

**SAMD21 ARM Cortex-M0+
Microcontroller & Harmony v3
SAM2002_{ADV} v1.01**



A Leading Provider of Smart, Connected and Secure Embedded Control Solutions



SMART | CONNECTED | SECURE

Libra Chien
CAE Taiwan
Microchip Technology Inc.
April 2024

SAM series course history and brief

- **SAM1001 – MCU32 Peripheral Library (PLIB)**
 - (2017~2018)
 - Peripheral Library (PLIB) on APP045v2.0 (SAM D21 Arm Cortex-M0+)
 - Advance Software Framework (ASF) v3.x + GNU C compiler + Microchip Studio 7
 - (2019~2022)
 - Peripheral Library (PLIB) on APP045v2.0/v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - Peripheral Library (PLIB) on APP045v4.x/v5.x (SAM D21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE
- **SAM2001ADV – MCU32 Advanced Peripheral Library (PLIB)**
 - (2021~2022)
 - Peripheral Library (PLIB) Advanced modules on APP045v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - Peripheral Library (PLIB) Advanced modules on APP045v4.x/v5.x (SAM D21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE
- **SAM2002 – MCU32 System Service and Driver**
 - (2019~2022)
 - System Service and Driver on APP045v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - System Service and Driver on APP045v4.x/v5.x (SAM D21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE
- **SAM2002ADV – MCU32 Touch and USB Device**
 - (2021~2022)
 - QTouch Library and USB Device Driver on APP045v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
 - (2023~2024)
 - QTouch Library and USB Device Driver on APP045v4.x (SAM D21 Arm Cortex-M0+)
 - MCC Harmony + XC32 Compiler + MPLAB X IDE

Background Knowledge and Preparation

- **SAM2002ADV is for advanced users, then we will not introduce :**
 - MPALB X IDE and Harmony Configuration. (Please study from SAM2001 course)
 - How to download(prepare) latest Harmony Framework. (Please study from SAM2001 course)
 - How to create a new project. (Please study from SAM2001 course)
 - PLIB (Peripheral Library) implementation. (Please study from SAM2001 and SAM2001ADV course)
 - System Service and Driver implementation. (Please study from SAM2002 course)
- **SAM2002ADV will use below tool versions for Labs.**
 - MPLAB® X IDE v6.15 <https://www.microchip.com/mplab/mplab-x-ide>
 - MPLAB® XC32 v4.35 <https://www.microchip.com/mplab/compilers>
 - MPLAB® Code Configurator v5.4.1 (Install in MPLAB X IDE Plug-in)
 - SAMD21 DFP v3.6.144 (Device Family Pack download from X IDE \Tools\Packs)
 - CMSIS v5.8.0 (Common Microcontroller Software Interface Standard)
 - MCC Core v5.6.1 (Install with MPLAB X IDE v6.15)
 - MPLAB® MCC Harmony Framework (From X IDE MCC Harmony Content Manager)
 - Required Content (Must install)
 - csp 3.18.4
 - harmony-services 1.3.2
 - CMSIS_5 5.9.0
 - quick_docs 1.6.0
 - Optional Content (Manual select in Content Manager)
 - touch v3.15.0
 - usb v3.12.0
 - dev_packs v3.18.1
 - core v3.13.3

Advance Course

- [Introduction to QTouch® Peripheral Touch Controller \(PTC\)](#)
- [Harmony Touch Configurator](#)
- [APP045 EVB QTouch® Design and Pattern](#)
 -  [Lab1 Touch Buttons](#)
 -  [Lab2 Touch Slider](#)
 -  [Lab3 Data Visualizer for Touch](#)
- [USB Fundamentals](#)
- [USB Device Library \(USB Device Architecture\)](#)
- [Harmony USB Device - Device Layer Configurator](#)
 -  [Lab4 USB Device - Device Layer Initialize](#)
- [USB Device CDC Class](#)
 -  [Lab5 USB Device CDC Class - Basic TX and RX](#)
- [USB Device HID Class](#)
 -  [Lab6 USB Device HID Class - Basic TX and RX](#)

Course Coverage

Course will not include :

- The principle of Capacitive Touch.
- Pattern design for Capacitive Touch.
- The principle of our PTC/CVD.
- PTC tuning parameters.

Course will include :

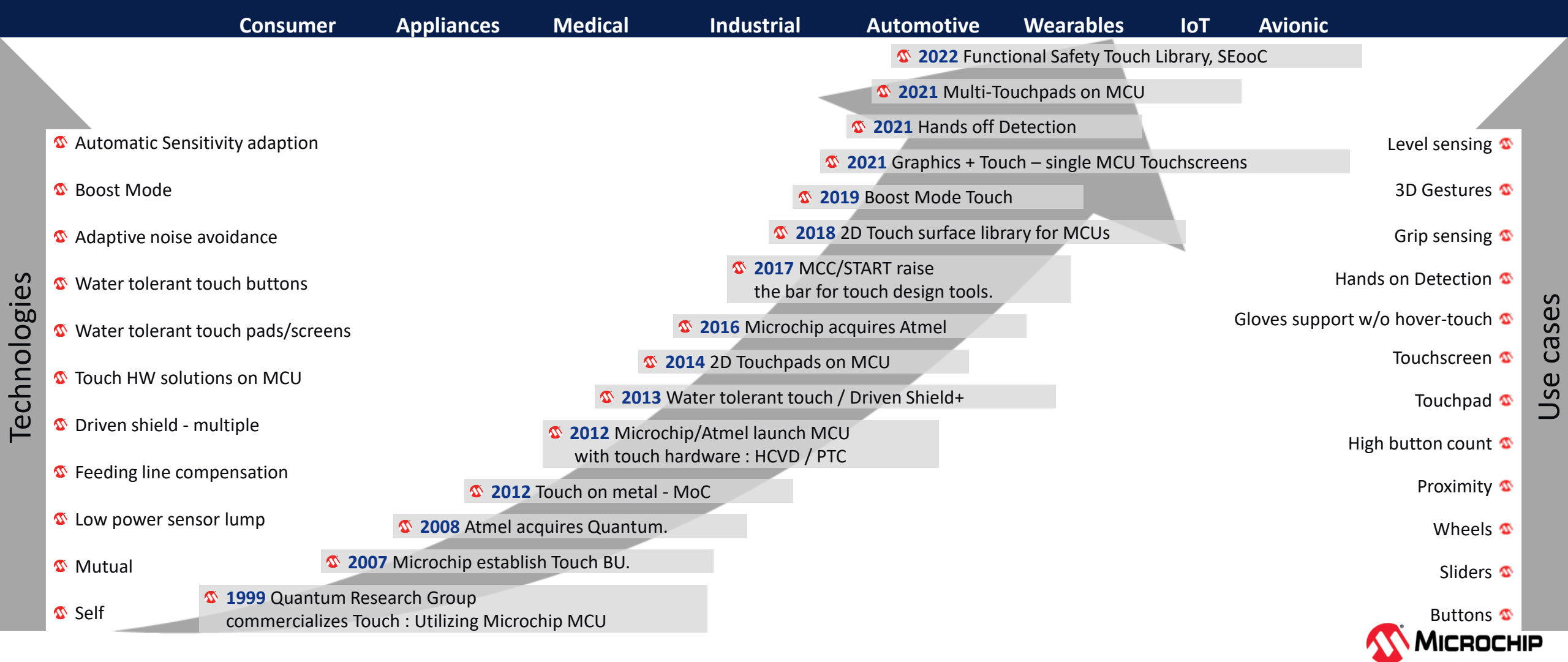
- Usage of MCC Harmony Touch library.
- Introduction to the MCC Harmony Touch Configurator.
- Touch visualization with MPLAB Data Visualizer.

Introduction to QTouch® Peripheral Touch Controller (PTC)

Introduction to QTouch® Peripheral Touch Controller (PTC)

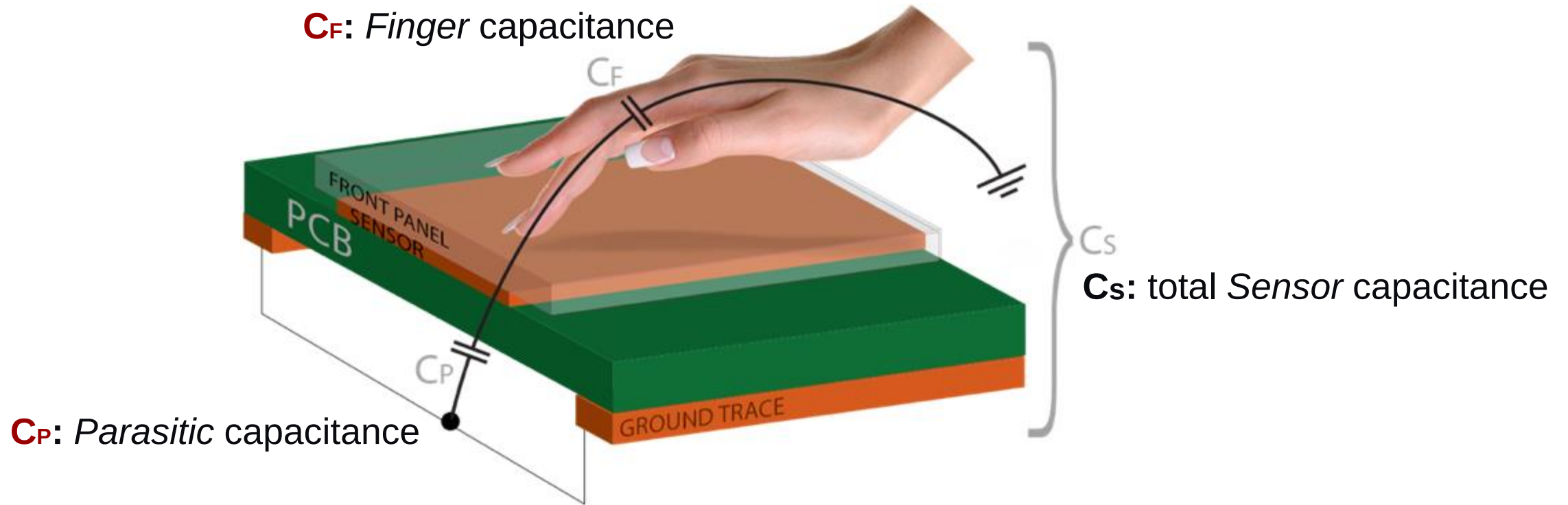
Long-Term innovative Player in Touch

Microchip serves all Markets and all use cases for touch.



Introduction to QTouch® Peripheral Touch Controller (PTC)

Capacitive Touch Sensor

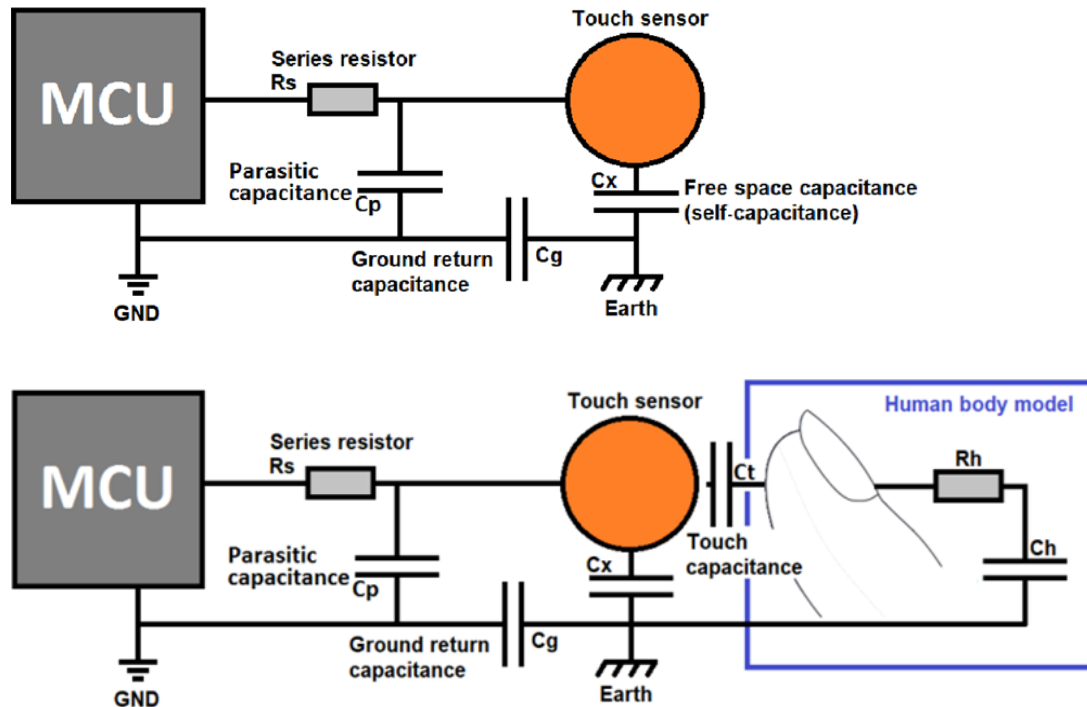


$$\text{Sensor Capacitance } (C_S) = C_P + C_F$$

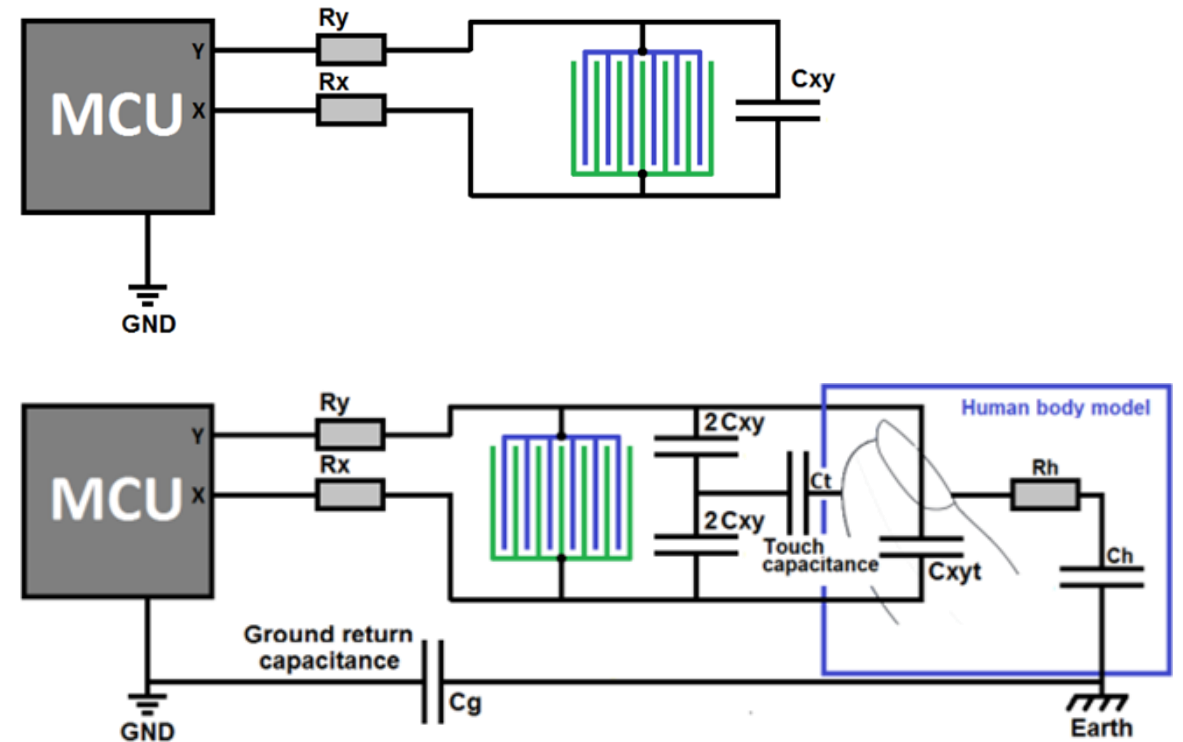
Introduction to QTouch® Peripheral Touch Controller (PTC)

Capacitance Touch Measurement

Self Capacitance Touch

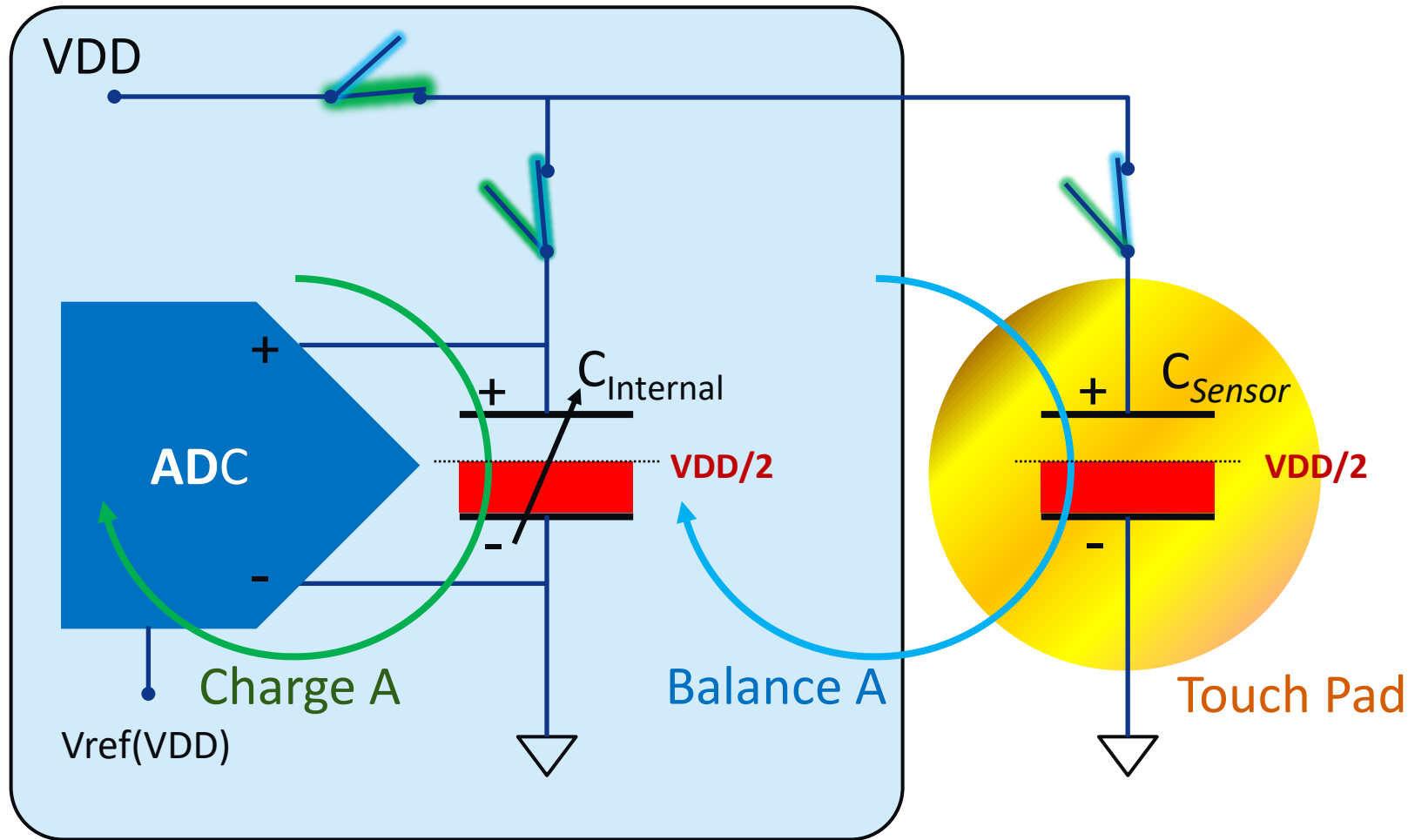


Mutual Capacitance Touch



Introduction to QTouch® Peripheral Touch Controller (PTC)

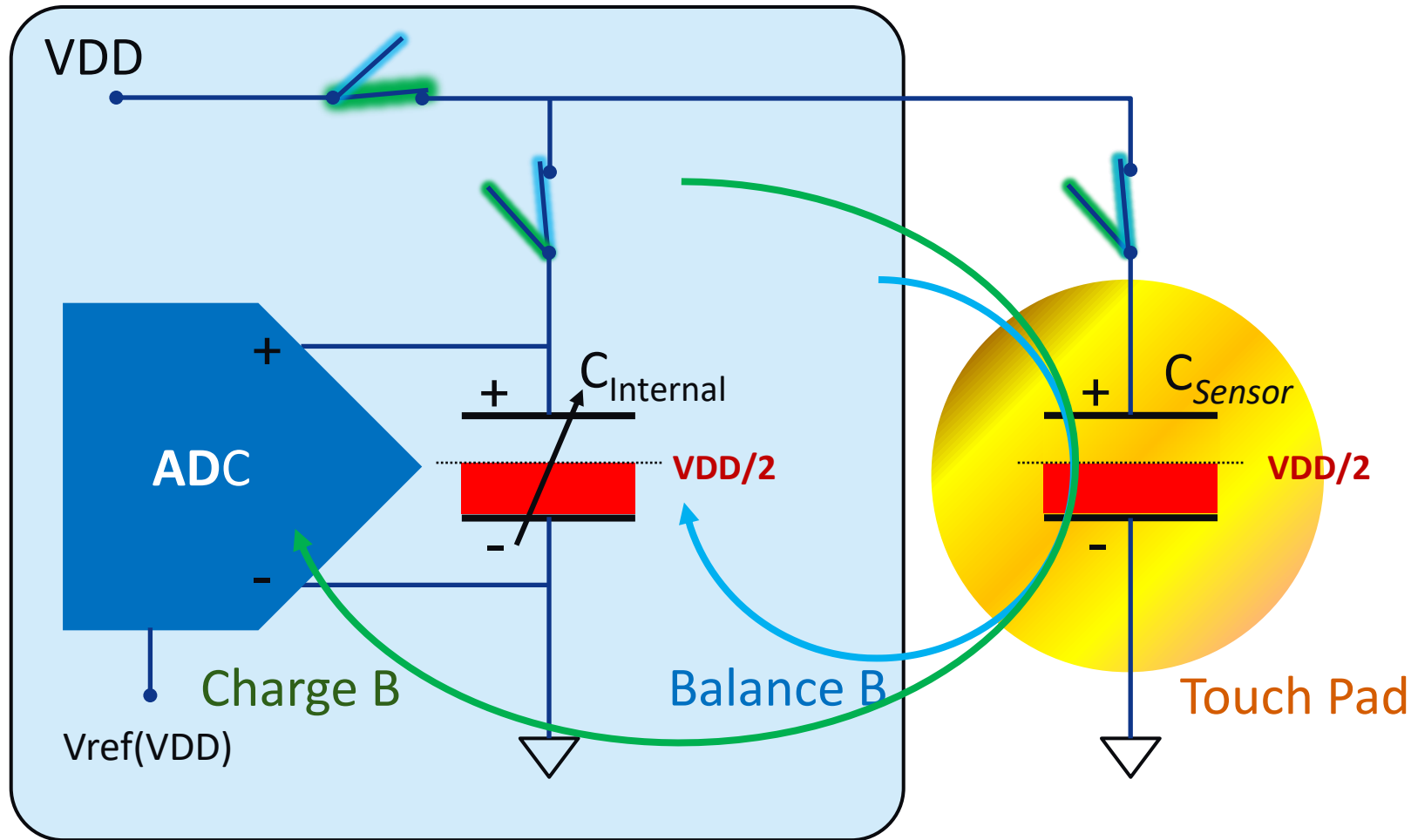
Charge Transfer



$$V_{BalanceA} = VDD/2$$

Introduction to QTouch® Peripheral Touch Controller (PTC)

