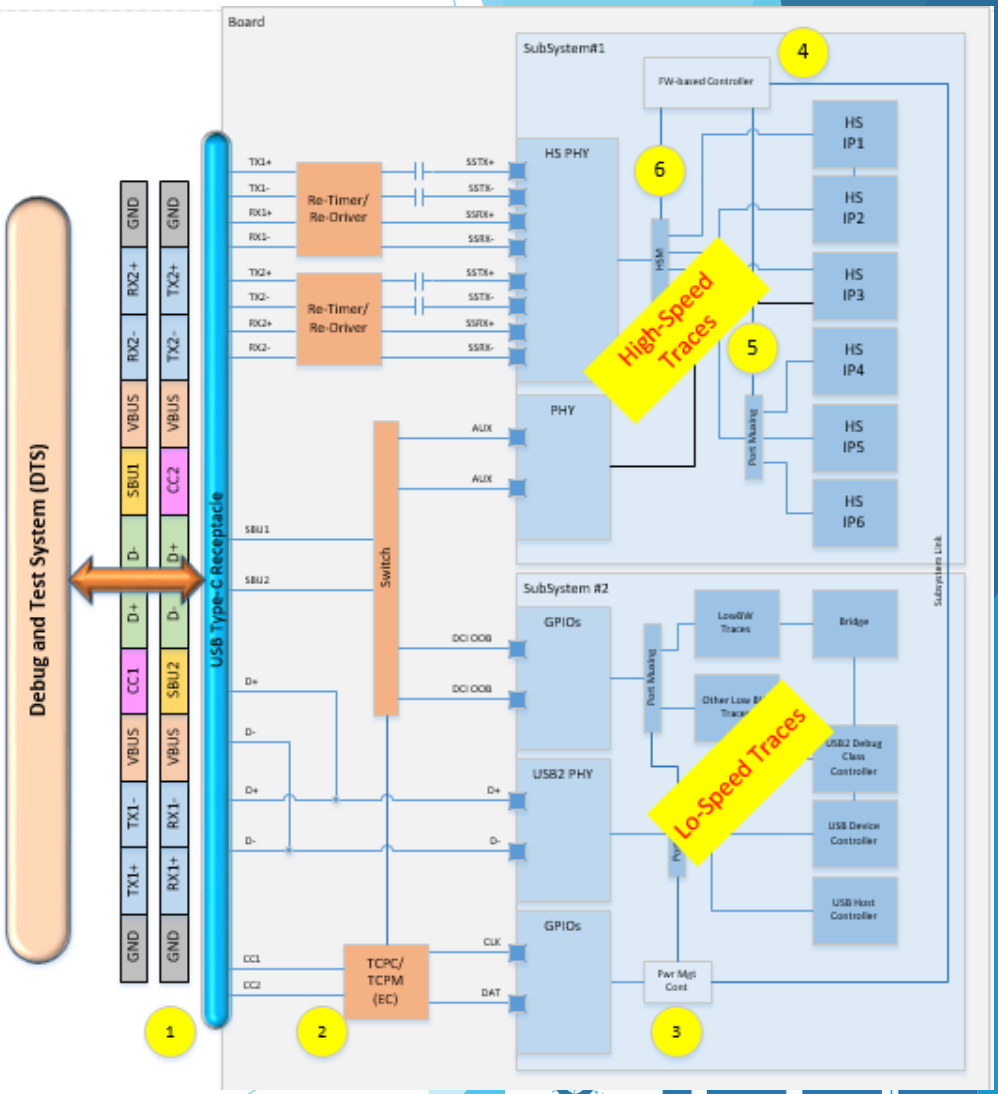
One Ubiquitous Connector for Closed-Chassis Debug

* IEEE 1149.7

Council on Electronic Design Automation

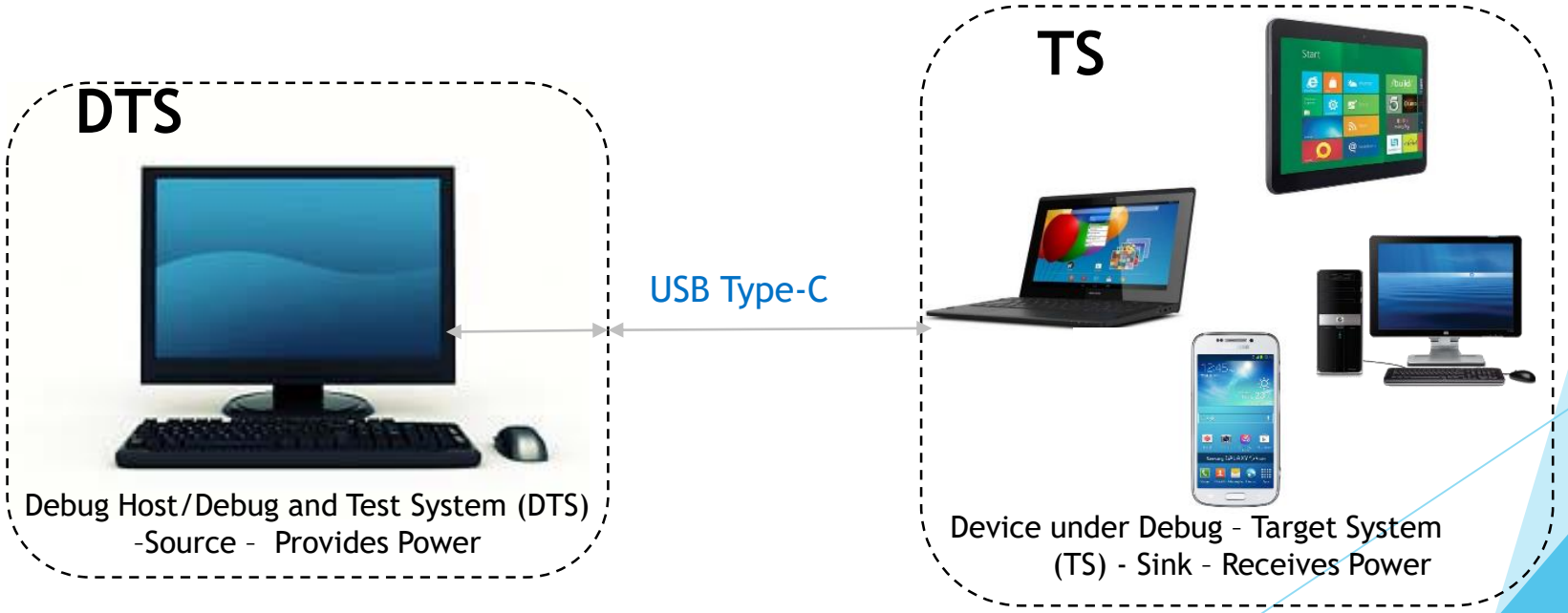
Platform Capability for Closed-Chassis and Infield Debug

- FW interaction and Type-C enabling for Debug:
  - **With Type-C connect to debugger:**
    - Uses captive cable or direct-connect
    - Connection made based on Rd/Rd or Rp_1/Rp_2 over CC
    - Informs PMC to initiate port-mux selection for USB2.DbC
    - Sends message to FW-based controller in Subsystem#1
    - FW-based controller selects the port-mux shown as 5
    - HSM (High-Speed) Mux shown as 6 used to select HTI/USB3.DbC over Type-C
  - **Closed-Chassis and In-field Debug:**
    - Once connection is done
    - SBU pins used for Bare-Metal Debug
    - USB2 and USB3 DbC are made available over the respective pins
    - Display Port can be used for High BW Trace



Customer Platform Debug Usage

1. Easy Diagnosis of Customer Platform Issues/Bugs
2. Ability to Observe, Analyze and Optimize Products
    1. Customer Expectations: Effective, Robust, Consistent, Easy-to-use Solution requiring no Platform support, without any Expensive Hardware, etc.
    2. Use of USB Type-C for most customer Closed-Chassis Debug/Analysis



Security Implications

1. Closed-Chassis Debug needs strong Security
    1. Note that Security authorization is required to get any information from system
    2. Enables Hardware, Software and Firmware debug at the platform level
2. Since DFX port is one of the vulnerable interfaces for hacking
    1. Blackhat exposed some vulnerability of the DFX port

3. Server-based Security Authentication
    1. Physical access to the device does not result in vulnerability
    2. Any remote access vulnerability is mitigated by server-based security authentication