Charge Transfer



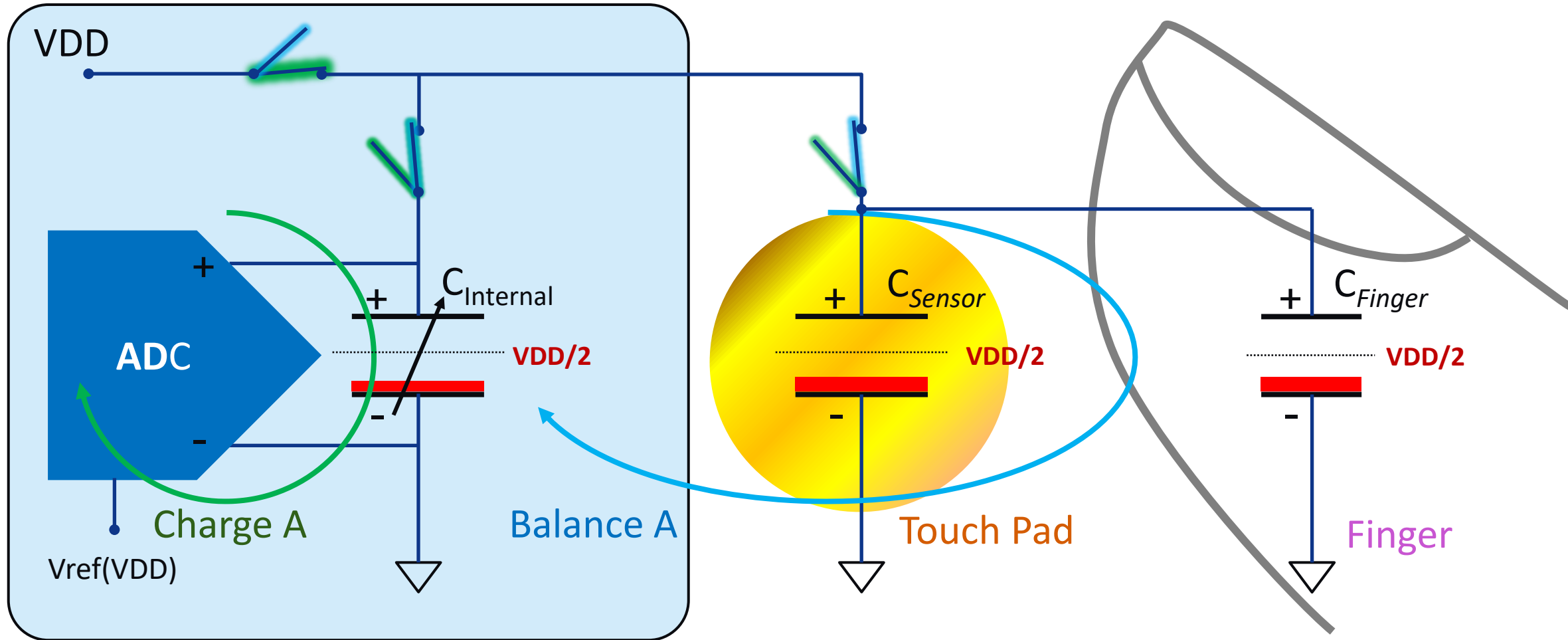
$$V_{BalanceA} = VDD/2$$

$$V_{BalanceB} = VDD/2$$

$$\text{No Touch : } V_{BalanceB} - V_{BalanceA} \approx 0$$

Introduction to QTouch® Peripheral Touch Controller (PTC)

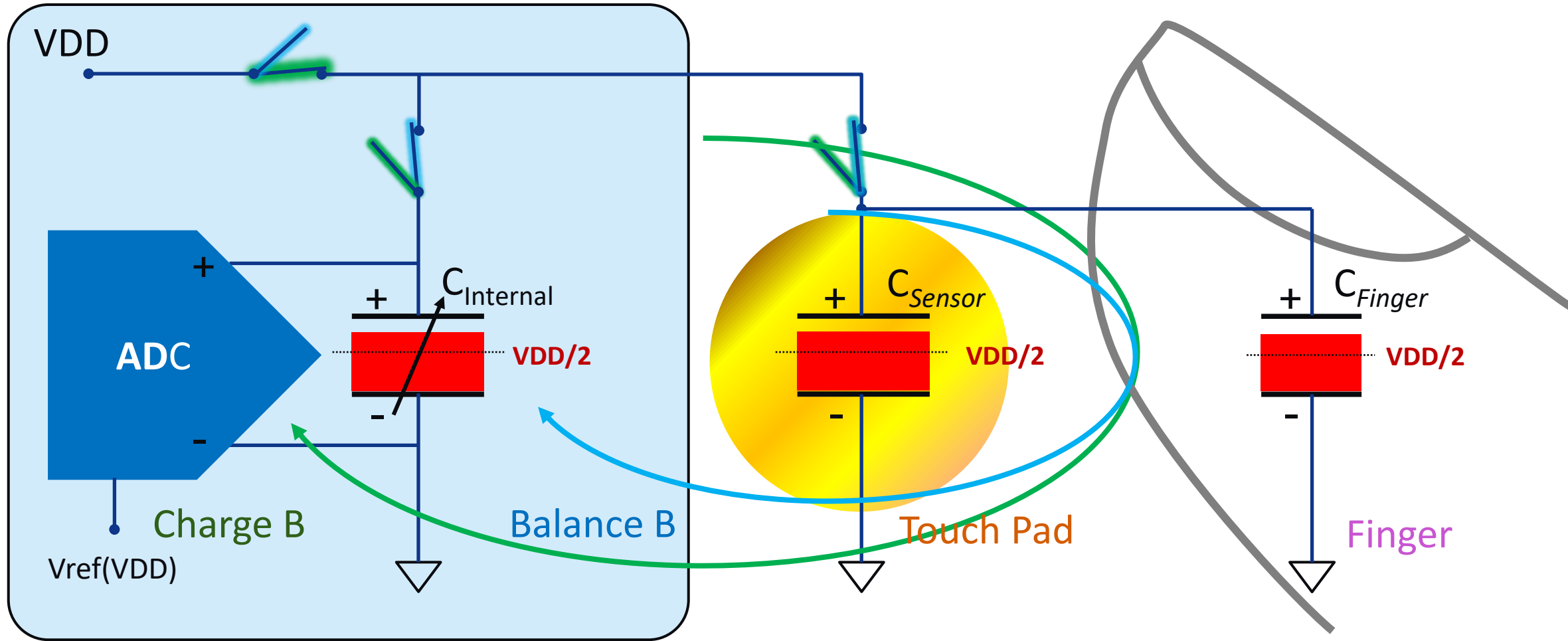
Charge Transfer



$$V_{BalanceA} < VDD/2$$

Introduction to QTouch® Peripheral Touch Controller (PTC)

Charge Transfer



$$V_{BalanceA} < VDD/2$$

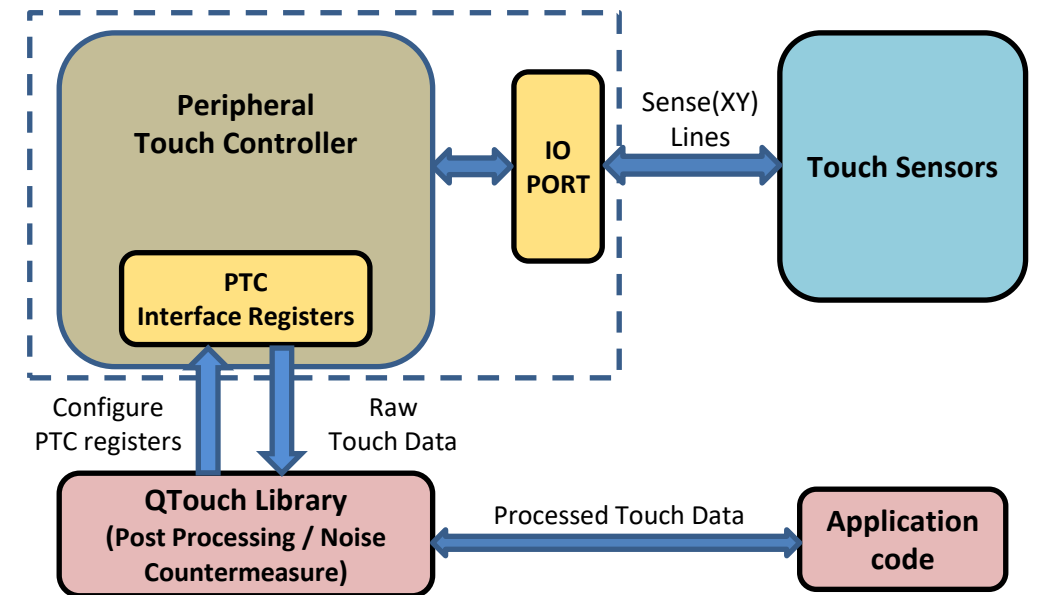
$$V_{BalanceB} > VDD/2$$

$$\text{Touched : } V_{BalanceB} - V_{BalanceA} > 0$$

Introduction to QTouch® Peripheral Touch Controller (PTC)

Peripheral Touch Controller (PTC)

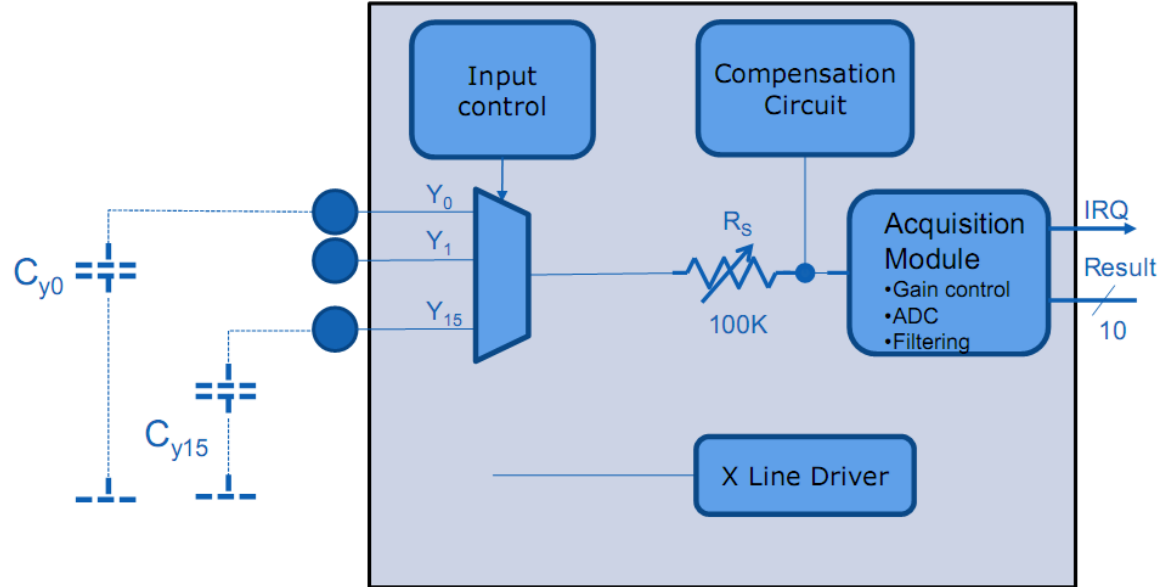
- The Peripheral Touch Controller (PTC) acquires signals in order to detect touch on capacitive sensors.
- The external capacitive touch sensor is typically formed on a PCB, and the sensor electrodes are connected to the analog front end of the PTC through the I/O pins in the device.
- The PTC supports both self- and mutual-capacitance sensors.
- In the mutual-capacitance mode, sensing is done using capacitive touch matrices in various X-Y configurations, including indium tin oxide (ITO) sensor grids. The PTC requires one pin per X-line and one pin per Y-line.
- In the self-capacitance mode, the PTC requires only one pin (Y-line) for each touch sensor. It also supports a Shield Driver to provide a driven shield on electrodes for better noise immunity and moisture tolerance.



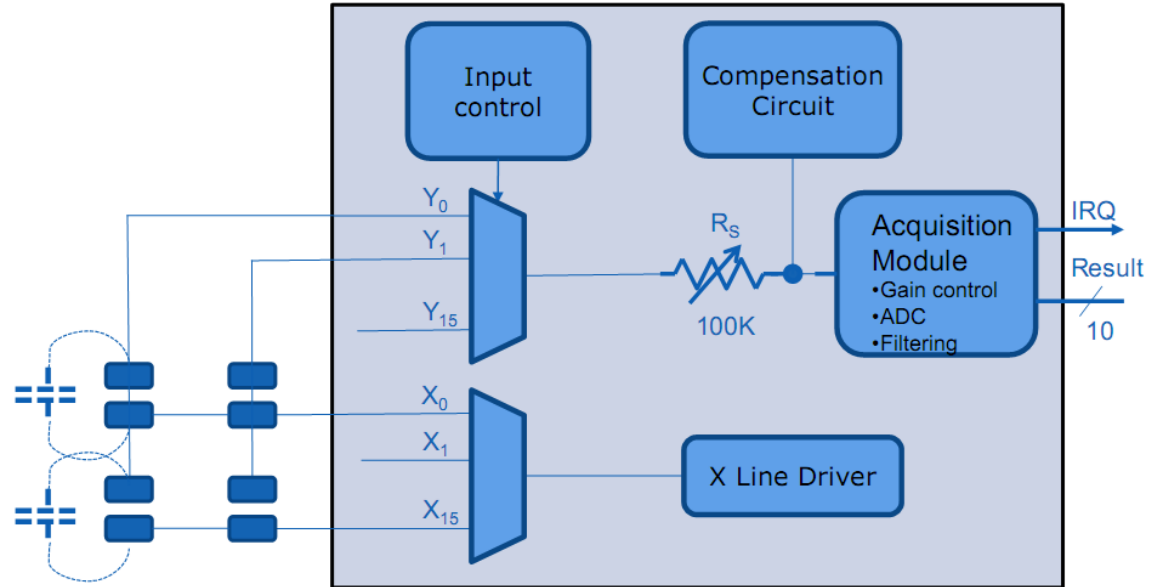
Introduction to QTouch® Peripheral Touch Controller (PTC)

PTC Block Diagram

Self Capacitance Touch



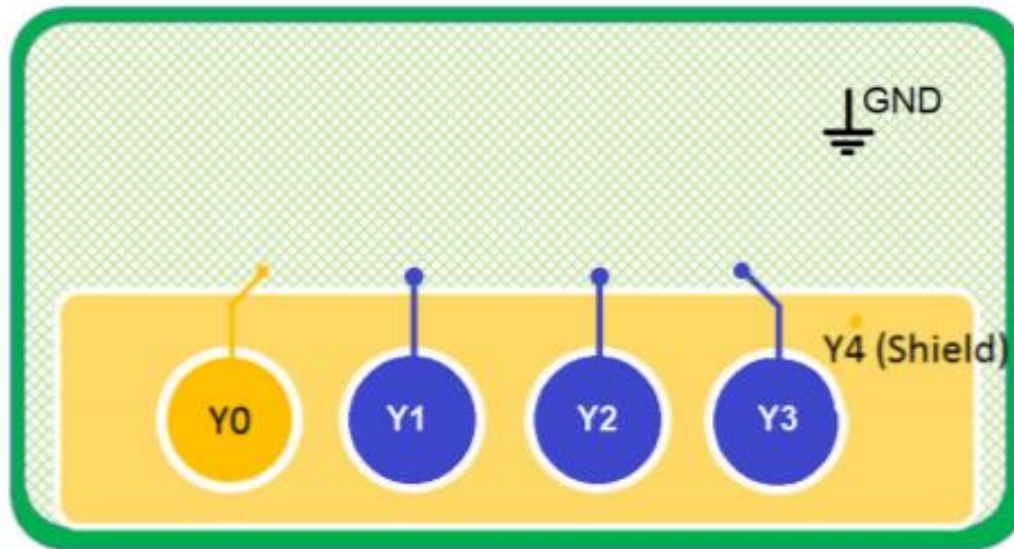
Mutual Capacitance Touch



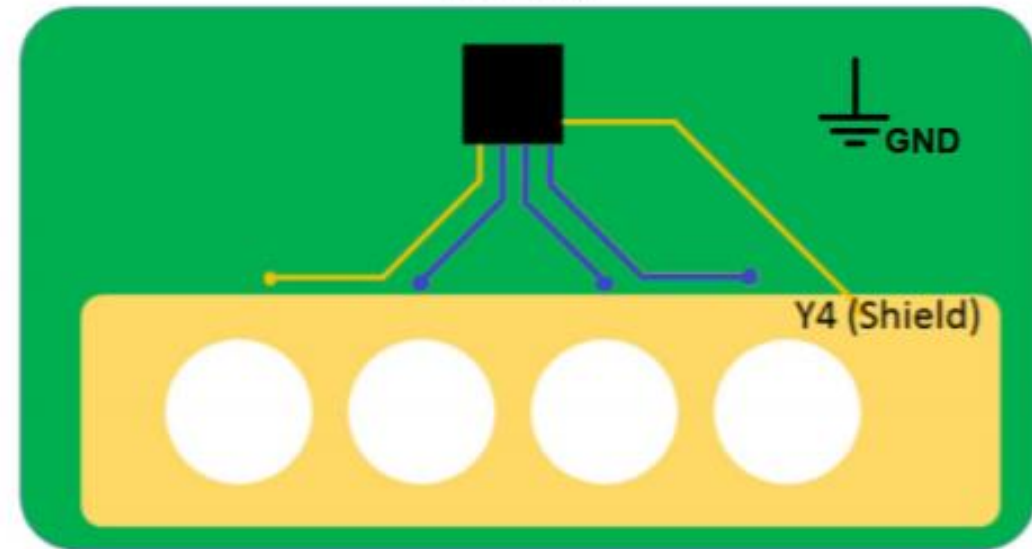
Introduction to QTouch® Peripheral Touch Controller (PTC)