IEEE CEDA
Council on Electronic Design Automation

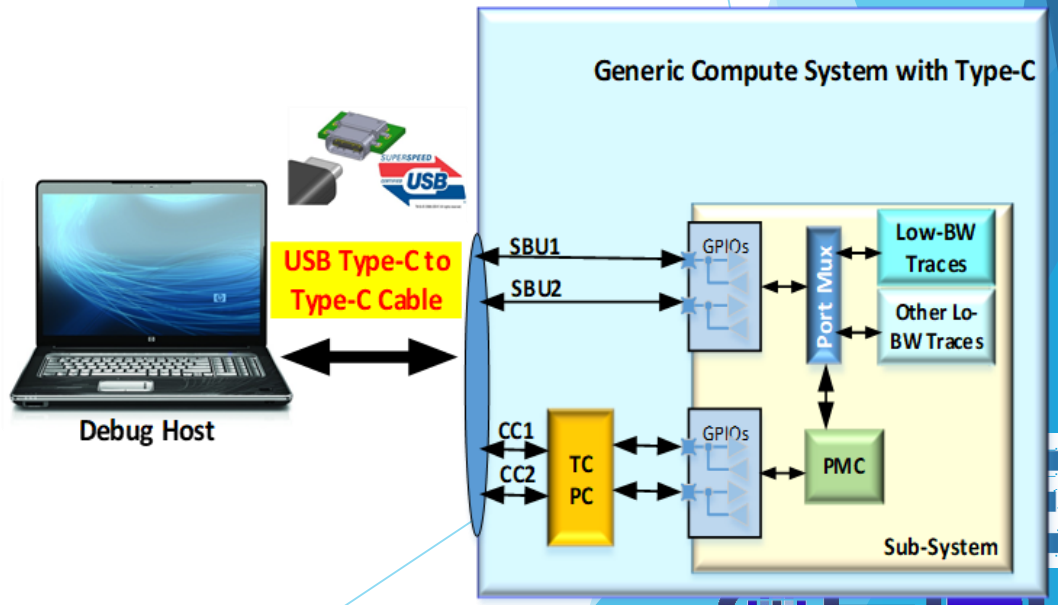
Overall Debug of SoC

- Closed-Chassis debug platform is useful for Infield debug and for Life-Cycle Management
  - MIPI I3C, USB Type-C enable capabilities for closed-chassis, in-field debug

- For GPU and EC with Firmware debug
  - Capability achieved with multi-drop capability inherent in MIPI Debug for I3C

E.g., USB-C, USB-A, Display Port etc.

Additionally, USB Debug Class available over USB2 and USB3 Interface

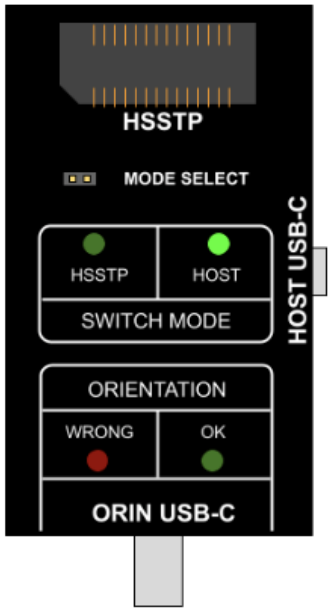


Few current day implementation examples…. (1 of 4)

- USB-C to USB-C (HSSTP) switch to automate Nvidia Orin USB port switching, used for:
  - Debug and Trace via HSSTP
  - Device USB connection
  - 40-pin Samtec/USB-C to USB-C (HSSTP) switch simplifies switching between two modes of USB port
- Lauterbach has developed several adaptors/dongles for USB-C debug
  - Trace32® converter: MIPI 10 to USB-C, Samtec40 HSSTP to USB-C

Target Adaptation for Converter Samtec40 HSSTP to USB-C (DAM, D1)

- **Product**



The **Host USB-C** port is connected to the Orin USB-C port.

Few current day implementation examples…. (2 of 4)

- Lauterbach has developed several adaptors/dongles for USB-C debug
  - HSDP Adapter for PowerTrace Serial
  - USB-C to USB-C breakout module
  - Package CombiProbe Intel® DCI-OOB
  - Debugging via USB

HSDP Adapter for PowerTrace Serial

Cable USB-C to USB-C Breakout Module

Package CombiProbe Intel® DCI OOB

Debugging over USB

OVERVIEW

Full Debug and Trace Experience via USB Connection

In some applications you have to debug and trace targets without physical access to the Debug/Trace interfaces, e.g. if the target is located in a closed chassis. TRACE32 ® **supports several silicon IP solutions that enable full debug and trace functionality** without limitations by using a USB connection between the target and your (host) computer. These solutions provide low-level access to the target without requiring physical access to Debug/Trace interfaces.