Driven Shield

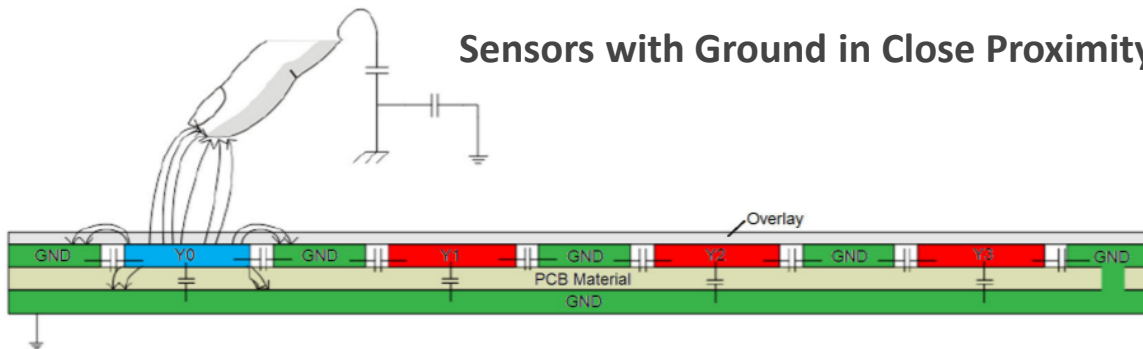
Top



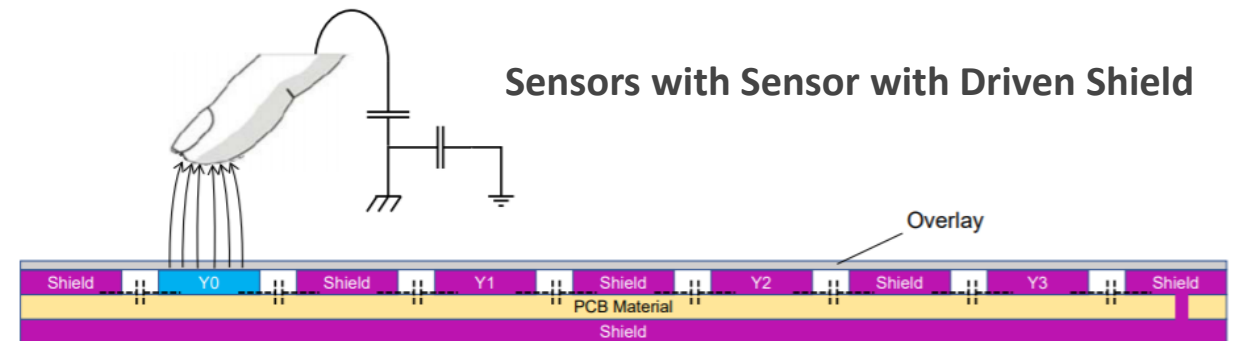
Bottom



Sensors with Ground in Close Proximity



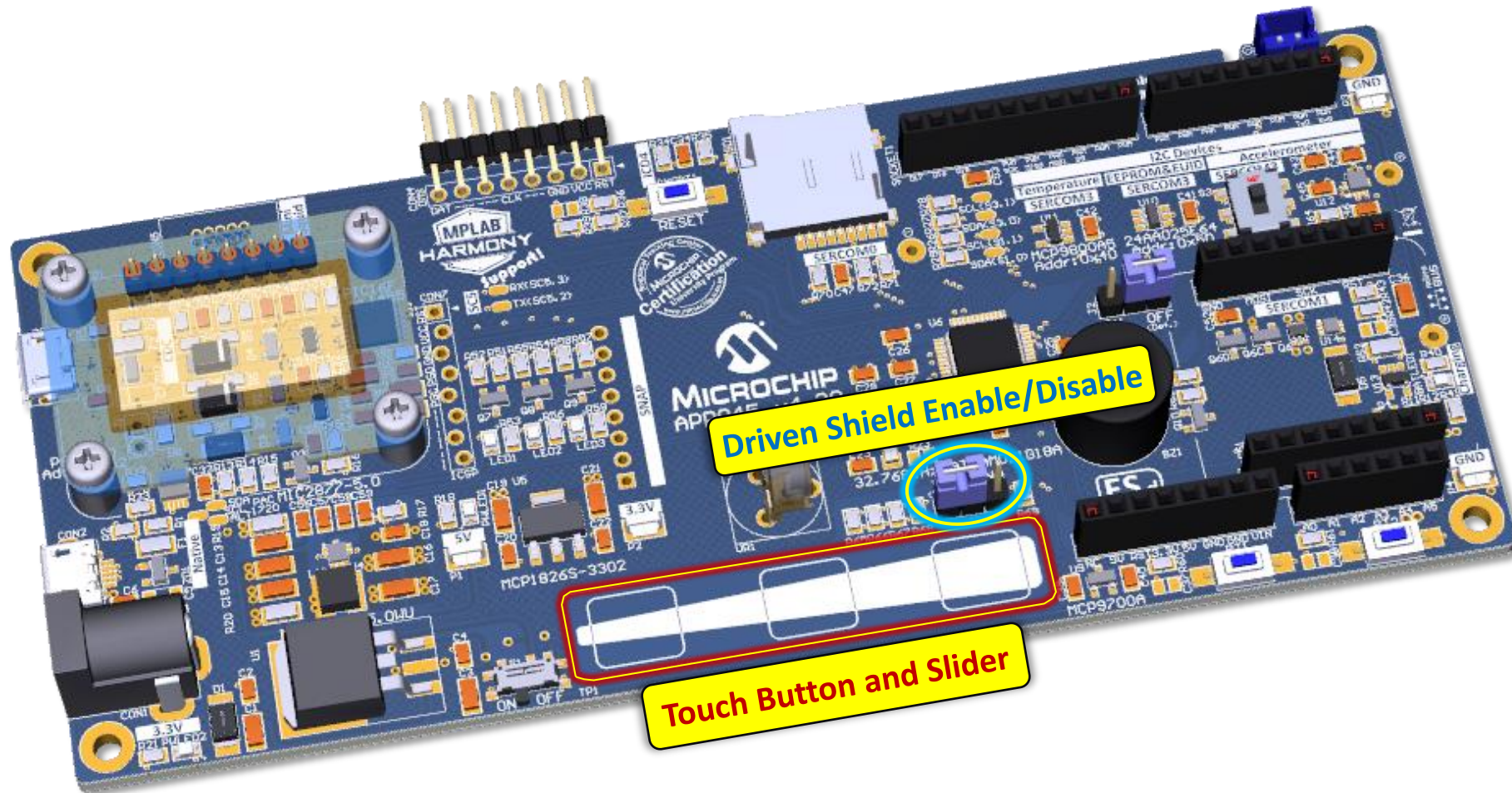
Sensors with Sensor with Driven Shield



APP045 EVB QTouch® Design and Pattern

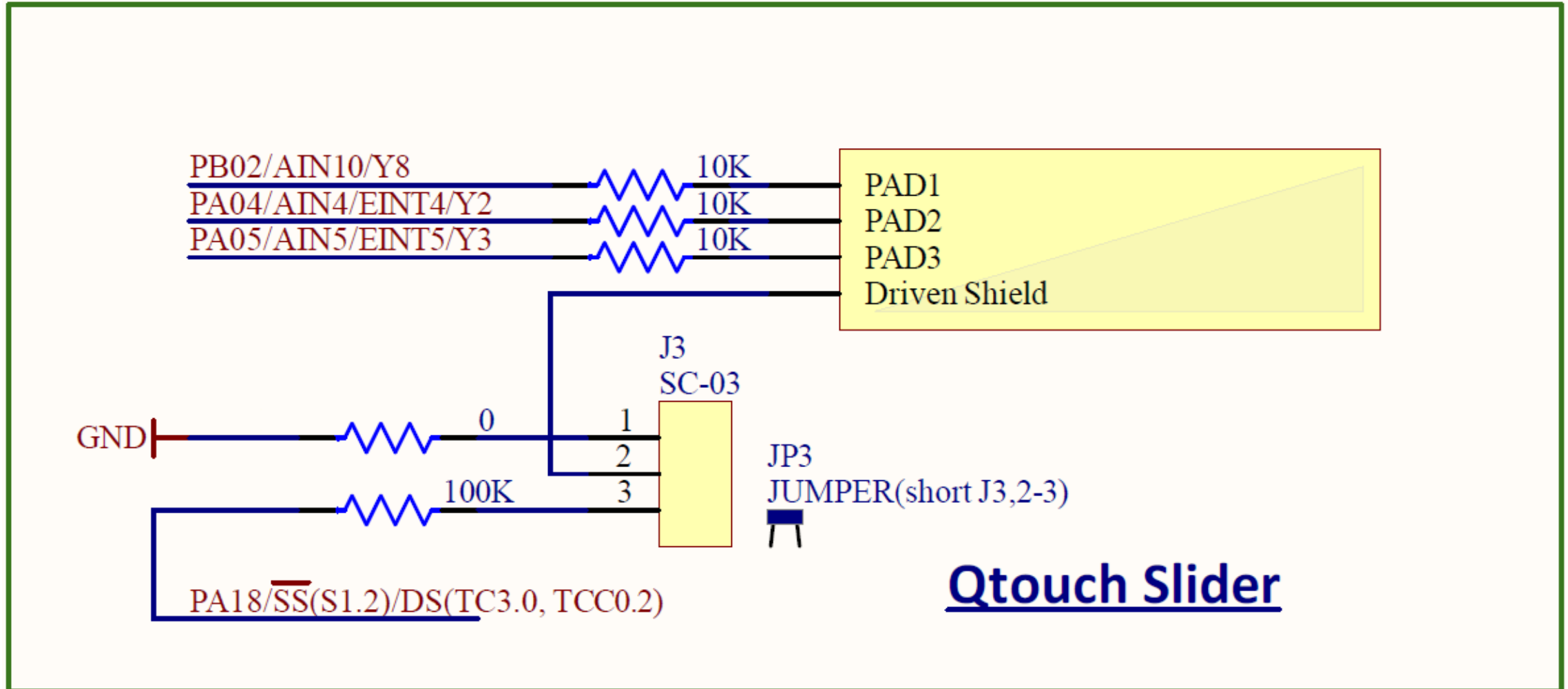
APP045 EVB QTouch® Design and Pattern

- APP045v4 EVB to designed a three channels Self Capacitance Sensing Touch Button and Touch Slider with optional Driven Shield feature.



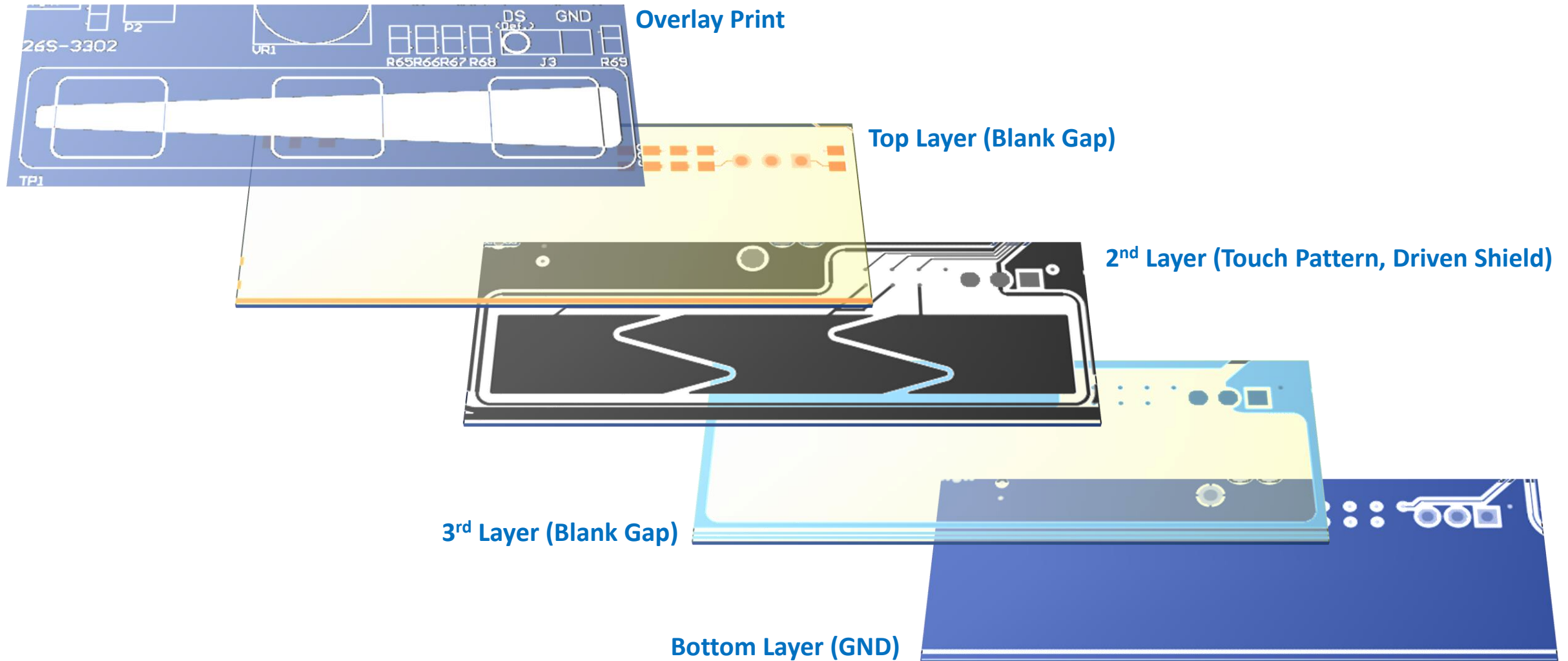
APP045 EVB QTouch® Design and Pattern

Schematic



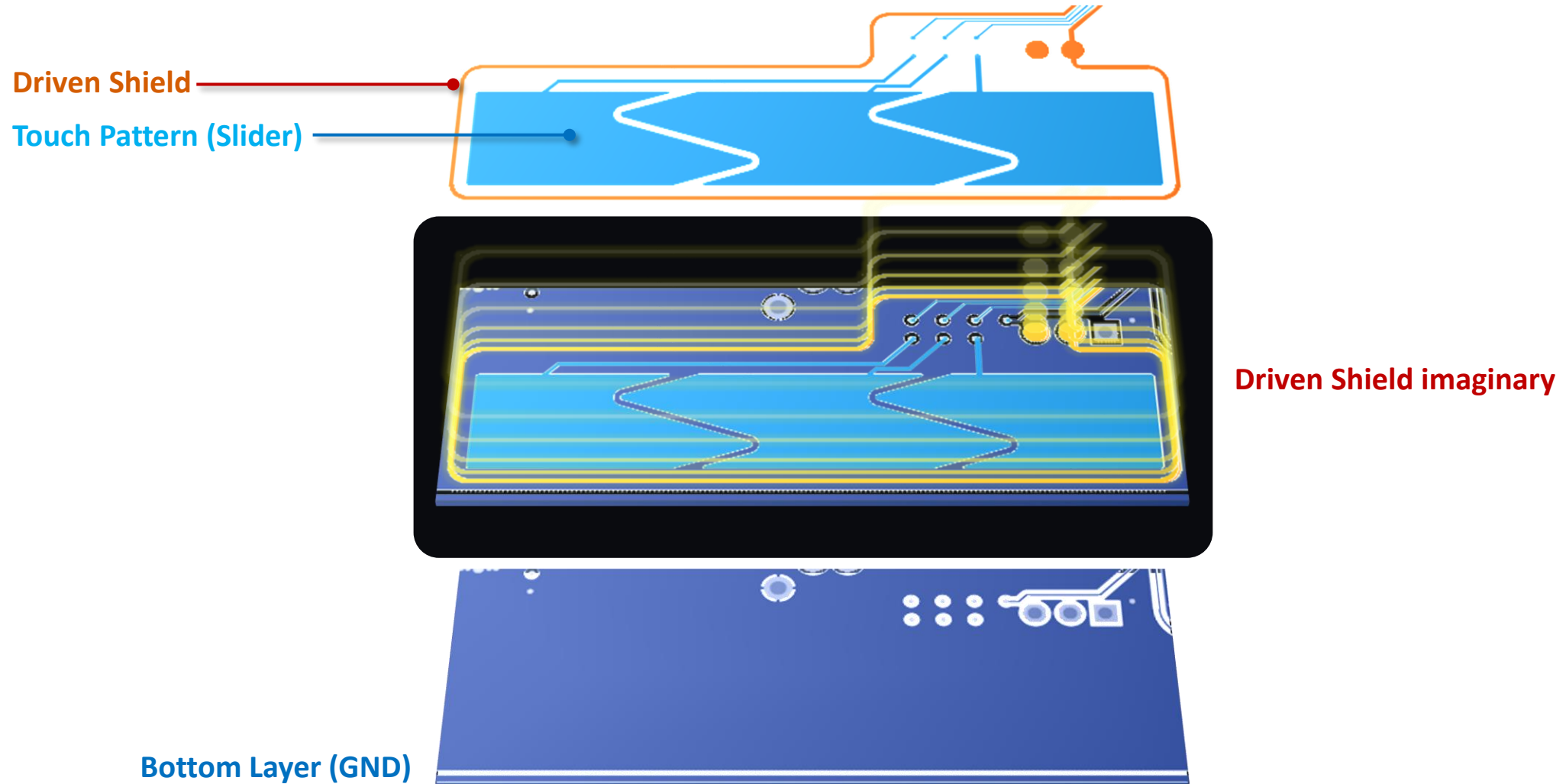
APP045 EVB QTouch® Design and Pattern

PCB Stack



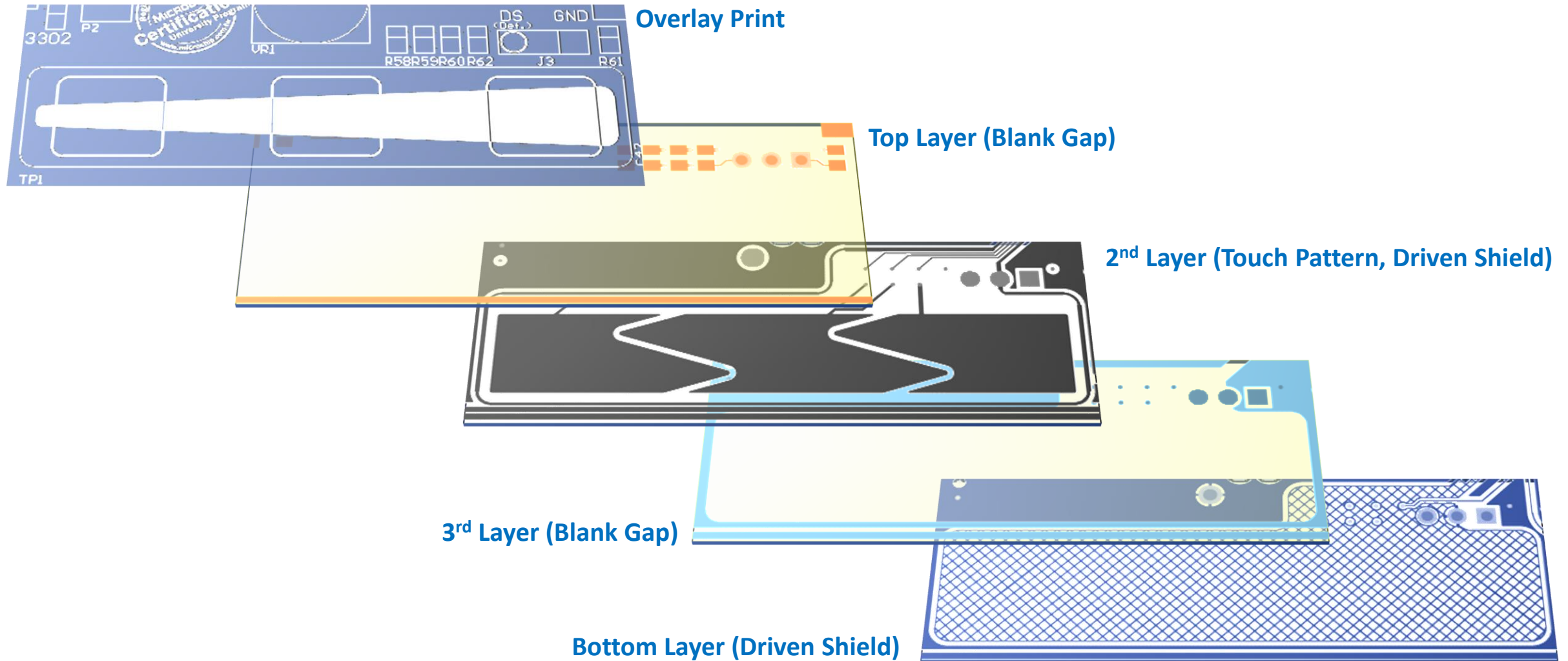
APP045 EVB QTouch® Design and Pattern

Touch Pattern and Driven Shield



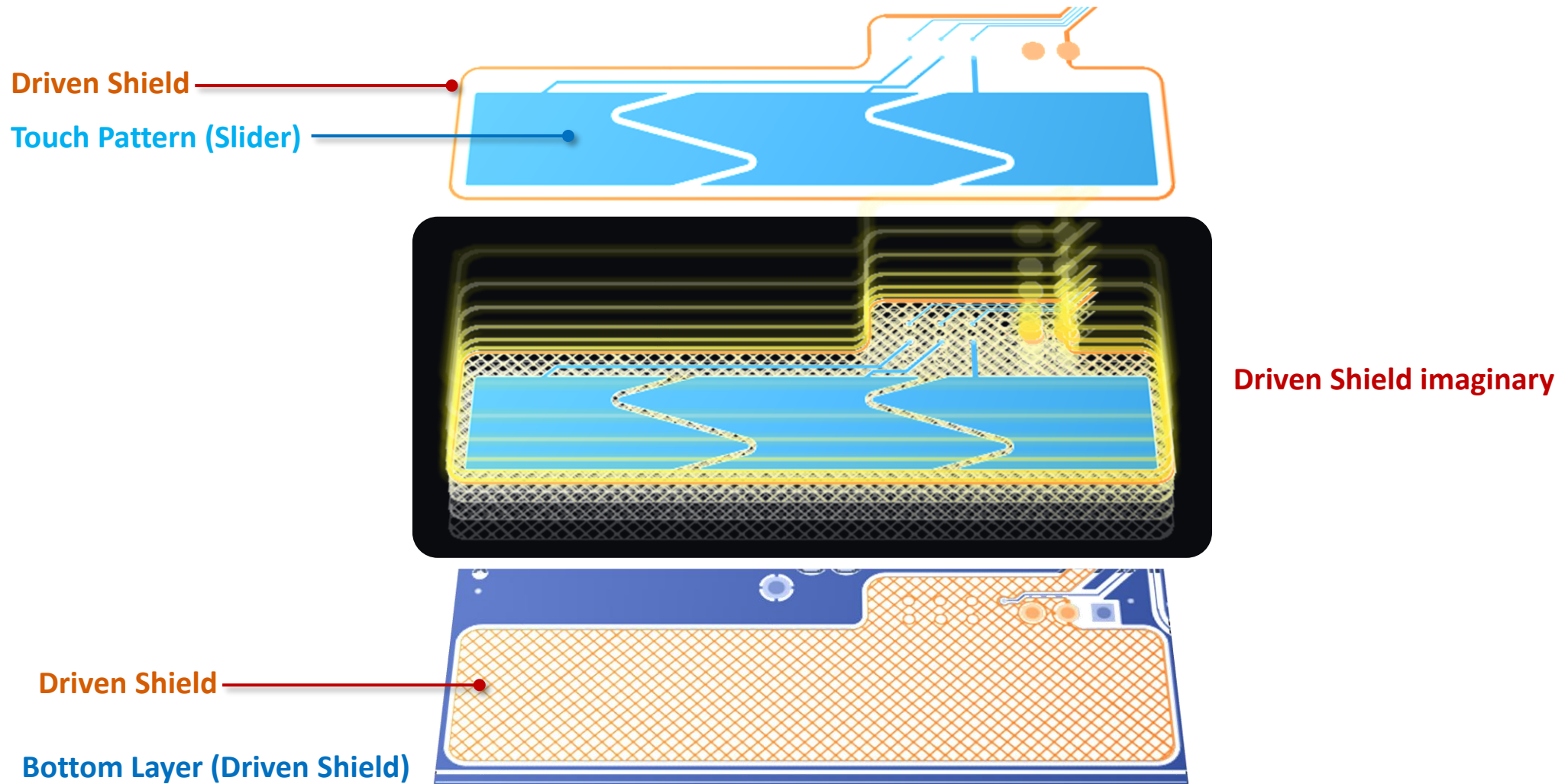
APP045 EVB QTouch® Design and Pattern

PCB Stack



APP045 EVB QTouch® Design and Pattern

Touch Pattern and Driven Shield



Harmony Touch Configurator

Harmony Touch Configurator

Create : Select Sensing Type and add Touch Sensors

Touch Configuration

Create | Configure | Tune | Summary

Configure as Low Power | Important Notes 0

Create Lump Sensor | Delete | Self-Cap ▼

Self-Cap

Mutual-Cap

Buttons: 1 | Add

Sliders: 1 | Segments: 4 | Add

Wheels: 1 | Segments: 4 | Add

Horizontal segments: 4 | Vertical segments: 4 | Gesture Type: No gestures ▼ | Add

Information

The clock for CPU and peripherals are set and Sensor pins are auto assigned on creation. Go to configure > Sensor Pins to change.

✓ Ok

Touch Configuration

Create | Configure | Tune | Summary

Buttons: 1 | Sliders: 1 | Segments: 4 | Wheels: 1 | Segments: 4 | Horizontal segments: 4 | Vertical segments: 4 | Gesture Type: No gestures ▼

Microchip

Harmony Touch Configurator

Configure – Sensor Pins : Sensors Pins signal configuration

The screenshots illustrate the 'Sensor Pins' configuration window in the Harmony Touch Configurator. The window has tabs for 'Create', 'Configure', 'Tune', and 'Summary'. The 'Configure' tab is active, and the 'Sensor Pins' section is selected.

Sensor Pins

☒ Pins view
☐ Surface view

Signals will be swapped if it's used by other sensor/segment during self-test.

Sensors	Signals
Button 0	Y0 (PA02)
Slider 0	Y1 (PA03)
	Y2 (PA04)
	Y3 (PA05)
	Y4 (PA06)
Wheel 0	Y5 (PA07)
	Y8 (PB02)
	Y9 (PB03)
	Y14 (PB08)

Note: Please click on the coordinates in any one of the corner matrix to change the orientation of the surface.

(0,255) S5 S6 S7 S8 (255,255)

S4 S3 S2 S1

(0,0) --- --- --- --- (255,0)

Y15 (PB09)

Drag and drop sensor to modify its lines. Change the multiplexed x line to y line and vice versa by clicking it. The recommended pin configuration for a mutual cap slider/wheel is a common Y line with multiple X lines. **B1** = Button 1, **S1:2** = Slider 1: Segment 2.

	X0	X1	X2	X3	X4	X5	X6	X7	X8
Y0	B0	S0:1	S0:2	S0:3	S0:4	W0:1	W0:2	W0:3	W0:4
Y1	SUR	SUR	SUR	SUR					
Y2	SUR	SUR	SUR	SUR					
Y3	SUR	SUR	SUR	SUR					
Y4	SUR	SUR	SUR	SUR					

☒ Pins view
☐ Surface view

Sensor Pins

- Sensor Parameters
- Common Parameters
- Gesture Parameters
- Driven Shield
- Boost Mode
- Frequency Hop / AFA
- Functional Safety
- Self Test (BIST)
- Host Interface

● Contact [support](#)
● Change device
● Change technology

Configure – Sensor Parameters : Filter/Gain/Clock/Threshold/AKS, etc



Harmony Touch Configurator

Configure – Common Parameters : Acquisition, Touch Drift, Recalibration threshold, etc

The screenshot displays the 'Touch Configuration' window with the 'Configure' tab selected. The 'Common Parameters' section is highlighted in the sidebar. The main area shows two sub-sections: 'Acquisition' and 'Sensor'.

Acquisition Parameters:

Parameter	Value	Control
Scan Rate (ms)	20	Up/Down arrows
PTC Interrupt Priority	3	Up/Down arrows
Acquisition Frequency	FREQ_SEL_0	Dropdown arrow

Sensor Parameters:

Parameter	Value	Control
Detect Integration	4	Up/Down arrows
Away from Touch Recal Intergration Count	5	Up/Down arrows
Away from Touch Recal Threshold	100 percent of Touch threshold	Dropdown arrow
Touch Drift Rate	20	Up/Down arrows
Away from Touch Drift Rate	5	Up/Down arrows
Drift Hold Time	20	Up/Down arrows
Re-burst mode	Reburst sensors only in process of calibration / filter in / filter out and AKS groups	Dropdown arrow
Max ON Duration	0	Up/Down arrows

Sidebar Navigation:

- Sensor Pins
- Sensor Parameters
- Common Parameters**
- Gesture Parameters
- Driven Shield
- Boost Mode
- Frequency Hop / AFA
- Functional Safety
- Self Test (BIST)
- Host Interface

Footer:

- Contact [support](#)
- Change device
- Change technology

Harmony Touch Configurator

Configure – Gesture Parameters : Gesture Tap/swipe/wheel settings

The screenshot displays the 'Touch Configuration' window with the 'Configure' tab selected. The 'Gesture Parameters' section is active, showing settings for Tap, Swipe, and Wheel gestures. The 'Enable Gestures - 1 Finger' toggle is turned on. The 'Gesture - Tap' section includes settings for Tap Release timeout (20), Tap Hold timeout (100), Touch Drift Rate (20), and Seq Tap distance threshold (50). The 'Gesture - Swipe' section includes settings for Edge Boundary (0), Swipe Timeout (70), Horizontal Swipe distance threshold (30), and Vertical Swipe distance threshold (40). The 'Gesture - Wheel' section includes a setting for Wheel Post-scaler (1). The right sidebar shows a list of configuration categories, with 'Gesture Parameters' highlighted. The bottom right corner contains links for 'Contact support', 'Change device', and 'Change technology'.

Touch Configuration

Create **Configure** Tune Summary

Gesture Parameters

Enable Gestures - 1 Finger ☒

Note: Enabling 1 Finger Gestures support only 1 Finger Gestures Support. If 1&2 Finger Gestures support is needed, delete the existing Surface sensor and add a new Surface sensor with 1&2 Finger Gestures option type.

Gesture - Tap

Tap Release timeout	20	↑ ↓
Tap Hold timeout	100	↑ ↓
Touch Drift Rate	20	↑ ↓
Seq Tap distance threshold	50	↑ ↓

Gesture - Swipe

Edge Boundary	0	↑ ↓
Swipe Timeout	70	↑ ↓
Horizontal Swipe distance threshold	30	↑ ↓
Vertical Swipe distance threshold	40	↑ ↓

Gesture - Wheel

Wheel Post-scaler	1	↑ ↓
-------------------	---	-----

Sensor Pins
Sensor Parameters
Common Parameters
Gesture Parameters
Driven Shield
Boost Mode
Frequency Hop / AFA
Functional Safety
Self Test (BIST)
Host Interface

● Contact [support](#)
● Change device
● Change technology

Configure – Driven Shield : Sensor design with external clock for Touch Nolie guard.



Harmony Touch Configurator

Configure – Frequency Hop / AFA(Automatic Frequency Adaptation)

Touch Configuration

Create **Configure** Tune Summary

Frequency Hop / AFA

Frequency Hop is a mechanism used in touch measurement to avoid noisy signal value. In Frequency Hop, more than one bursting frequency (user configurable) is used. Refer [Touch Modular Library Userguide](#) for more details on Frequency Hop.

Enable Frequency Hop / AFA

Enable Frequency Auto Tuning

AutoTune Parameters

Maximum Variance

25

Maximum Tune-in count

6

Channel Configuration

Frequency Hop Steps

3

Hop Frequency 0

Frequency 0

Hop Frequency 1

Frequency 1

Hop Frequency 2

Frequency 2

Search

Sensor Pins

Sensor Parameters

Common Parameters

Gesture Parameters

Driven Shield

Boost Mode

Frequency Hop / AFA

Functional Safety

Self Test (BIST)

Host Interface

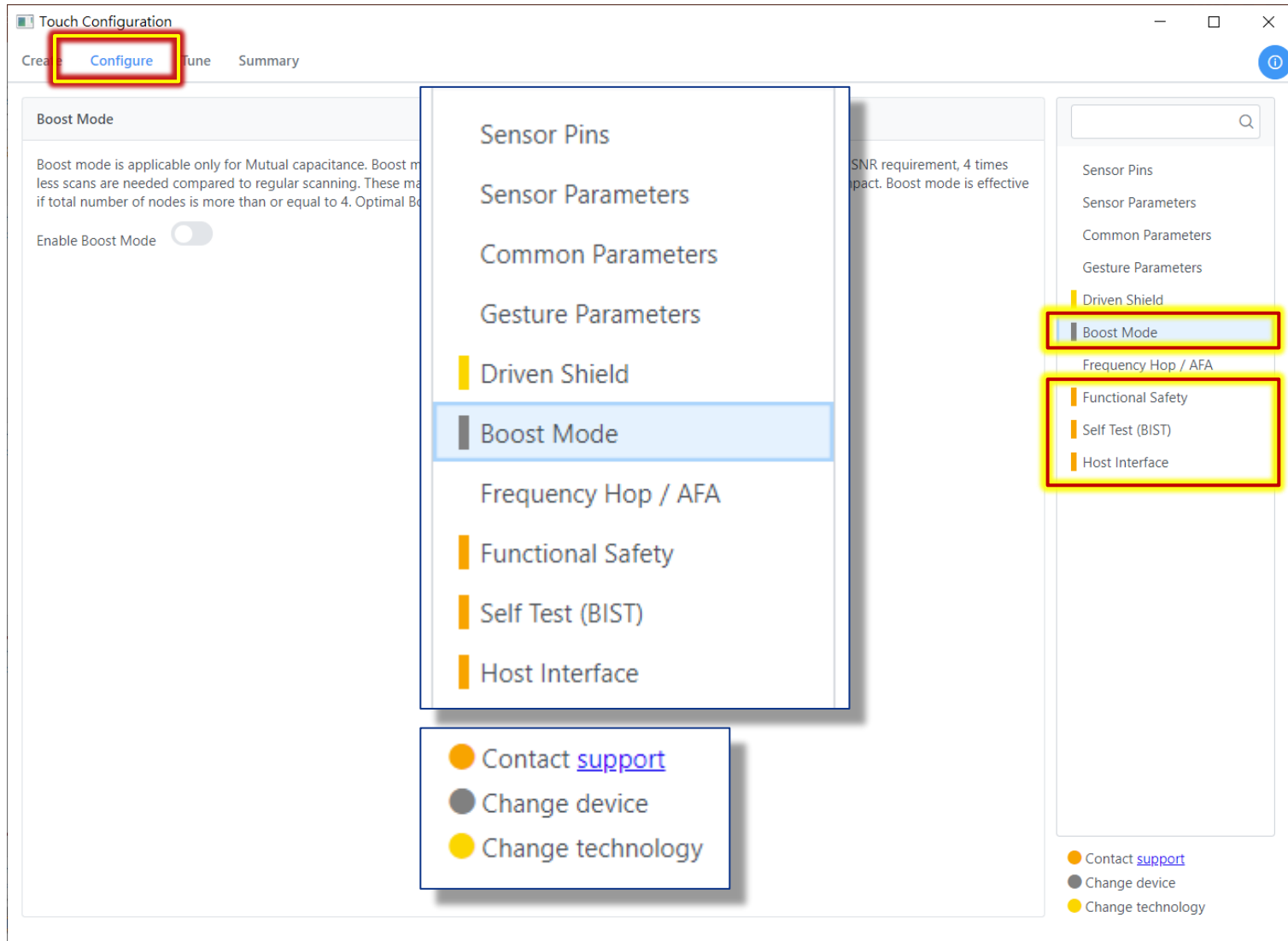
Contact [support](#)

Change device

Change technology

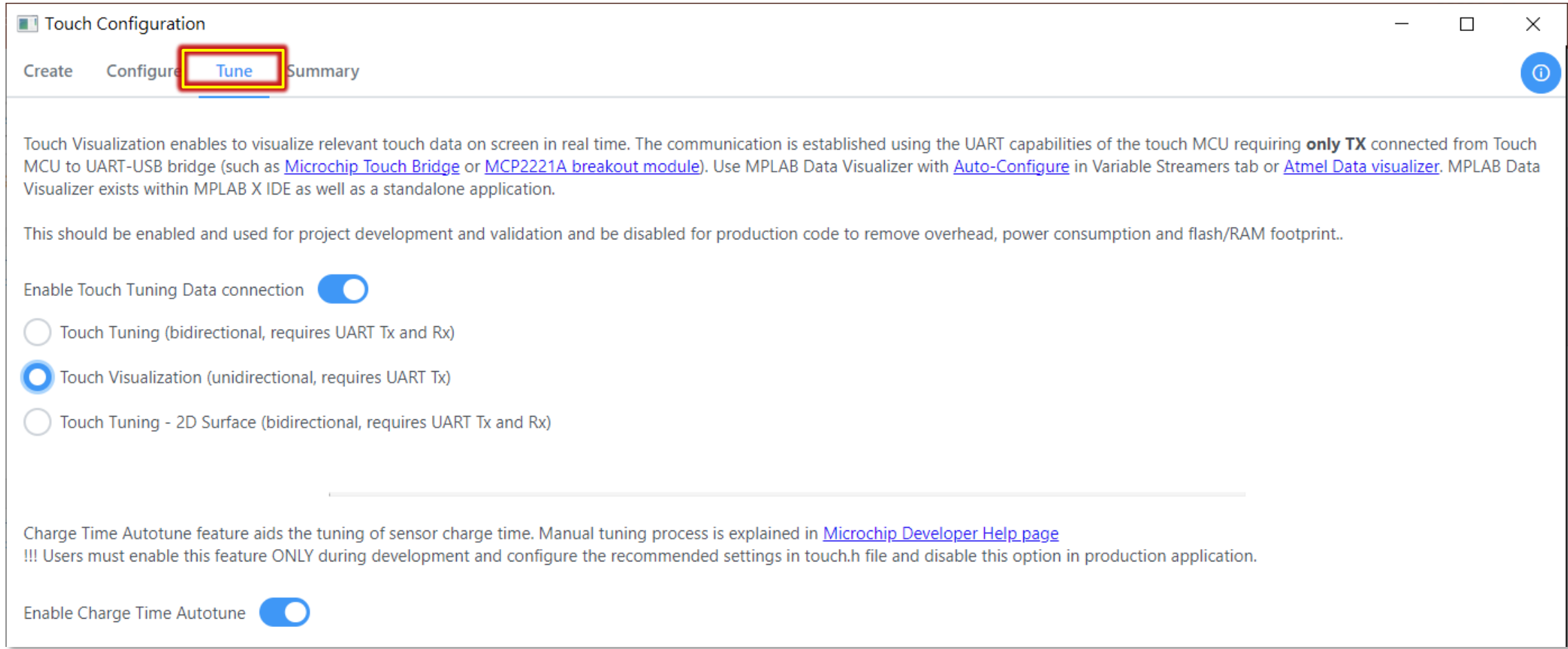
Harmony Touch Configurator

Configure – Others



Harmony Touch Configurator

Tune – Enable the debug output for Touch Tuning



The screenshot shows the 'Touch Configuration' window with the 'Tune' tab selected. The 'Tune' tab is highlighted with a red and yellow border. The window contains the following content:

Touch Visualization enables to visualize relevant touch data on screen in real time. The communication is established using the UART capabilities of the touch MCU requiring **only TX** connected from Touch MCU to UART-USB bridge (such as [Microchip Touch Bridge](#) or [MCP2221A breakout module](#)). Use MPLAB Data Visualizer with [Auto-Configure](#) in Variable Streamers tab or [Atmel Data visualizer](#). MPLAB Data Visualizer exists within MPLAB X IDE as well as a standalone application.

This should be enabled and used for project development and validation and be disabled for production code to remove overhead, power consumption and flash/RAM footprint..

Enable Touch Tuning Data connection ☒

☐ Touch Tuning (bidirectional, requires UART Tx and Rx)

☒ Touch Visualization (unidirectional, requires UART Tx)

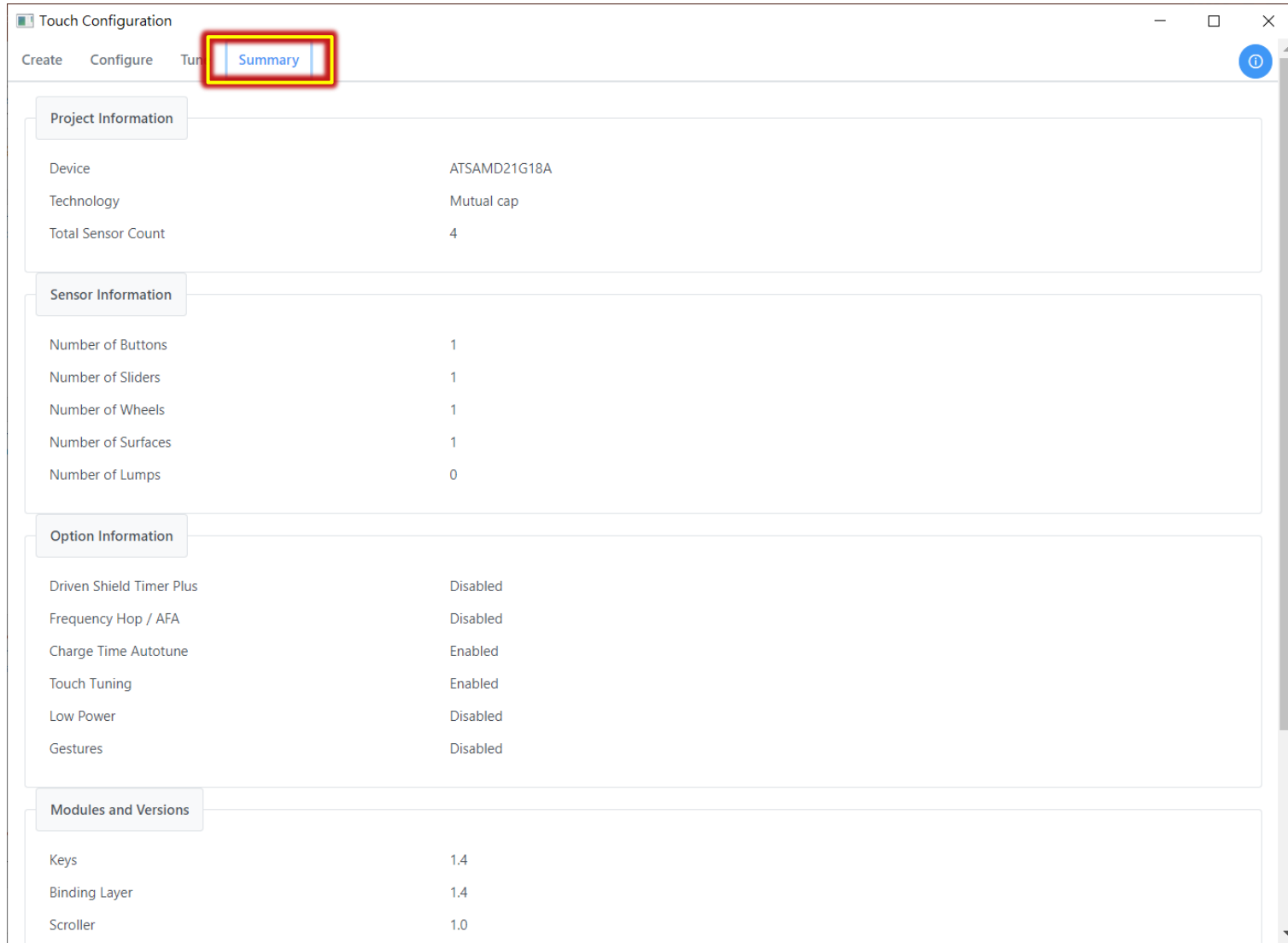
☐ Touch Tuning - 2D Surface (bidirectional, requires UART Tx and Rx)

Charge Time Autotune feature aids the tuning of sensor charge time. Manual tuning process is explained in [Microchip Developer Help page](#)
!!! Users must enable this feature ONLY during development and configure the recommended settings in touch.h file and disable this option in production application.

Enable Charge Time Autotune ☒

Harmony Touch Configurator

Summary – Configuration summary



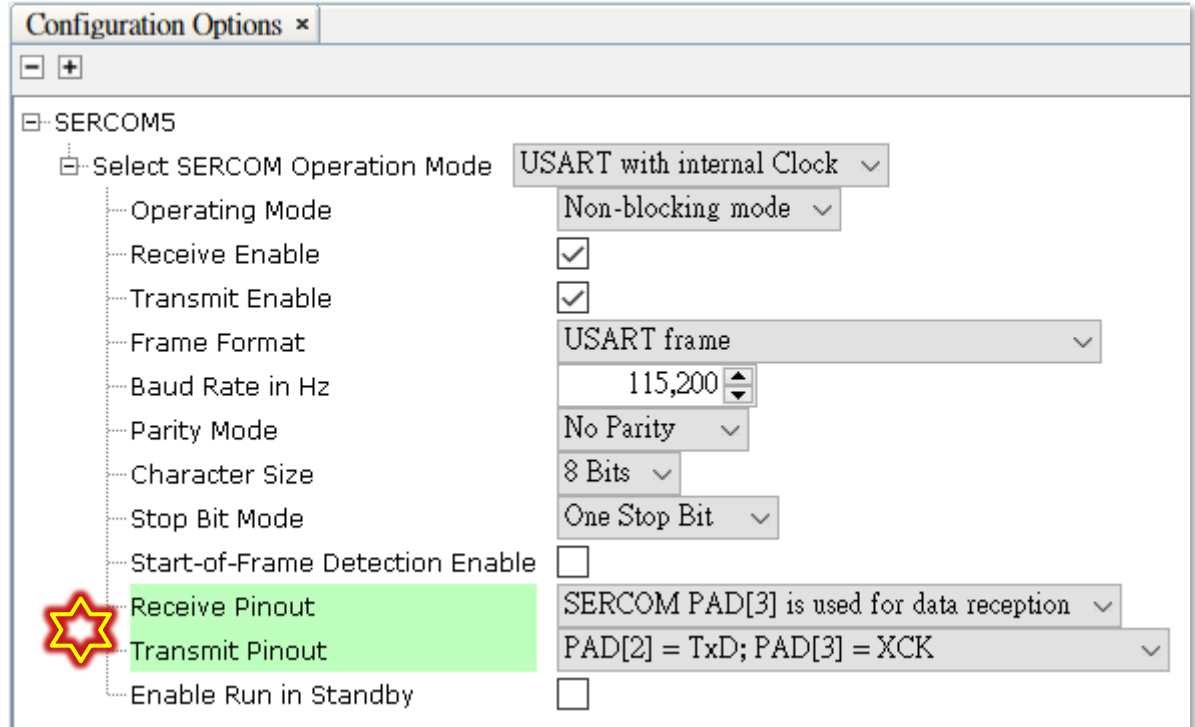
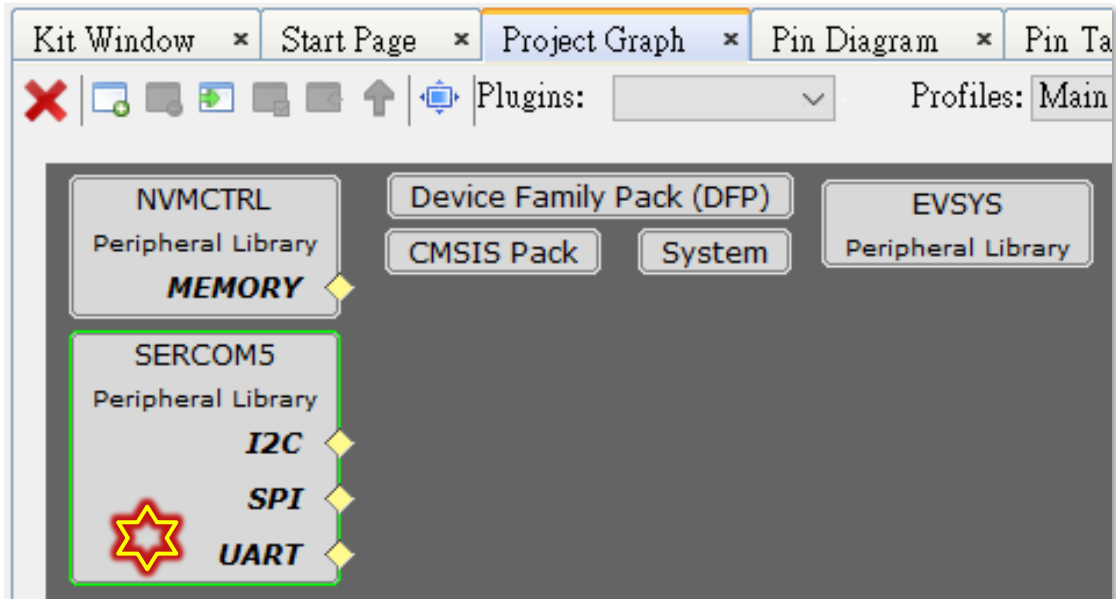
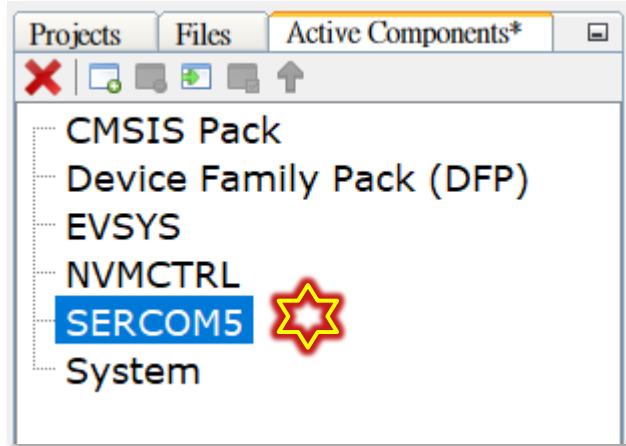
The screenshot shows the 'Summary' tab of the Harmony Touch Configurator. The 'Summary' tab is highlighted with a red and yellow border. The interface displays four sections of configuration data:

- Project Information**
 - Device: ATSAM21G18A
 - Technology: Mutual cap
 - Total Sensor Count: 4
- Sensor Information**
 - Number of Buttons: 1
 - Number of Sliders: 1
 - Number of Wheels: 1
 - Number of Surfaces: 1
 - Number of Lumps: 0
- Option Information**
 - Driven Shield Timer Plus: Disabled
 - Frequency Hop / AFA: Disabled
 - Charge Time Autotune: Enabled
 - Touch Tuning: Enabled
 - Low Power: Disabled
 - Gestures: Disabled
- Modules and Versions**
 - Keys: 1.4
 - Binding Layer: 1.4
 - Scroller: 1.0

Lab1 Touch Buttons

✖ Lab1 Touch Buttons

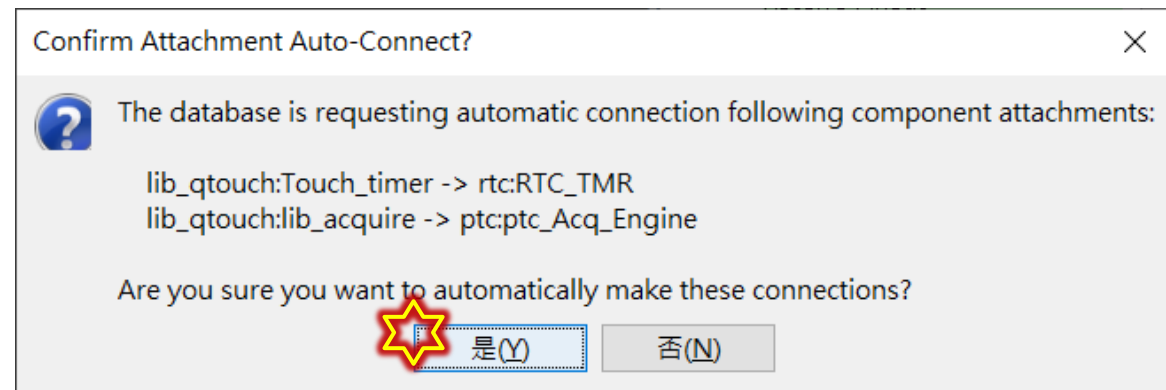
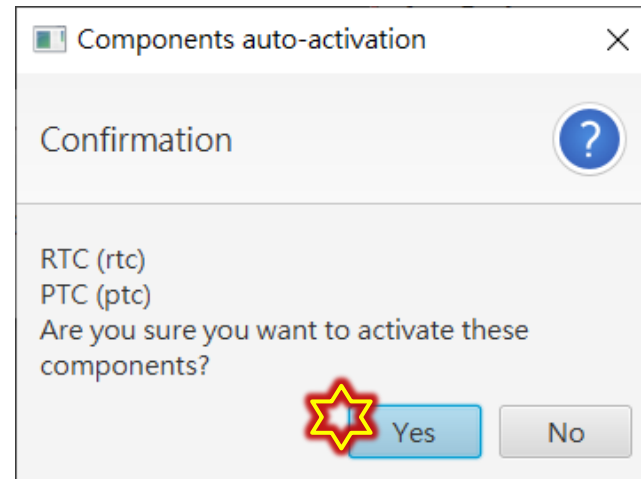
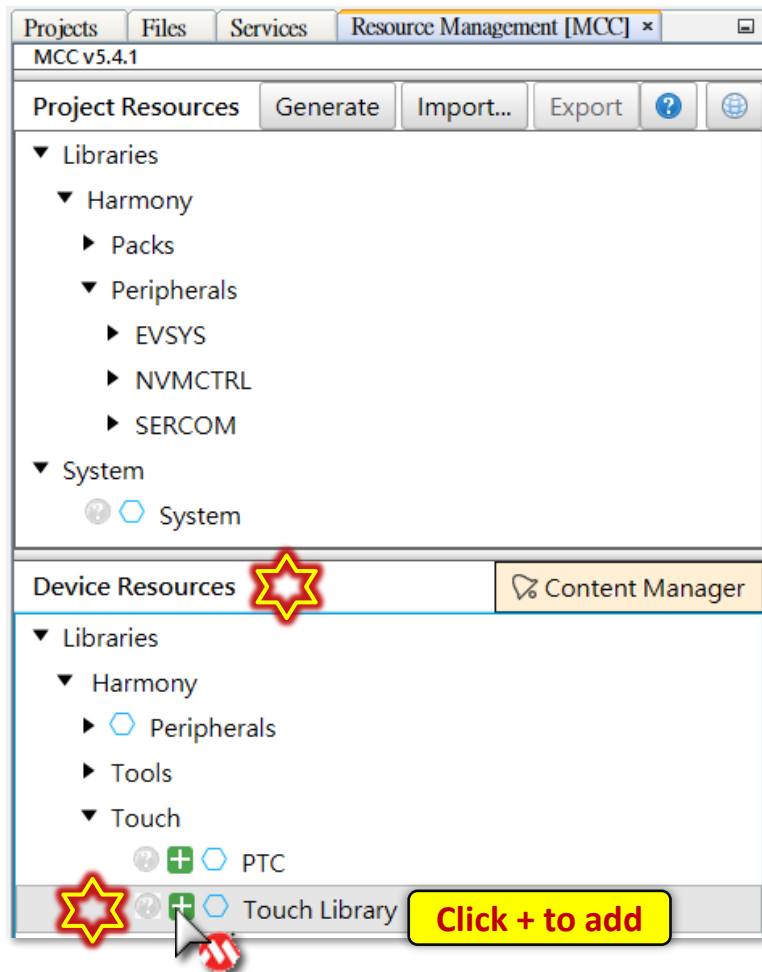
- Create a new project and add SERCOM5 in MHC for UART debug message.



Pin Number	Pin ID	Custom Name	Function
37	PB22		SERCOM5_PAD2
38	PB23		SERCOM5_PAD3

⚙️ Lab1 Touch Buttons

- Find **Touch Library** under **Touch** of **Device Resources** and add it to your project.
- Please **wait a while** after click to add the **Touch Library**.
- **RTC** and **PTC** will request to **Auto-Activation** together and **Auto-Connection**.



⚙️ Lab1 Touch Buttons

- Hint !! If you cannot find Touch Library in **Device Resources**, please go **Content Manager** to download it as below.

MPLAB X IDE v6.15 - Lab1_QTouch_Buttons : default

File Edit View Navigate Source Refactor Production Debug Team Tools Window Help

default PC: 0x0 How do I? Keyword(s)

Projects Files Services Resource Management [MCC] x

MCC v5.4.1

Project Resources Generate Import... Export ?

Libraries

- Harmony
 - Packs
- Peripherals
 - EVSYS
 - NVMCTRL
 - SERCOM
 - SERCOM5
- System
 - System

Device Resources

- DAC
- DSU
- EIC
- PM
- RAM
- RTC

Content Manager

Click Content Manager

MCC Content Manager x Kit Window x Start Page x Project Graph x Pin Diagram x Pin Table x Pin Settings x

Select Export/Import ... Submit Load Manifest Select Latest Version(s) Apply Cancel ?

Click Apply to download

Content Libraries MCC Core Versions

Type to Search Globally... Input "Touch" to search

touch

Component	Version	Status	Update progress	Description	License Agreement
Harmony 3 - Capacitive Touch solutions					
touch	v3.15.0				MPLAB Harmony license
touch_apps	v3.15.0 - [remote]				MPLAB Harmony license
touch_host_driver	v3.14.0 - [remote]				MPLAB Harmony license
	v3.13.1 - [remote]				MPLAB Harmony license
	v3.13.0 - [remote]				
	v3.12.1 - [remote]				

Select latest version

⚡ Lab1 Touch Buttons

- After add the **Touch Library** and **Auto-Connection**.
- Don't need to configure the **PTC**, **Touch Library** or **RTC**, we will use **Touch Configurator**.

The screenshot displays the Microchip Studio IDE interface. The **Project Graph** window shows the project components, including the **Touch Library** and **PTC** (Peripheral Touch Controller). The **Configuration Options** window shows the **RTC** (Real-Time Clock) settings. The **RTC** settings are expanded, showing the **RTC Operation Mode** and **RTC MODE 0 Configuration**. The **RTC** settings are marked with a red starburst icon, indicating they are not needed. The **PTC** and **Touch Library** components are also marked with red starburst icons, indicating they are not needed. The **Configuration Options** window shows the **RTC** settings, including the **RTC Operation Mode** and **RTC MODE 0 Configuration**. The **RTC** settings are marked with a red starburst icon, indicating they are not needed. The **PTC** and **Touch Library** components are also marked with red starburst icons, indicating they are not needed. The **Configuration Options** window shows the **RTC** settings, including the **RTC Operation Mode** and **RTC MODE 0 Configuration**. The **RTC** settings are marked with a red starburst icon, indicating they are not needed. The **PTC** and **Touch Library** components are also marked with red starburst icons, indicating they are not needed.

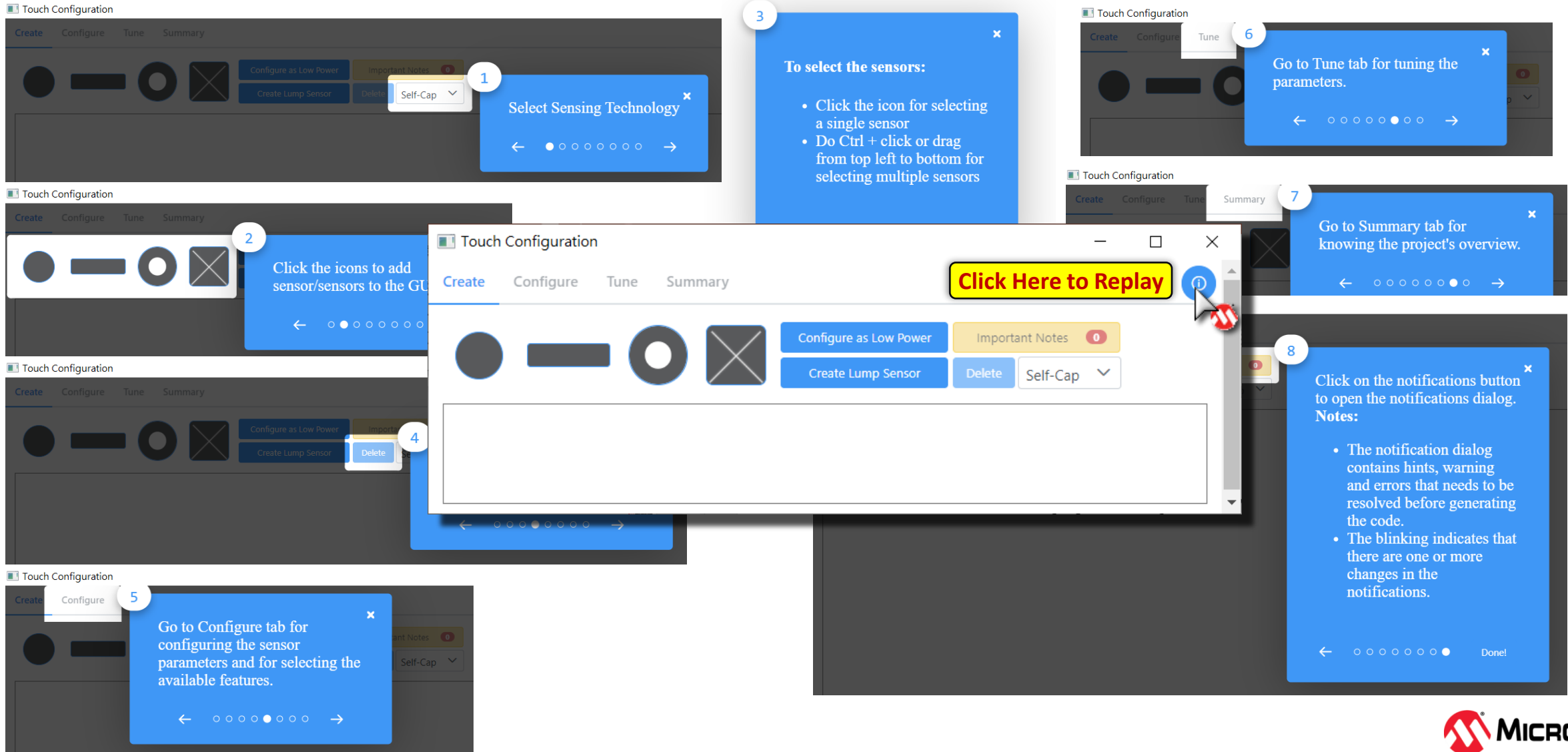
⚙️ Lab1 Touch Buttons

- Open **Touch Configuration** in **Plugins** of Project Graph.

The screenshot displays the Microchip MCC v5.4.1 software interface. On the left, the 'Project Resources' panel shows a tree view of libraries, including 'Harmony', 'Peripherals', 'RTC', 'SERCOM', 'Touch', and 'System'. The 'Touch' section is expanded, showing 'PTC' and 'Touch Library' with status icons. On the right, the 'Project Graph' window shows a block diagram of the system components. A yellow star icon is placed over the 'Plugins' dropdown menu, which is open, showing a list of configuration options: 'Clock Configuration', 'Event Configurator', 'NVIC Configuration', 'Pin Configuration', 'DMA Configuration', and 'Touch Configuration'. A yellow callout box with the text 'Click Touch Configuration' points to the 'Touch Configuration' option. The 'Project Graph' also shows various peripheral blocks like 'NVMCTRL', 'SERCOM5', 'RTC', 'PTC', and 'Touch Library' with their respective sub-components and connections.

Lab1 Touch Buttons

- An Info wizard will help you to quick understand what you could do in configurator.



1 Select Sensing Technology

2 Click the icons to add sensor/sensors to the GUI

3 To select the sensors:

- Click the icon for selecting a single sensor
- Do Ctrl + click or drag from top left to bottom for selecting multiple sensors

4

5 Go to Configure tab for configuring the sensor parameters and for selecting the available features.

6 Go to Tune tab for tuning the parameters.

7 Go to Summary tab for knowing the project's overview.

8 Click on the notifications button to open the notifications dialog.

Notes:

- The notification dialog contains hints, warning and errors that needs to be resolved before generating the code.
- The blinking indicates that there are one or more changes in the notifications.

Click Here to Replay

⚙️ Lab1 Touch Buttons

- In **Create** Tab, click on Button Icon and input number **Buttons** to **3**.
- Click **Add** then a Warning dialog will show up to notice “The clock for CPU and peripherals are set and Sensor pins are auto assigned on creation”.

1 Touch Configuration

2 Buttons

3 3

4 Add

5 Information

The clock for CPU and peripherals are set and Sensor pins are auto assigned on creation.
Go to configure > Sensor Pins to change.

Ok

0 1 2

✂ Lab1 Touch Buttons

- In **Configure** Tab, Click on **Sensor Pins** and configure Buttons Signals pins as below.

- Button0 (Signals) : Y8 (PB02)
- Button1 (Signals) : Y2 (PA04)
- Button2 (Signals) : Y3 (PA05)

Touch Configuration

Create **Configure** Tune Summary

Sensor Pins

Signals will be swapped if it's used by other sensor/segment during selection

Sensors	Signals
Button 0	Y8 (PB02)
Button 1	Y2 (PA04)
Button 2	Y3 (PA05)

Sensor Pins

Sensor Parameters

Common Parameters

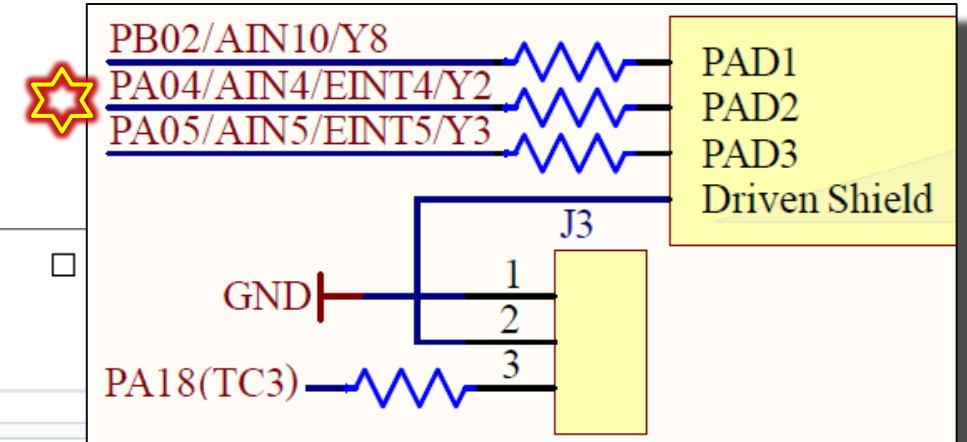
Driven Shield

Boost Mode

Frequency Hop / AFA

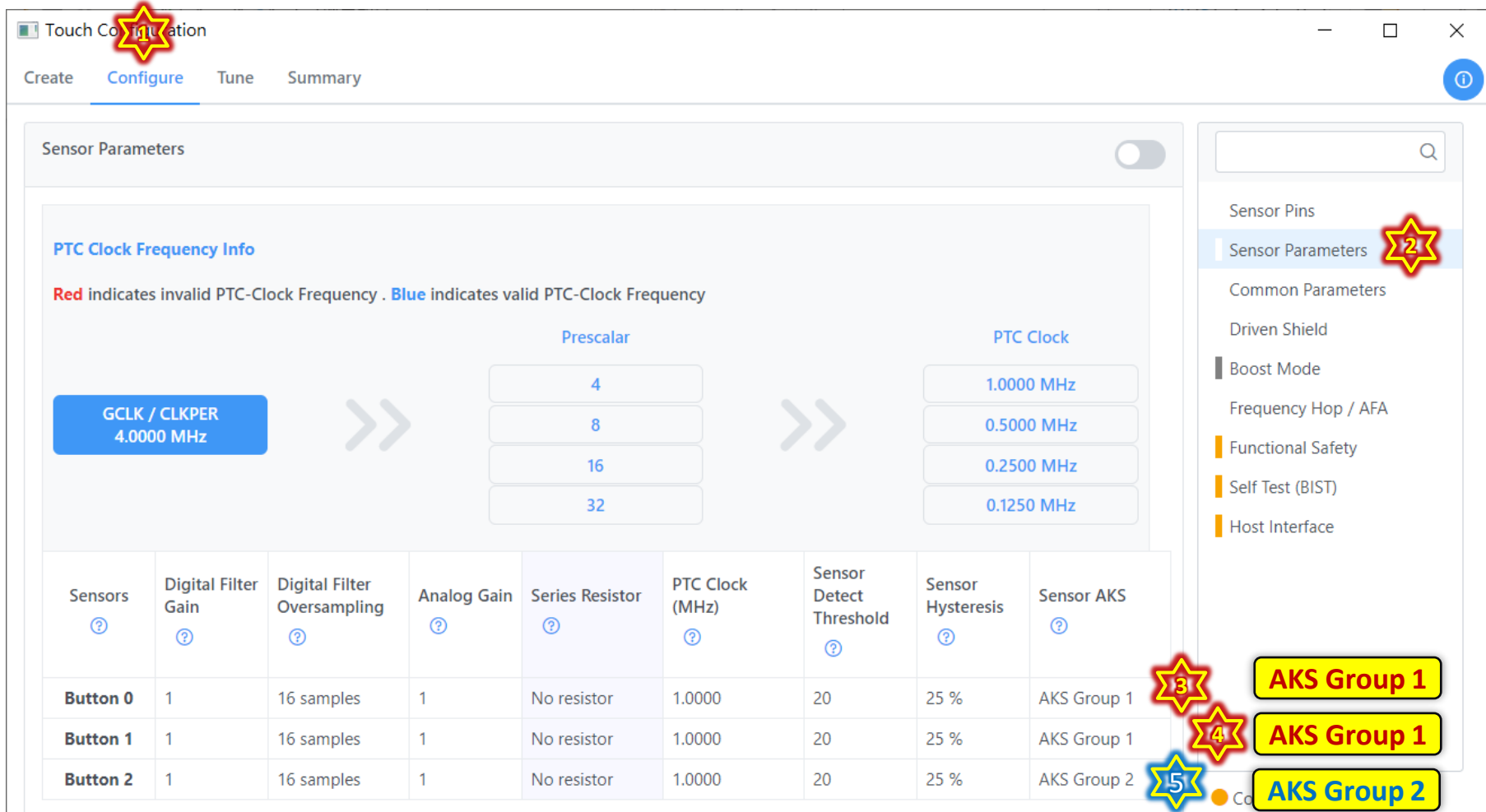
Functional Safety

Self Test (BIST)