IEEE CEDA
Council on Electronic Design Automation

Few current day implementation examples…. (3 of 4)

- Apple:
  - Home – Bonobo JTAG/SWD Debug Cable (bonoboswd.com)
  - Apple introducing USB-C Diagnostic tool
- Google Chrome:
  - USB-PD sniffing dongle with Type-C connections
    - Supported by Chrome devices
  - Case Closed Debugging Debugger

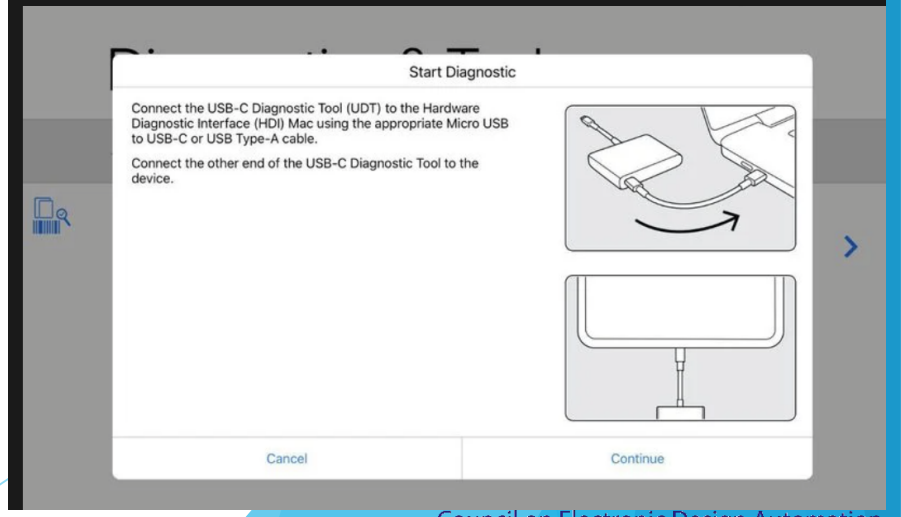
This page describes a USB-PD sniffing dongle with Type-C connectors.

C2D2: Case-Closed Debugging Debugger

Apple Introducing New Internal USB-C Diagnostic Tool

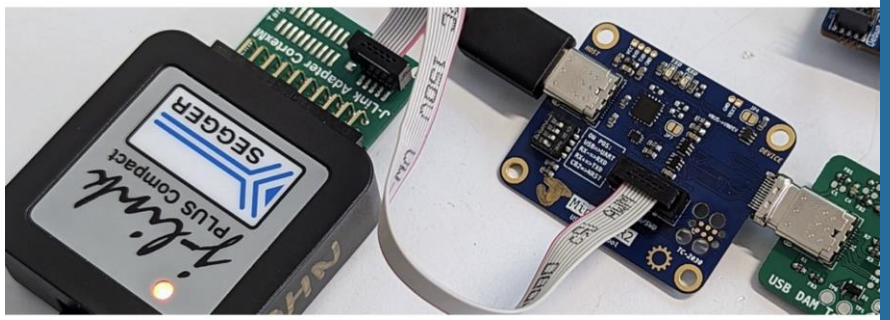
Sunday May 31, 2020 7:26 pm PDT by Frank McShan

Apple is introducing a new internal USB-C Diagnostic Tool as a successor to its existing Serial Number Reader, which can be used to both collect a device's serial number directly from its logic board and test power on a device itself.