Lab1 Touch Buttons

- In **Configure** Tab, Click on **Sensor Parameters** and configure **Sensor AKS(Adjacent Keys Suppression)** group as below.
- Button0 : **AKS Group 1** , Button1 : **AKS Group 1** and Button2: **AKS Group 2**



Touch Configuration

Create **Configure** Tune Summary

Sensor Parameters

PTC Clock Frequency Info

Red indicates invalid PTC-Clock Frequency . Blue indicates valid PTC-Clock Frequency

Prescalar

PTC Clock

GCLK / CLKPER 4.0000 MHz

Sensors	Digital Filter Gain	Digital Filter Oversampling	Analog Gain	Series Resistor	PTC Clock (MHz)	Sensor Detect Threshold	Sensor Hysteresis	Sensor AKS
Button 0	1	16 samples	1	No resistor	1.0000	20	25 %	AKS Group 1
Button 1	1	16 samples	1	No resistor	1.0000	20	25 %	AKS Group 1
Button 2	1	16 samples	1	No resistor	1.0000	20	25 %	AKS Group 2

Sensor Pins

Sensor Parameters

Common Parameters

Driven Shield

Boost Mode

Frequency Hop / AFA

Functional Safety

Self Test (BIST)

Host Interface

AKS Group 1

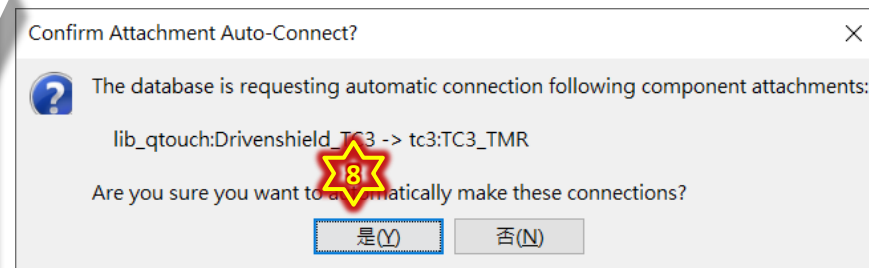
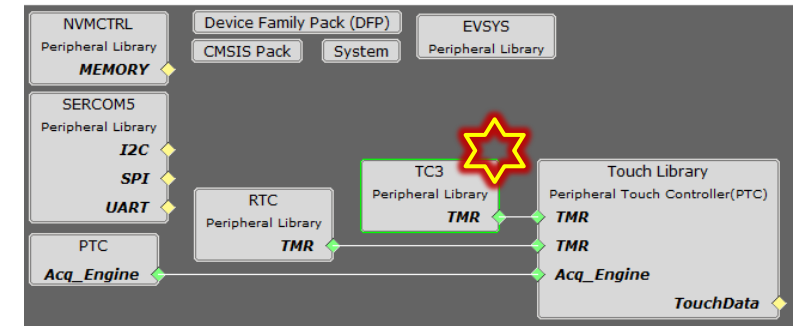
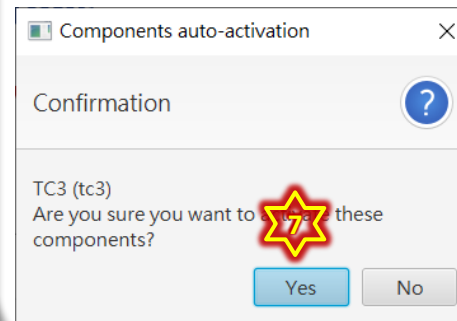
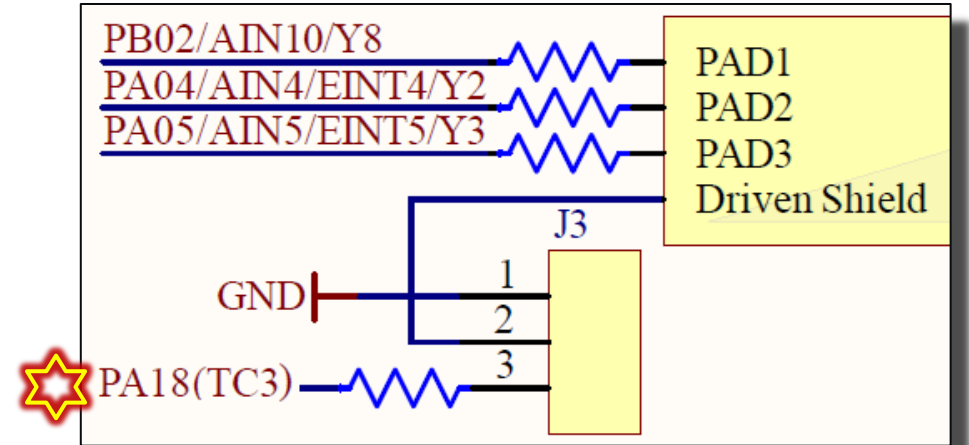
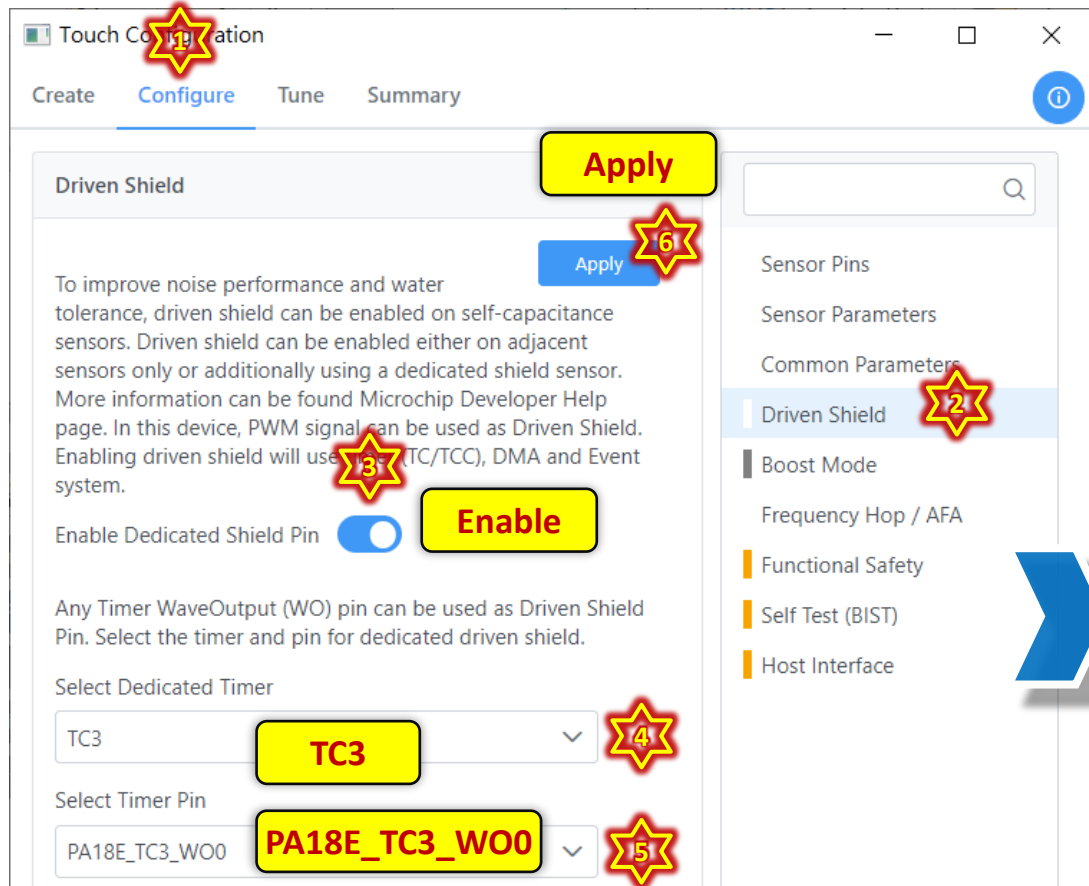
AKS Group 1

AKS Group 2

✖ Lab1 Touch Buttons

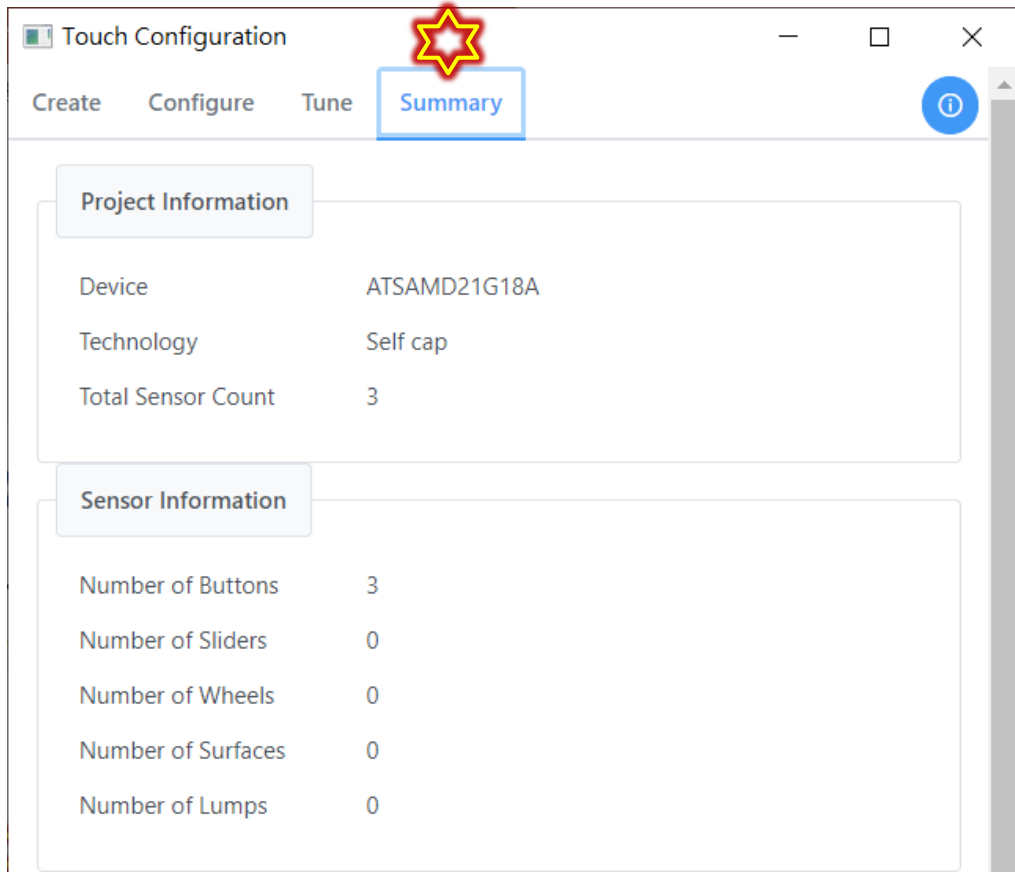
- In **Configure** Tab, Click on **Driven Shield** and configure Driven Shield as below.

- Enable Dedicated Driven Shield : 
- Select Dedicated Timer : TC3
- Select Timer Pin : PA18E_TC3_WO0



Lab1 Touch Buttons

- In **Summary** Tab, to figure out current configuration of Touch.



Touch Configuration

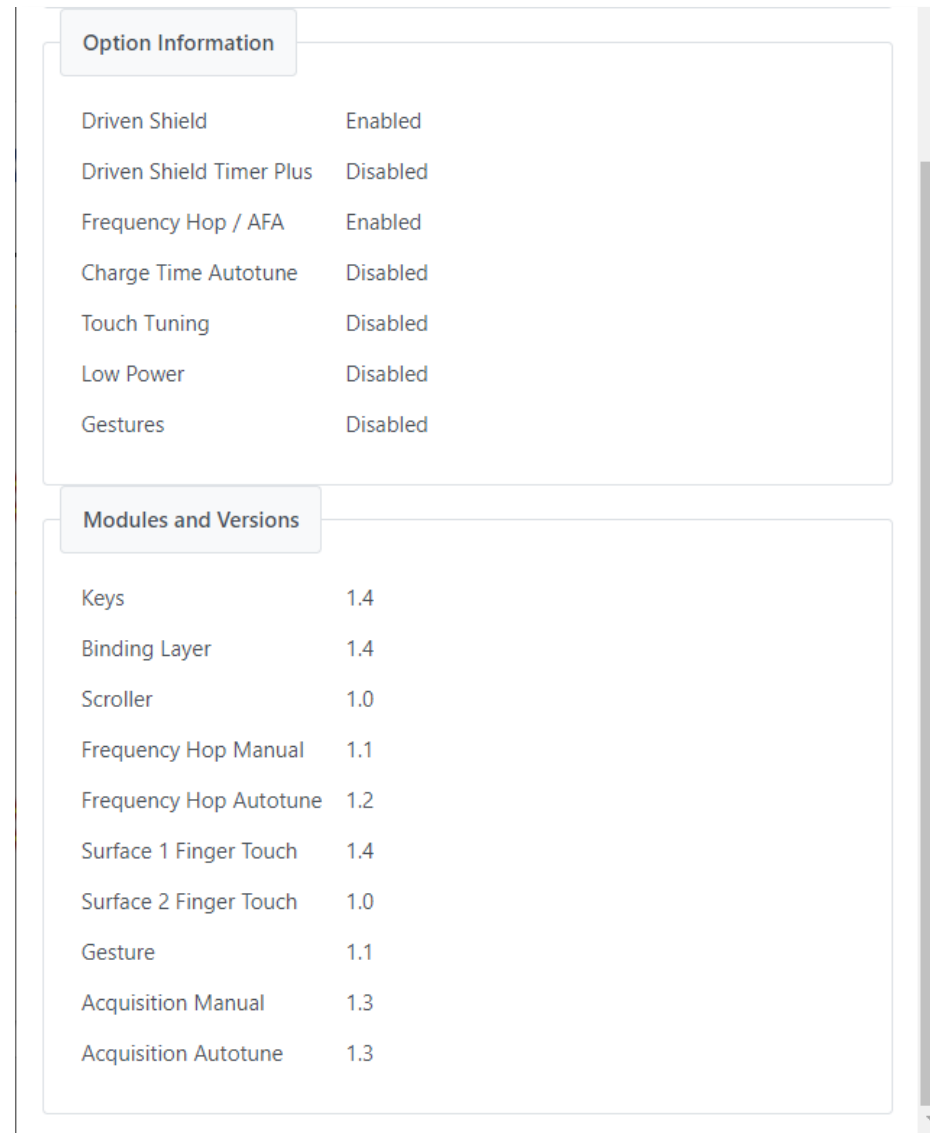
Create Configure Tune **Summary**

Project Information

Device	ATSAMD21G18A
Technology	Self cap
Total Sensor Count	3

Sensor Information

Number of Buttons	3
Number of Sliders	0
Number of Wheels	0
Number of Surfaces	0
Number of Lumps	0



Option Information

Driven Shield	Enabled
Driven Shield Timer Plus	Disabled
Frequency Hop / AFA	Enabled
Charge Time Autotune	Disabled
Touch Tuning	Disabled
Low Power	Disabled
Gestures	Disabled

Modules and Versions

Keys	1.4
Binding Layer	1.4
Scroller	1.0
Frequency Hop Manual	1.1
Frequency Hop Autotune	1.2
Surface 1 Finger Touch	1.4
Surface 2 Finger Touch	1.0
Gesture	1.1
Acquisition Manual	1.3
Acquisition Autotune	1.3

Lab1 Touch Buttons

- Check the Touch Sensor Pins in Plugin **Pin Configuration** of Project Graph as below.

Kit Window × Start Page × Project Graph × Pin Diagram × Pin Table × Pin Settings ×

Order: Pins Table View ☒ Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
9	PA04		PTC_Y2	Analog	High Impedance	n/a	☐	☐	NORMAL
10	PA05		PTC_Y3	Analog	High Impedance	n/a	☐	☐	NORMAL
47	PB02		PTC_Y8	Analog	High Impedance	n/a	☐	☐	NORMAL

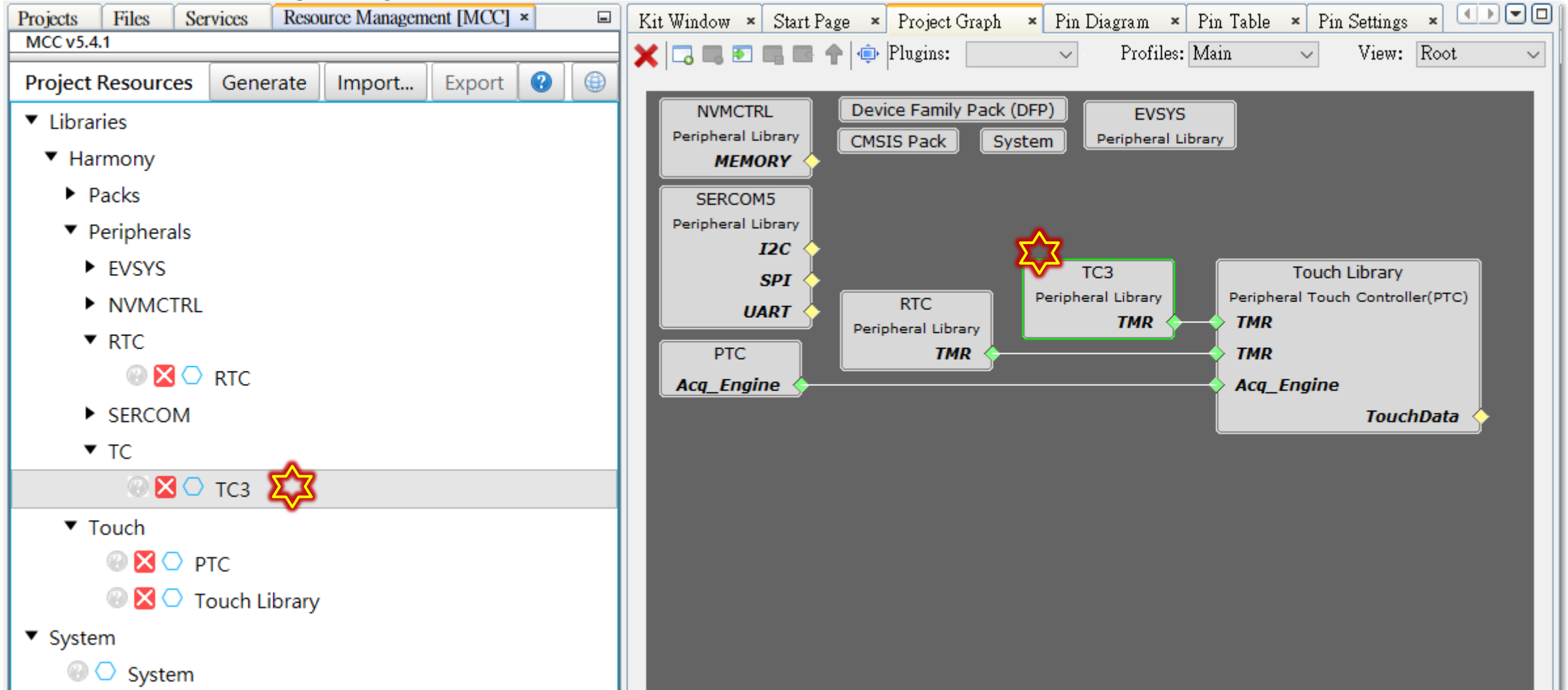
Lab1 Touch Buttons

- Configure LEDs and Buttons in Plugin **Pin Configuration** of Project Graph as below.

Kit Window × Start Page × Project Graph × Pin Diagram × Pin Table × Pin Settings ×									
Order: Pins ▼ Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
22	PA13		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
23	PA14	BT1	GPIO ▼	Digital	In ▼	Low	☐	☐	NORMAL ▼
24	PA15	BT2	GPIO ▼	Digital	In ▼	Low	☐	☐	NORMAL ▼
25	PA16		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
26	PA17	LED3	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
27	PA18		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
28	PA19		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
29	PA20	LED1	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
30	PA21	LED2	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
31	PA22		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼

⚙️ Lab1 Touch Buttons

- Check **Harmony Graph** to see the difference after added Driven Shield.



Lab1 Touch Buttons

API functions in Touch.c

uint8_t **measurement_done_touch**
touch_process ()

Touch acquisition done flag

Main processing function of touch library.