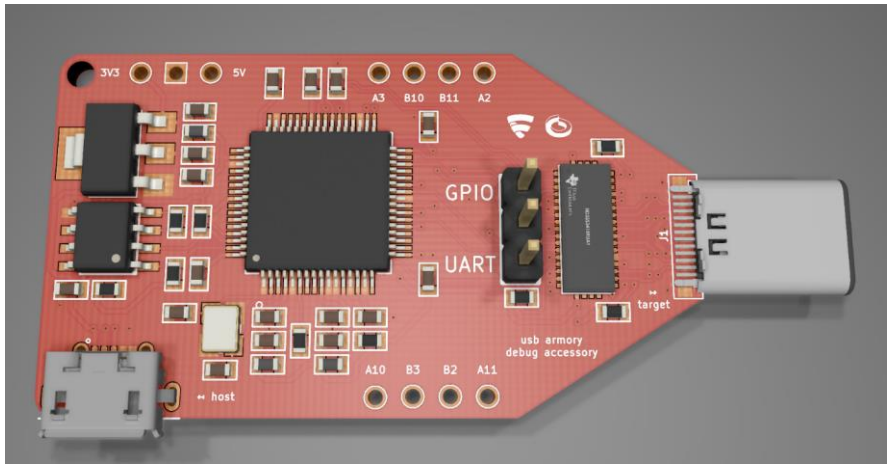


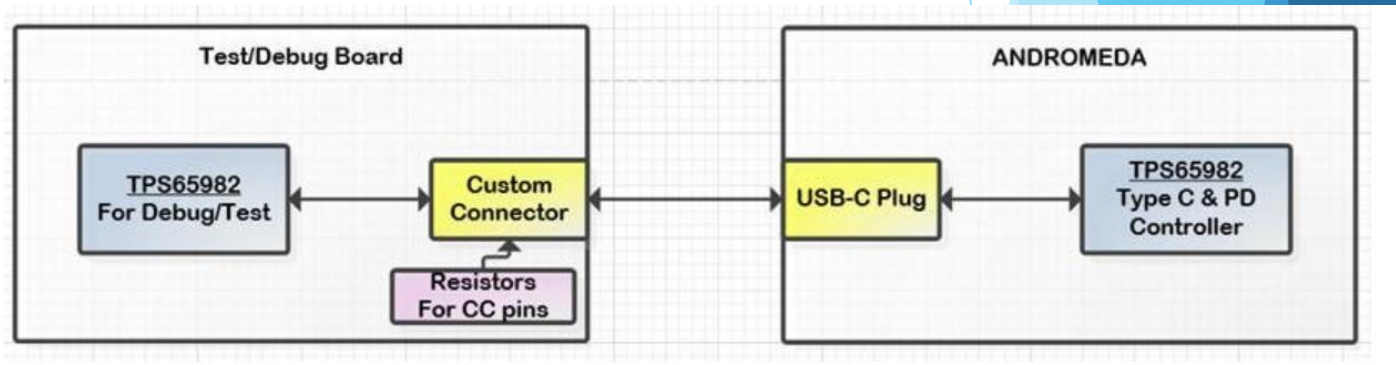
Few current day implementation examples…. (4 of 4)

- Other Links:
  - Minnow USB-C DAM Tool
  - USB armory MKII debug accessory
  - TI TPS65982: Debug Accessory mode documentation