This function initiates the acquisition, calls post processing after the acquisition complete and sets the flag for next measurement based on the sensor status.

touch_init ()

Initialization PTC, timer, and datastreamer modules of touch library.

touch_timer_config ()

Initialization necessary RTC timer for Touch in touch_init()

touch_sensors_config ()

Initialization of touch key sensors in touch_init()

get_sensor_node_signal ()

Get sensor node acquisition signals.

update_sensor_node_signal ()

Update sensor node acquisition signals.

get_sensor_node_reference ()

Get sensor node channel reference.

update_sensor_node_reference ()

Update sensor node channel reference.

get_sensor_cc_val ()

Get sensor node compensation capacitor value.

update_sensor_cc_val ()

Update sensor node compensation capacitor value.

get_sensor_state ()

Get sensor node state

update_sensor_state ()

Update sensor node state

calibrate_node ()

Sensor node calibration

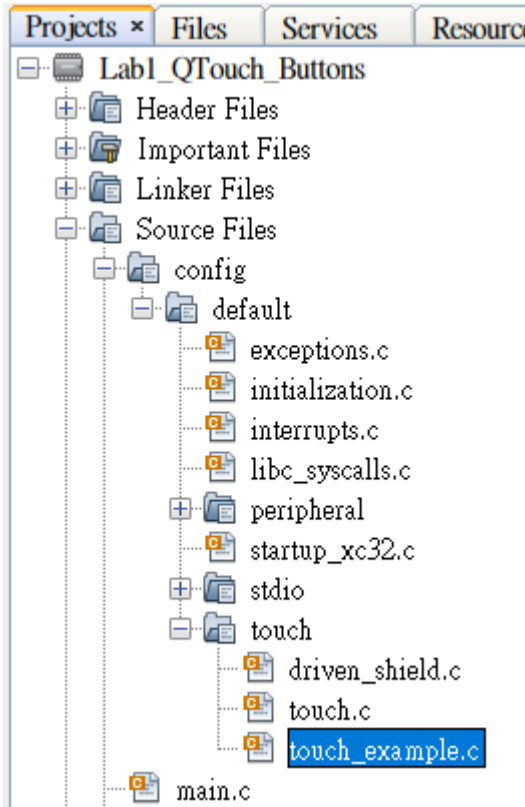
API functions in driven_shield.c

drivenshield_configure ()

Initialization Driven Shield for Touch in touch_init()

⚙️ Lab1 Touch Buttons

- Example functions in `touch_example.c` depend to sensors you to configured.



```
#include "touch_example.h"

void touch_mainloop_example(void){

    /* call touch process function */
    touch_process();

    if(measurement_done_touch == 1u)
    {
        measurement_done_touch = 0u;
        // process touch data
    }
}
```

```
/*=====
void touch_status_display(void)
-----
Purpose: Sample code snippet to demonstrate how to check the status of the
        sensors
Input   : none
Output  : none
Notes   : none
=====*/

void touch_status_display(void)
{
    uint8_t key_status = 0u;
    key_status = get_sensor_state(0) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }

    key_status = get_sensor_state(1) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }

    key_status = get_sensor_state(2) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }
}
```

Lab1 Touch Buttons

- Implement below function to output message on UART console.

```
#include <string.h>
#include <stdio.h>
#include <stdarg.h>

extern volatile uint8_t measurement_done_touch;

#define SYS_CONSOLE_PRINT_BUFFER_SIZE 200
static char consolePrintBuffer[SYS_CONSOLE_PRINT_BUFFER_SIZE];
void myprintf(const char *format, ...)
{
    size_t len = 0;
    va_list args = {0};

    va_start(args, format);
    len = vsnprintf(consolePrintBuffer, SYS_CONSOLE_PRINT_BUFFER_SIZE, format, args);
    va_end(args);

    if ((len > 0) && (len < SYS_CONSOLE_PRINT_BUFFER_SIZE))
    {
        consolePrintBuffer[len] = '\0';
        SERCOM5_USART_Write((uint8_t*)consolePrintBuffer, len);
        while (SERCOM5_USART_WriteIsBusy());
    }
}

int main ( void )
{
    SYS_Initialize ( NULL );

    myprintf("\033[2J");
    myprintf("\033[1;1H");
    myprintf("\033[?25l");
    myprintf("-----\r\n");
    myprintf("  Touch Button\r\n");
    myprintf("-----\r\n");
    myprintf("AKS  1    1    2\r\n");
    ...
}
```

Lab1 Touch Buttons

- Implement below function in **while loop** of **main.c** to make Touch Buttons works.

```
int main ( void )
{
    bool KeyStatus[3] = {false, false, false};
    ...

    while ( true )
    {
        touch_process();

        if( measurement_done_touch == 1 )
        {
            measurement_done_touch = 0;
            switch( get_sensor_state(0) )
            {
                case QTM_KEY_STATE_DETECT: KeyStatus[0]=true; LED1_Set(); break;
                case QTM_KEY_STATE_NO_DET:  KeyStatus[0]=false; LED1_Clear(); break;
            }

            switch( get_sensor_state(1) )
            {
                case QTM_KEY_STATE_DETECT: KeyStatus[1]=true; LED2_Set(); break;
                case QTM_KEY_STATE_NO_DET:  KeyStatus[1]=false; LED2_Clear(); break;
            }

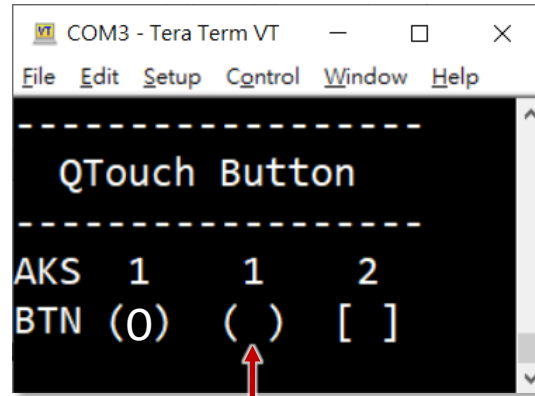
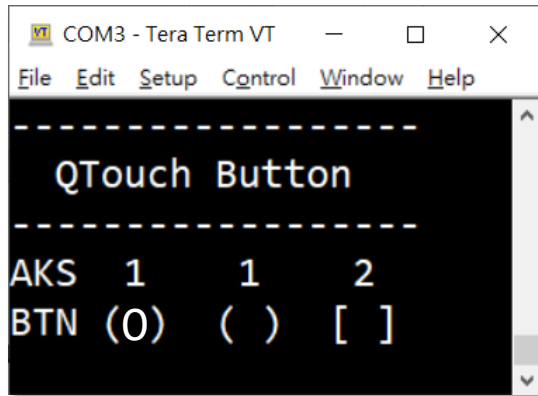
            switch( get_sensor_state(2) )
            {
                case QTM_KEY_STATE_DETECT: KeyStatus[2]=true; LED3_Set(); break;
                case QTM_KEY_STATE_NO_DET:  KeyStatus[2]=false; LED3_Clear(); break;
            }

            myprintf("\033[5;1H");
            myprintf("BTN (%c) (%c) [%c]", (KeyStatus[0]?'O':' '),
                                                            (KeyStatus[1]?'O':' '),
                                                            (KeyStatus[2]?'X':' '));

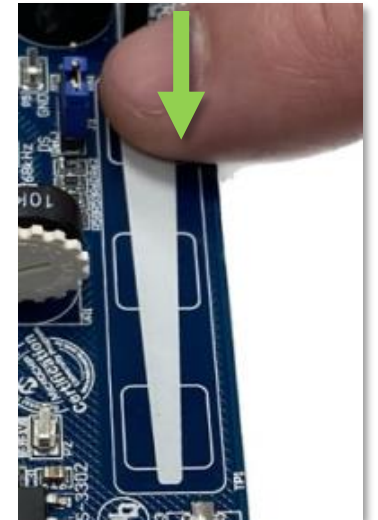
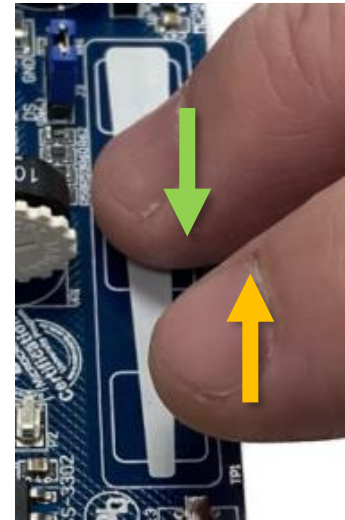
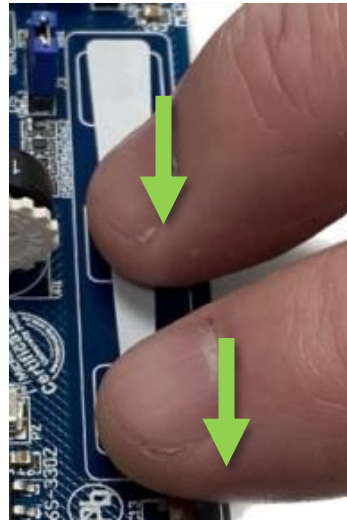
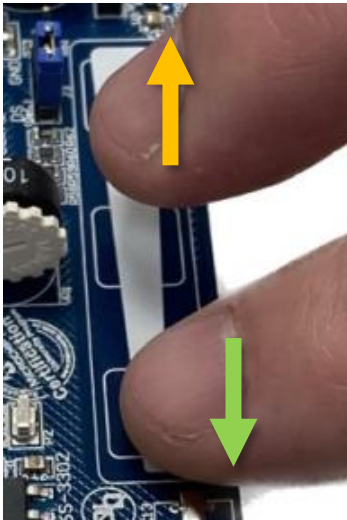
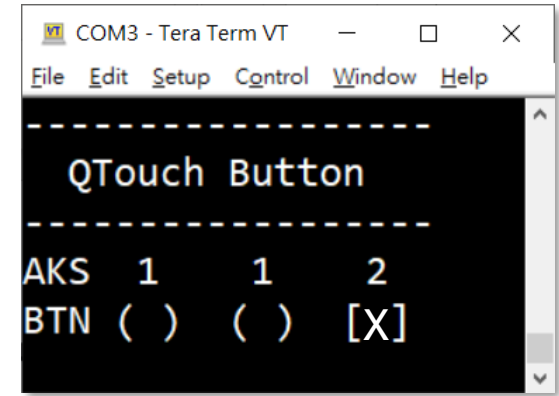
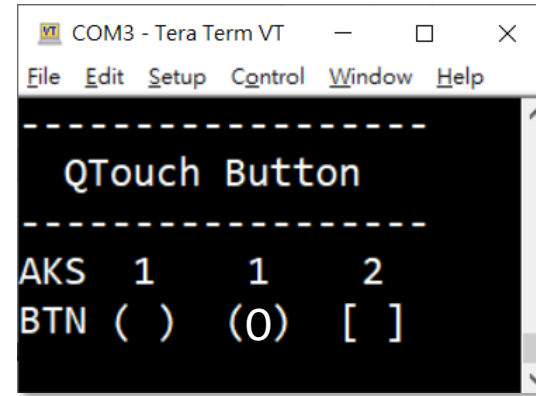
        }
        ...
    }
}
```

⚙️ Lab1 Touch Buttons

- Result of Touch Key implementation.



(Adjacent **R**ed **S**uppression)



Lab1 Touch Buttons

Discuss

- Implement manual Touch Calibration feature.

```
int main ( void )
{
    bool KeyStatus[3] = {false, false, false};
    ...

    while ( true )
    {
        touch_process();

        if(BT1_Get()==0)
        {
            while(BT1_Get()==0);
            calibrate_node(0);
            calibrate_node(1);
            calibrate_node(2);
        }

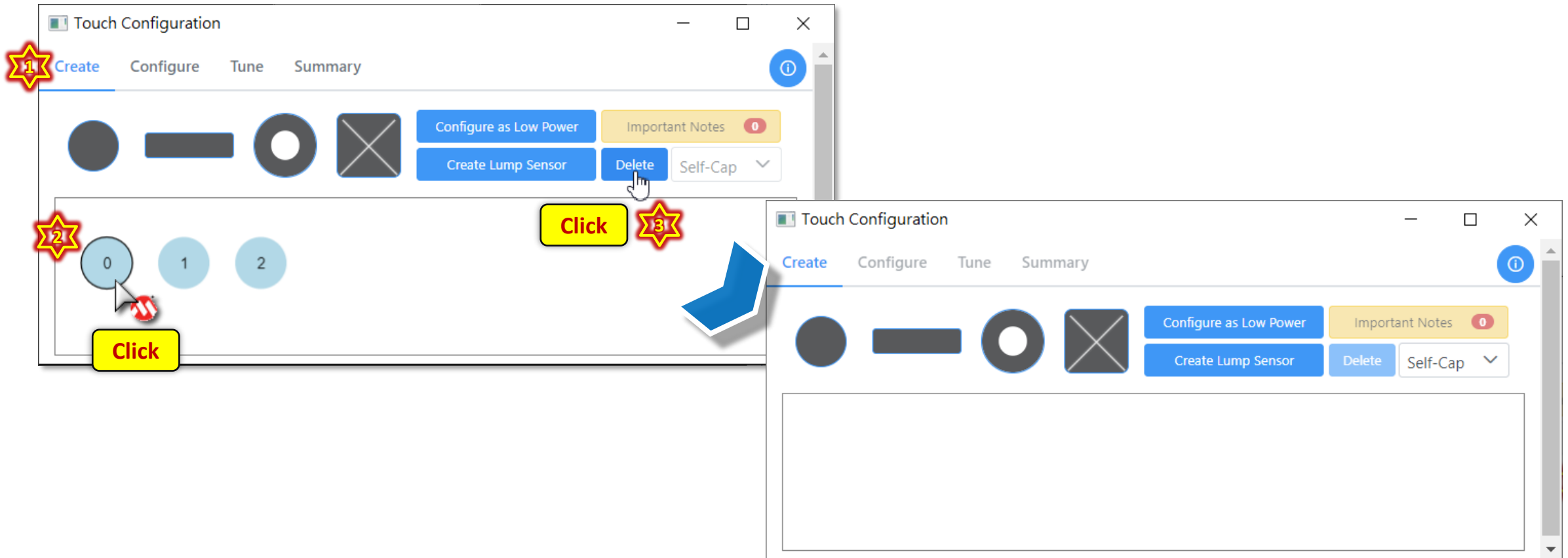
        if( measurement_done_touch == 1 )
        {
            measurement_done_touch = 0;
            switch( get_sensor_state(0) )
            {
                case QTM_KEY_STATE_DETECT:  KeyStatus[0]=true;  LED1_Set();  break;
                case QTM_KEY_STATE_NO_DET:  KeyStatus[0]=false; LED1_Clear(); break;
            }

            switch( get_sensor_state(1) )
            {
                case QTM_KEY_STATE_DETECT:  KeyStatus[1]=true;  LED2_Set();  break;
                case QTM_KEY_STATE_NO_DET:  KeyStatus[1]=false; LED2_Clear(); break;
            }
            ...
        }
    }
}
```

Lab2 Touch Slider

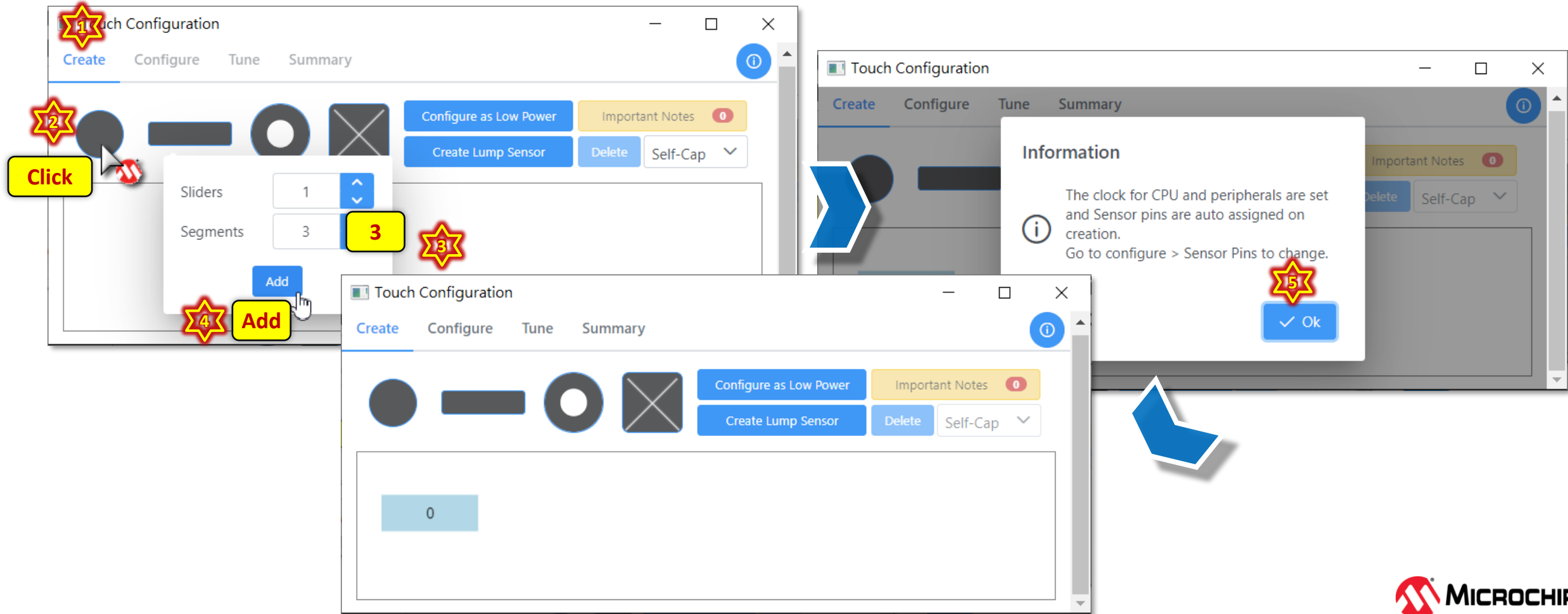
Lab2 Touch Slider

- Return Harmony Touch Configurator, **if you are continuing from previous Lab1.**
- In **Create** Tab, click on each Buttons Icon and click **[Delete]** to remove all of them.



Lab2 Touch Slider

- In **Create** Tab, click on Slider Icon and change number **Segments** to **3**.
- Click Add then a Warning dialog will show up to notice “The clock for CPU and peripherals are set and Sensor pins are auto assigned on creation”.



The image illustrates the steps to create a touch slider in the Microchip Studio IDE. It shows three sequential screenshots of the 'Touch Configuration' window, with numbered callouts (1-5) and arrows indicating the workflow.

Step 1: The 'Touch Configuration' window is open to the 'Create' tab.

Step 2: A yellow star with the number 2 is next to the slider icon. A yellow callout box with the word 'Click' and a mouse cursor points to the slider icon.

Step 3: A yellow star with the number 3 is next to the 'Segments' dropdown menu, which now shows the value '3'. A yellow callout box with the number '3' is next to the dropdown.

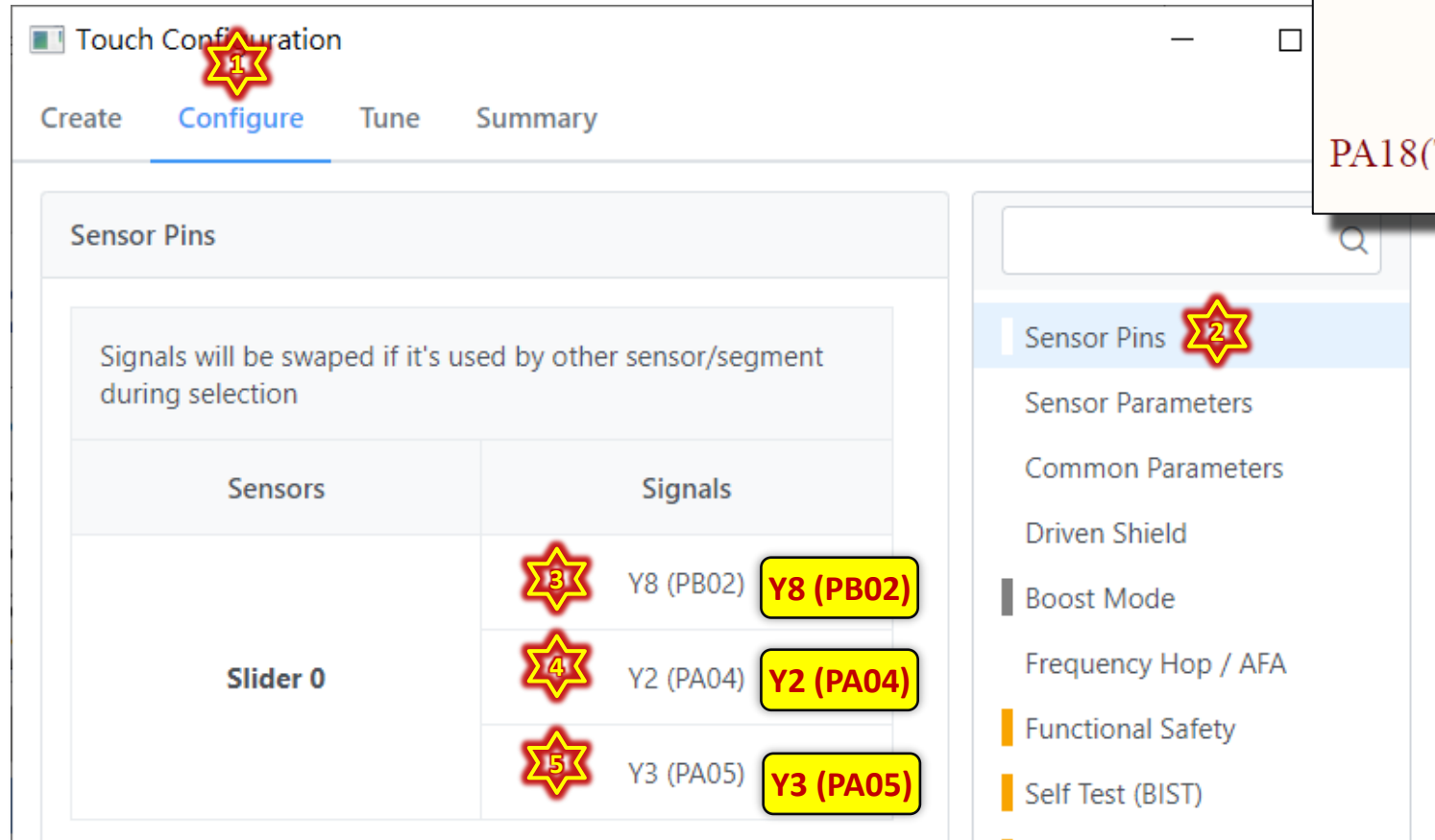
Step 4: A yellow star with the number 4 is next to the 'Add' button. A yellow callout box with the word 'Add' and a mouse cursor points to the button.

Step 5: A yellow star with the number 5 is next to the 'Ok' button in an 'Information' dialog box. The dialog box contains the text: 'The clock for CPU and peripherals are set and Sensor pins are auto assigned on creation. Go to configure > Sensor Pins to change.'

✂ Lab2 Touch Slider

- In **Configure** Tab, Click on **Sensor Pins** and configure Slider 0 Signals pins as below.