Minnow USB-C DAM Tool





IEEE CEDA
Council on Electronic Design Automation

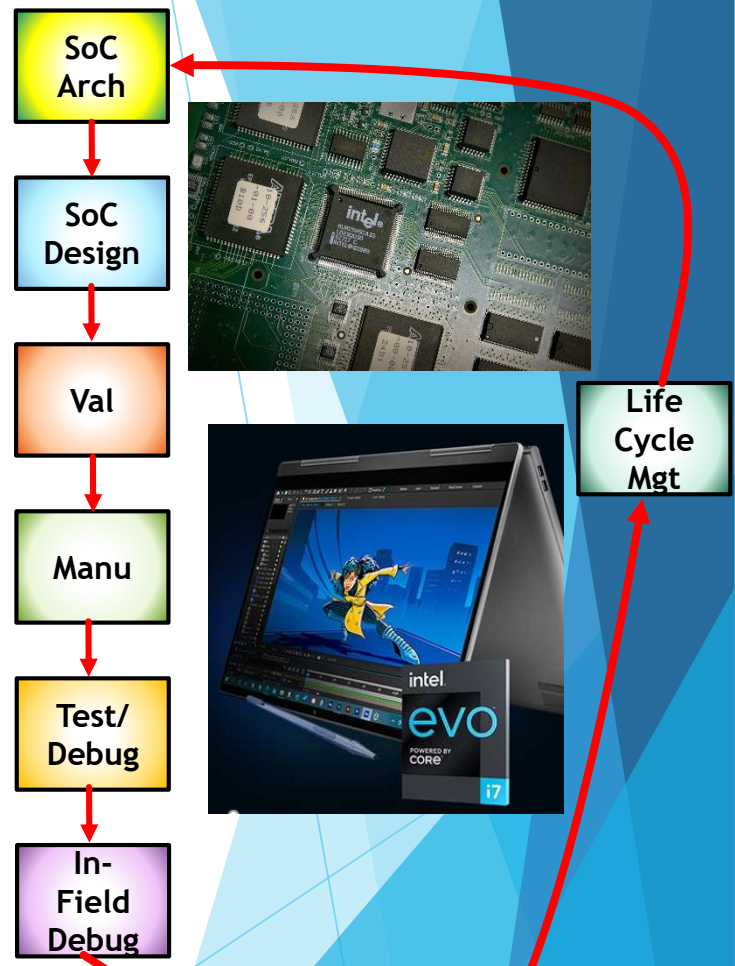
In-field Debug

1. "In-Field Debug" refers to the process of identifying and fixing bugs in a live system
    1. System is actively being used in its intended environment (Mission Mode), as opposed to debugging in a controlled testing environment, e.g., ADAS, Automated driving etc.
    2. This involves analyzing HW behavior by periodic real-time testing, reporting errors
    3. If possible to make adjustments on the fly while minimizing disruption
2. Infield debugging requires diagnosing HW/SW issues while maintaining system availability
    1. Some of the HW bugs are manifestations of validation/manufacturing bug escapes
    2. SW bugs are in products due incorrect system behavior & low system availability
3. Conventional debug methods interfere with the performance and availability of the system
    1. Closed-Chassis debug helps capture in-field debug info without opening chassis
    2. Can extract functional and in-field debug information by interleaving the two
        1. Note: Security authorization is required to get any information from system

SoC/Platform Life Cycle Management

1. Silicon Life Cycle Management:
    1. Ability to improve silicon health by way of monitoring, analyzing and optimizing devices as they are designed, manufactured as well as tested and deployed in end-user devices/systems
    2. Owing to the increase in complexities of SoCs and Systems as well as the need for increased reliability and performance and the main reason for requiring on-going maintenance and optimization of devices throughout the life cycle

2. SoC/Platform Life Cycle Management
    1. Closed-chassis allows collection of SoC/Platform Life Cycle Management info
    2. Provides approaches to monitor and extract SoC/Platform parameters
    3. To optimize even after the SoC/Platforms are deployed
    4. Enables capturing information right after manufacturing all the way till the EOL (End of Life) of the product from “Cradle to Grave”



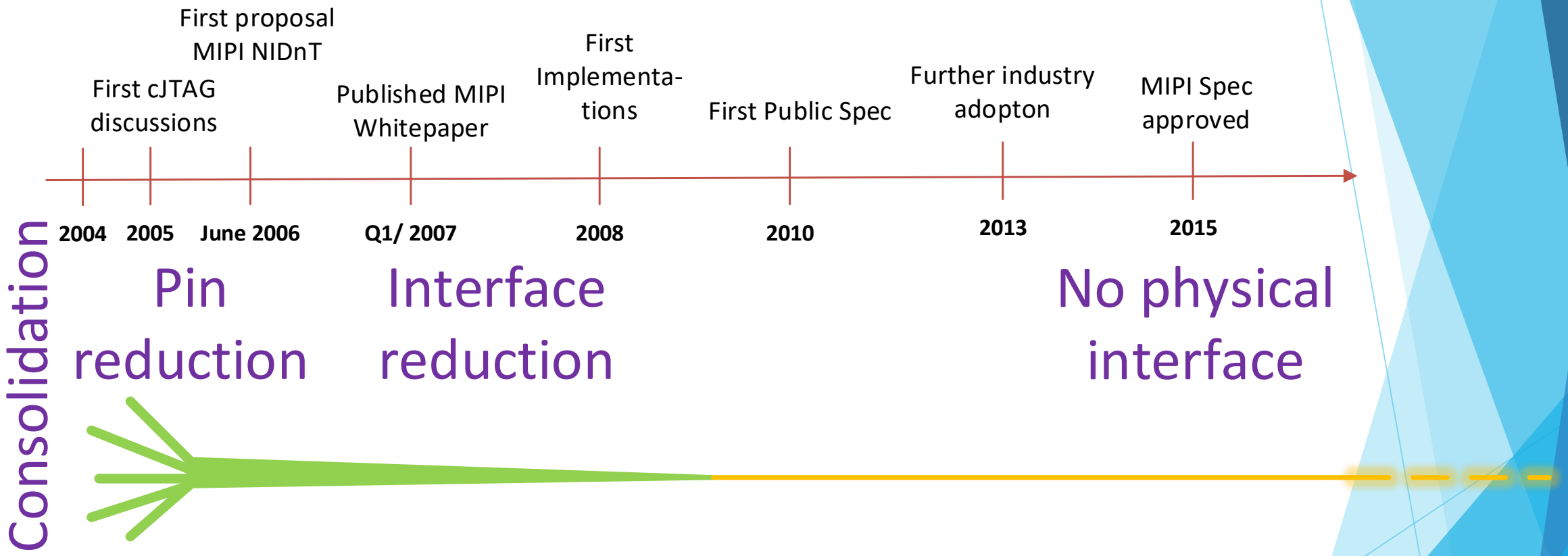
IEEE CEDA
Council on Electronic Design Automation

IEEE P2929 for “In-Field Test/Debug” and “Silicon Life Cycle Management”

1. IEEE P2929 Standards effort titled “Standard for System-level State Extraction for Functional Validation and Debug”
    1. P2929 enables a standardized way to debug systems/platforms by outlining methods to obtain the functional states by extracting the scan states and array states
2. In-field Test and Debug requires that the internal test features, such as LogicBIST, MemoryBIST etc., be enabled
    1. IEEE P2929 enables activation of tests such as LogicBIST and MemoryBIST for enabling in-field test/debug
3. In-field test and debug also requires that the internal states be made observable for debug purposes
    1. IEEE P2929 enables extraction of the functional state via capturing the scan states as well as the array states
    2. Another capability that IEEEP2929 is working hard to achieve is to back-annotate the netlist information obtained for scan to RTL level for ease of debug
4. P2929 test/debug standards can be implemented
    1. In both Open-Chassis and Closed-Chassis systems

IEEE CEDA
Council on Electronic Design Automation

Standardization took many years of effort….



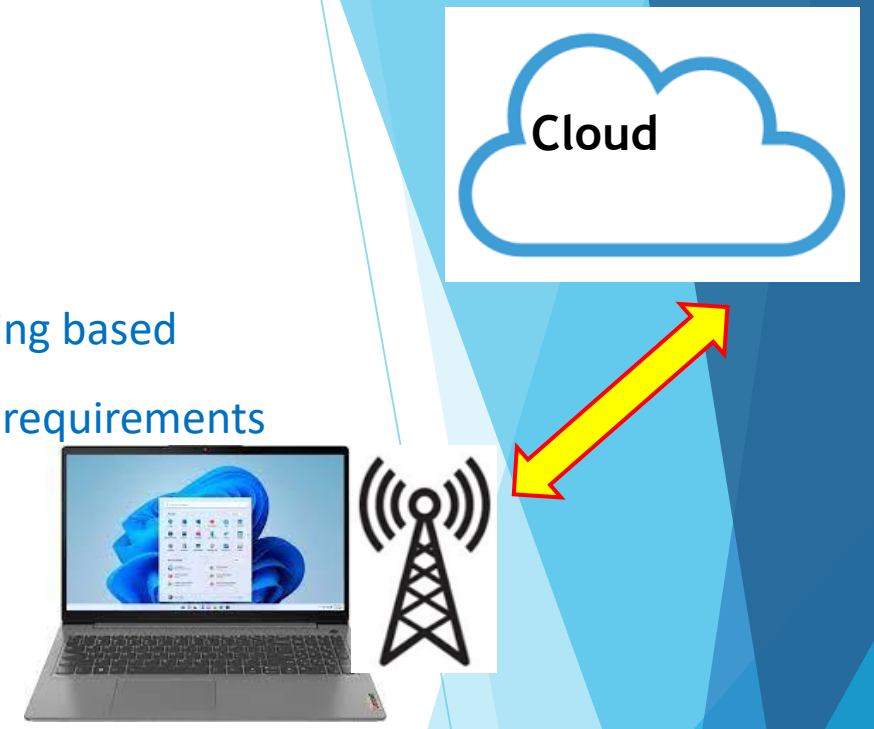
Drivers are cost reduction, miniaturization/ Ubiquitous compute and new fields of use.

* MIPI NIDnT(SM) Example

IEEE CEDA
Council on Electronic Design Automation

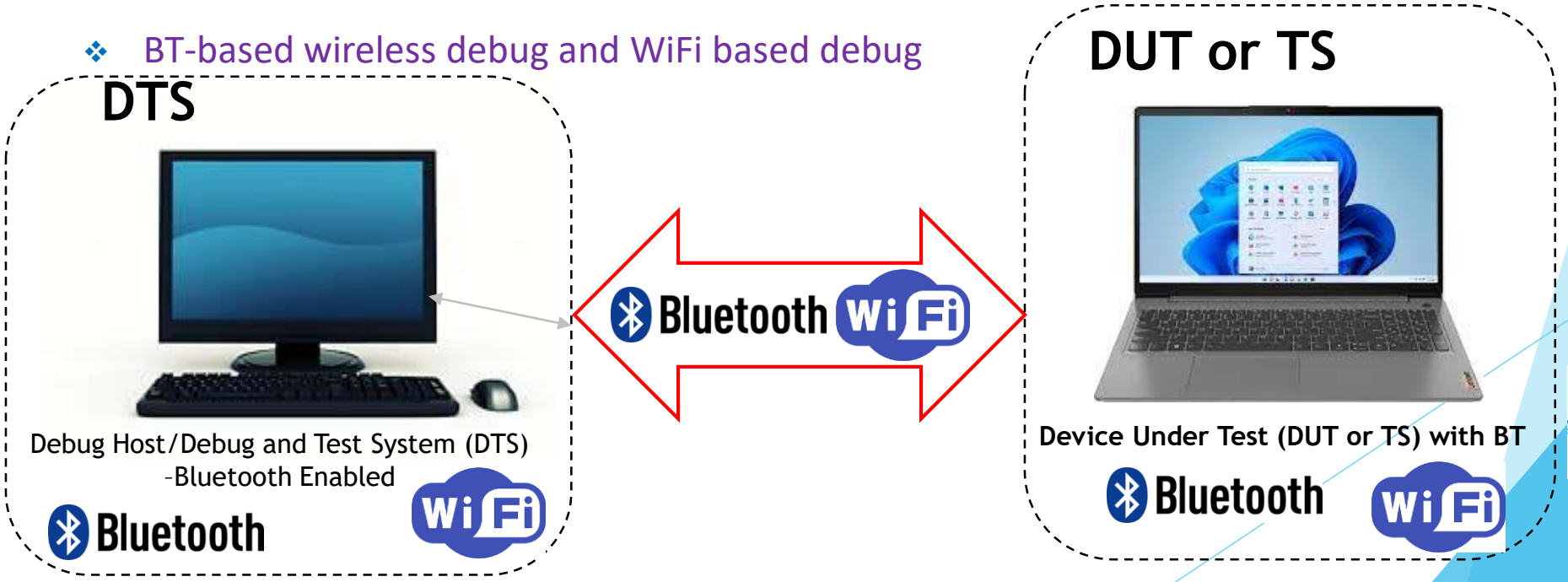
Conclusions and Future Work (1 of 2)

- One Ubiquitous connector for Closed-Chassis Debug
  - Spans functional and debug needs
- Debug Accessory/Alternate Modes to cover major debug modes
  - Debug Accessory Mode: HW based and Debug Alternate Mode: PD Messaging based
  - USB2 DbC and USB3 DbC for covering low-power debug and High-BW trace requirements
  - SBU (Side Band Usage) pins for Bare-Metal Debug
- Closed-Chassis debug is not only useful for platform debug
  - Platform debug implies HW, SW and FW debug
  - This is specifically useful for SW/FW debug, especially with current days OTA updates
  - OTA updates commonly done for smartphones, IOTs, ADAS (Automated driving), industrial applications etc.



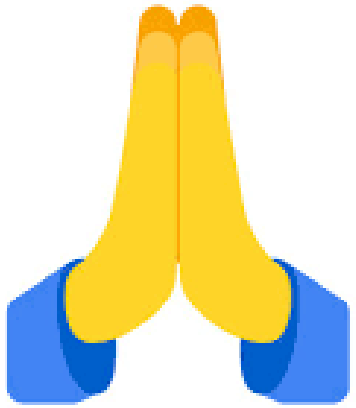
Conclusions and Future Work (2 of 2)

- In addition, useful for:
  - In-field debug and SoC/Platform Life Cycle Management
- Capability that this feature provides for Life-Cycle Management is:
  - Monitor and extract both SoC and Platform parameters to optimize after the SoC/Platforms are deployed
  - Capture information right after manufacturing all the way to EOL ➔ From “Cradle to Grave” for SoC/Platform
- To enable Life-Cycle Management traces to be sent to the cloud
  - Another future work is to use using wireless interface(s) for debug
    - BT-based wireless debug and WiFi based debug



IEEE CEDA
Council on Electronic Design Automation

Thank You for your undivided attention!



IEEE CEDA
Council on Electronic Design Automation

Q&A