- Slider 0 (Signals) : Y8 (PB02)
- Slider 0 (Signals) : Y2 (PA04)
- Slider 0 (Signals) : Y3 (PA05)



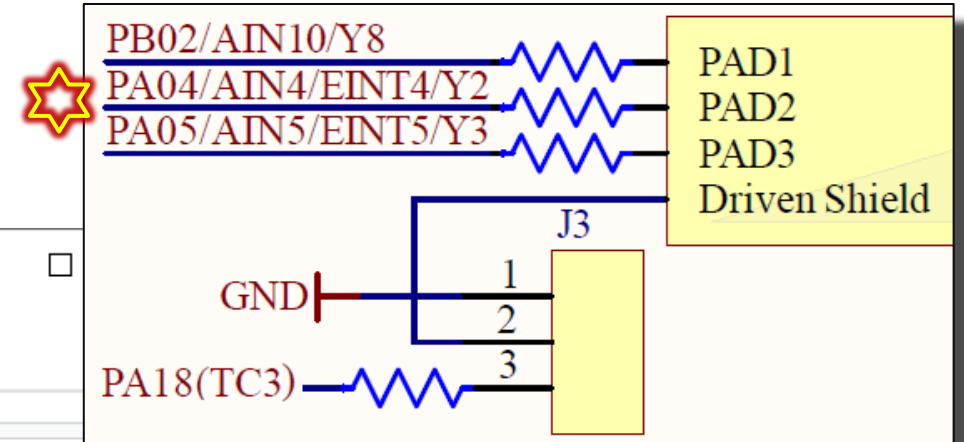
Touch Configuration

Create **Configure** Tune Summary

Sensor Pins

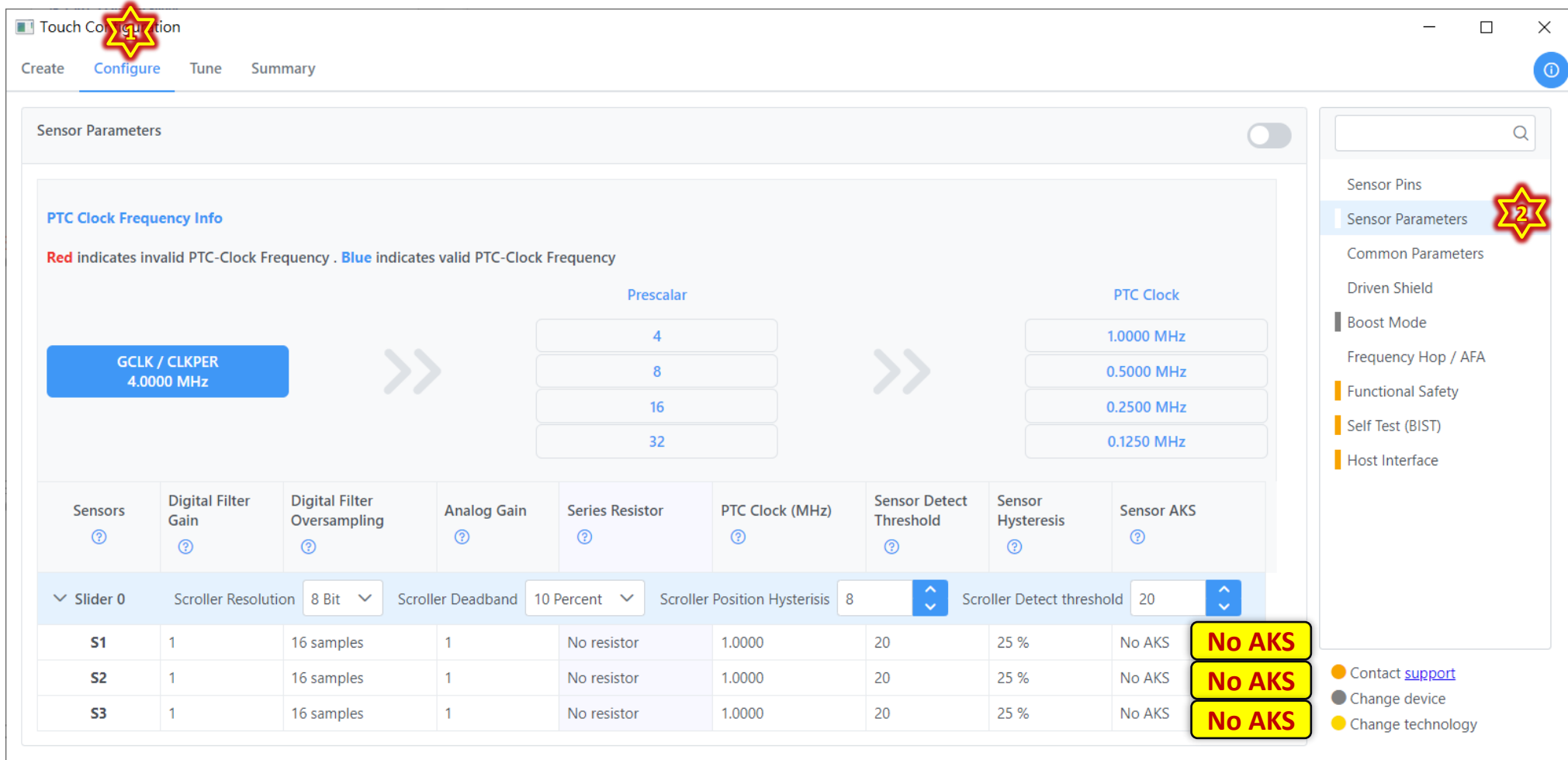
Signals will be swapped if it's used by other sensor/segment during selection

Sensors	Signals
Slider 0	Y8 (PB02)
	Y2 (PA04)
	Y3 (PA05)



Lab2 Touch Slider

- In **Configure** Tab, Click on **Sensor Parameters** and check the configuration of **Sensor AKS**(Adjacent **K**ey **S**uppression) group as below.
- S1 : no AKS, S2 : no AKS, S3 : no AKS



Touch Configuration

Create **Configure** Tune Summary

Sensor Parameters

PTC Clock Frequency Info

Red indicates invalid PTC-Clock Frequency . Blue indicates valid PTC-Clock Frequency

Prescalar

PTC Clock

GCLK / CLKPER
4.0000 MHz

4
8
16
32

1.0000 MHz
0.5000 MHz
0.2500 MHz
0.1250 MHz

Sensors	Digital Filter Gain	Digital Filter Oversampling	Analog Gain	Series Resistor	PTC Clock (MHz)	Sensor Detect Threshold	Sensor Hysteresis	Sensor AKS
S1	1	16 samples	1	No resistor	1.0000	20	25 %	No AKS
S2	1	16 samples	1	No resistor	1.0000	20	25 %	No AKS
S3	1	16 samples	1	No resistor	1.0000	20	25 %	No AKS

Slider 0 Scroller Resolution 8 Bit Scroller Deadband 10 Percent Scroller Position Hysteresis 8 Scroller Detect threshold 20

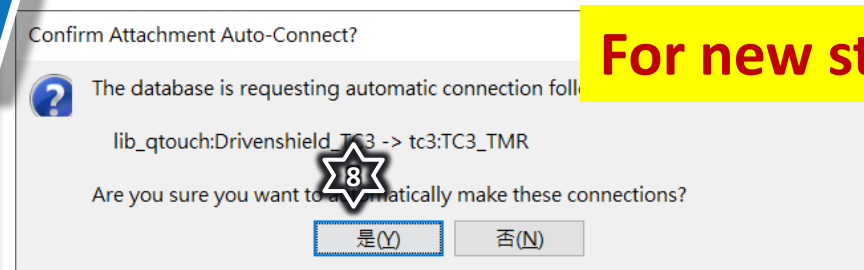
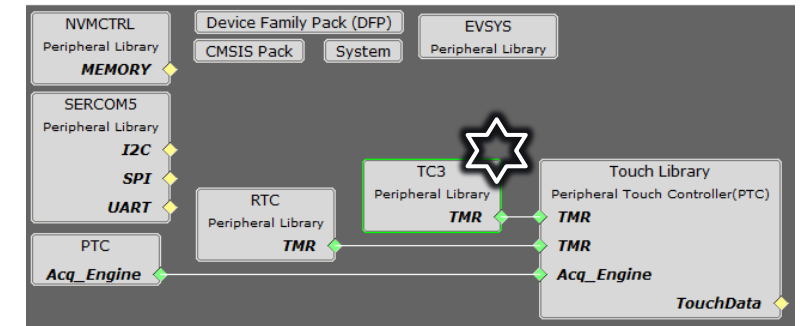
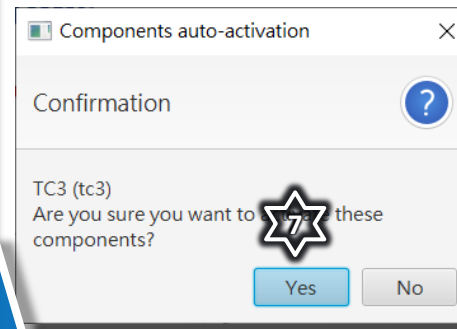
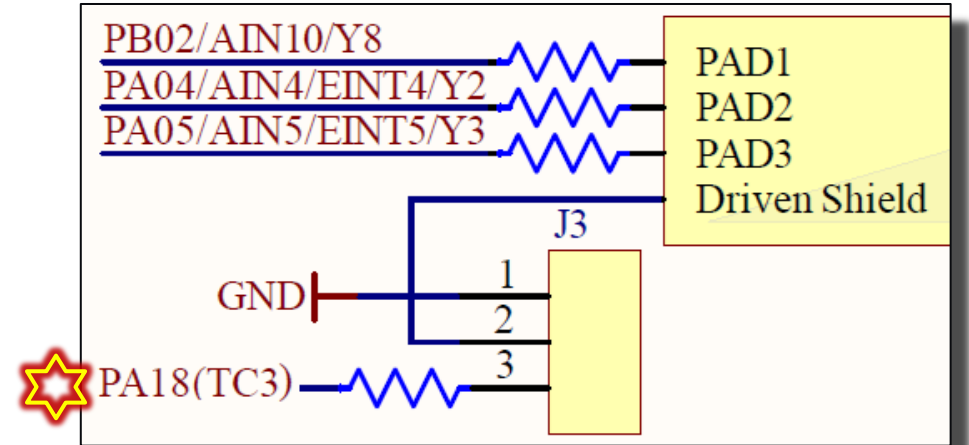
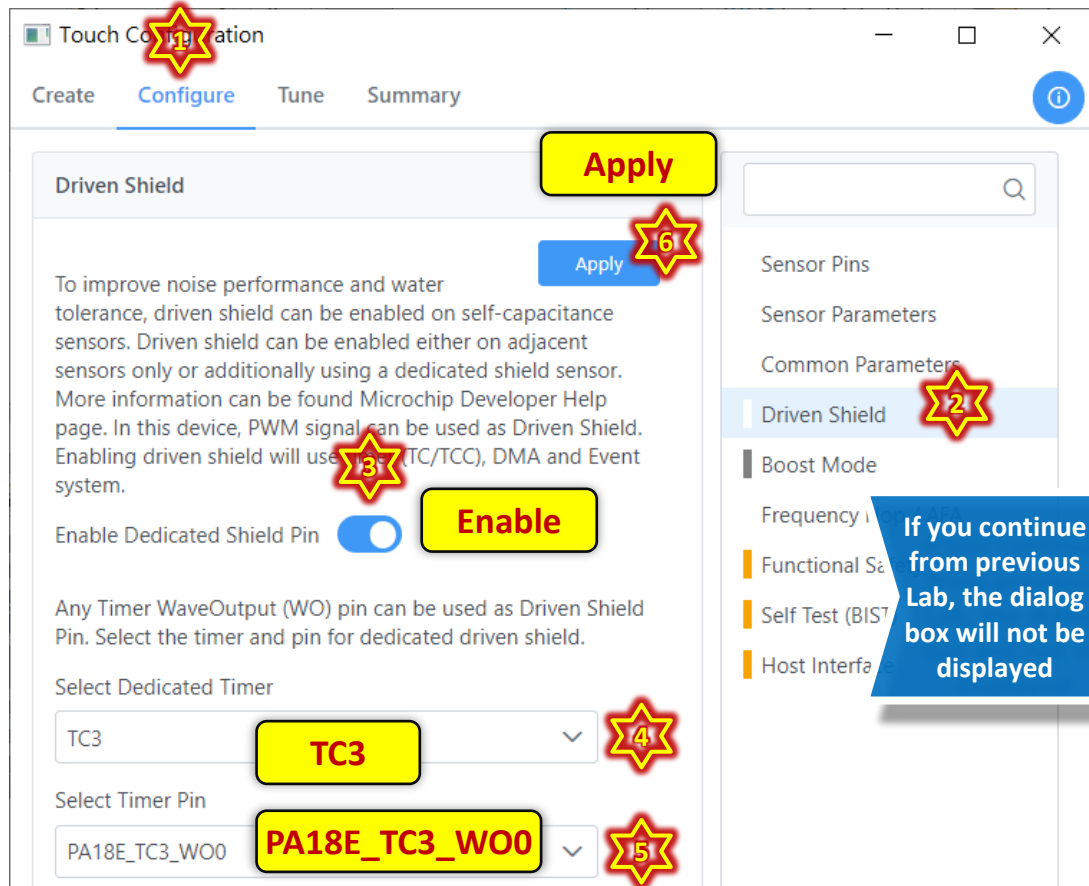
No AKS
No AKS
No AKS

Contact [support](#)
Change device
Change technology

✖ Lab2 Touch Slider

- In **Configure** Tab, Click on **Driven Shield** and configure Driven Shield as below.

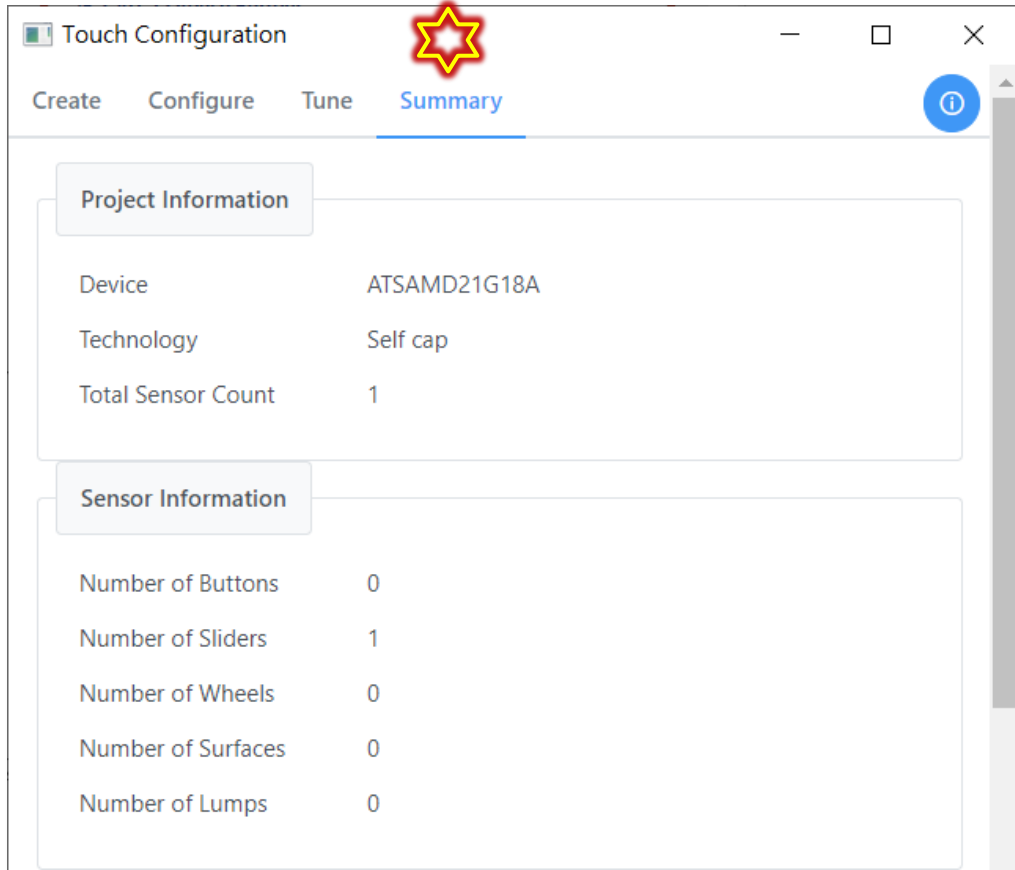
- Enable Dedicated Driven Shield : 
- Select Dedicated Timer : TC3
- Select Timer Pin : PA18E_TC3_WO0



For new start project only

Lab2 Touch Slider

- In **Summary** Tab, to figure out current configuration of Touch.



Touch Configuration

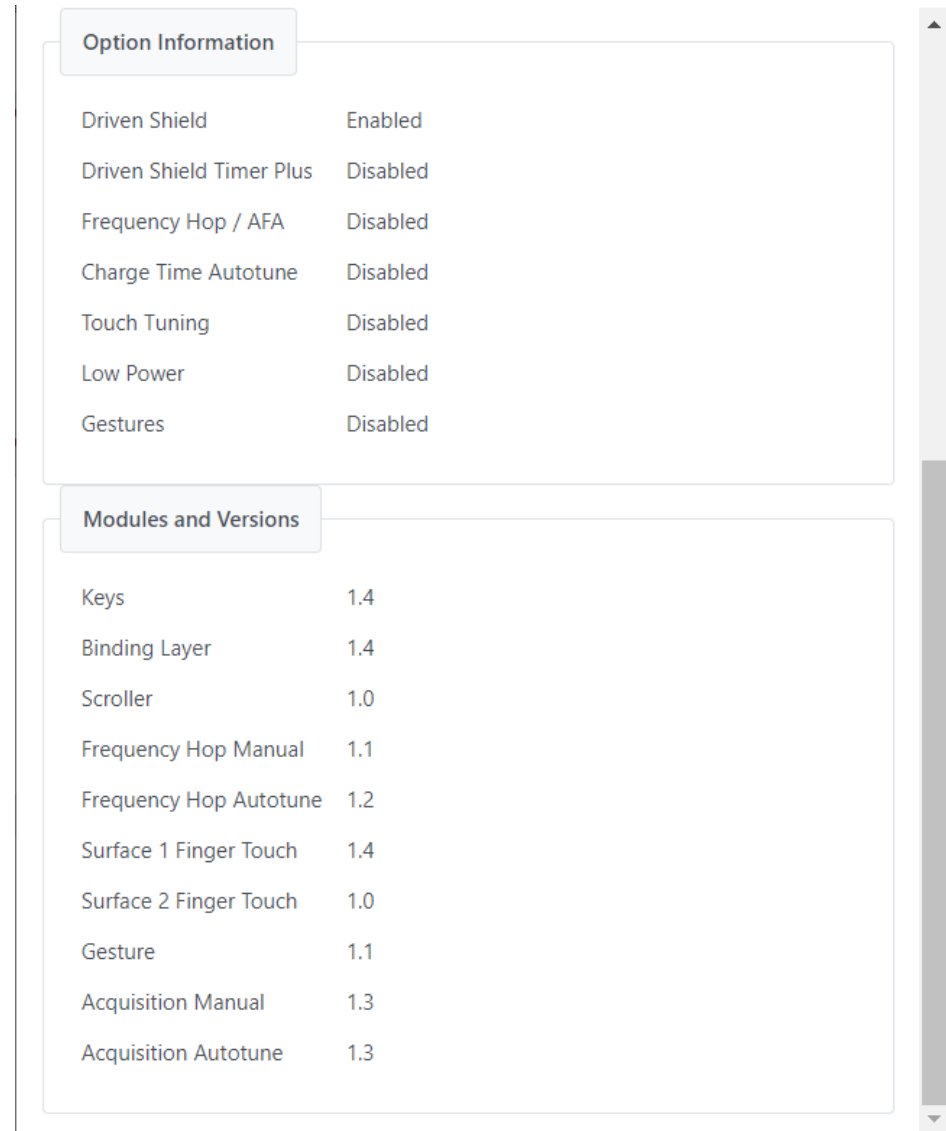
Create Configure Tune **Summary**

Project Information

Device	ATSAMD21G18A
Technology	Self cap
Total Sensor Count	1

Sensor Information

Number of Buttons	0
Number of Sliders	1
Number of Wheels	0
Number of Surfaces	0
Number of Lumps	0



Option Information

Driven Shield	Enabled
Driven Shield Timer Plus	Disabled
Frequency Hop / AFA	Disabled
Charge Time Autotune	Disabled
Touch Tuning	Disabled
Low Power	Disabled
Gestures	Disabled

Modules and Versions

Keys	1.4
Binding Layer	1.4
Scroller	1.0
Frequency Hop Manual	1.1
Frequency Hop Autotune	1.2
Surface 1 Finger Touch	1.4
Surface 2 Finger Touch	1.0
Gesture	1.1
Acquisition Manual	1.3
Acquisition Autotune	1.3

Lab2 Touch Slider

- Check the Touch Sensor Pins in Plugin **Pin Configuration** of Project Graph as below.

Kit Window × Start Page × Project Graph × Pin Diagram × Pin Table × Pin Settings ×									
Order: Pins ▼ Table View ☒ Easy View									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
9	PA04		PTC_Y2	Analog	High Impedance	n/a	☐	☐	NORMAL
10	PA05		PTC_Y3	Analog	High Impedance	n/a	☐	☐	NORMAL
47	PB02		PTC_Y8	Analog	High Impedance	n/a	☐	☐	NORMAL

Lab2 Touch Slider

- Configure LEDs and Buttons in Plugin **Pin Configuration** of Project Graph as below.

Kit Window × Start Page × Project Graph × Pin Diagram × Pin Table × Pin Settings ×									
Order: Pins ▼ Table View ☒ Easy View									
For new start project only									
Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
22	PA13		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
23	PA14	BT1	GPIO ▼	Digital	In ▼	Low	☐	☐	NORMAL ▼
24	PA15	BT2	GPIO ▼	Digital	In ▼	Low	☐	☐	NORMAL ▼
25	PA16		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
26	PA17	LED3	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
27	PA18		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
28	PA19		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼
29	PA20	LED1	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
30	PA21	LED2	GPIO ▼	Digital	Out ▼	Low	☐	☐	NORMAL ▼
31	PA22		Available ▼	Digital	High Impedance ▼	Low	☐	☐	NORMAL ▼

✖ Lab2 Touch Slider

- Check **Harmony Graph** to see the difference after added Driven Shield.

The screenshot displays the Microchip Harmony v5.4.1 Resource Management (MCC) interface. The left pane shows the 'Project Resources' tree with the following structure:

- Libraries
 - Harmony
 - Packs
 - Peripherals
 - EVSYS
 - NVMCTRL
 - RTC
 - RTC (disabled)
 - SERCOM
 - TC
 - TC3 (selected and highlighted with a yellow star)
 - Touch
 - PTC (disabled)
 - Touch Library (disabled)
 - System
 - System (disabled)

The right pane shows the 'Project Graph' window, which visualizes the hardware components and their interconnections. The graph includes the following components and connections:

- NVMCTRL** (Peripheral Library) connected to **MEMORY**.
- SERCOM5** (Peripheral Library) connected to **I2C**, **SPI**, and **UART**.
- PTC** (Peripheral Library) connected to **Acq_Engine**.
- RTC** (Peripheral Library) connected to **TMR**.
- TC3** (Peripheral Library) connected to **TMR** (highlighted with a yellow star).
- Touch Library** (Peripheral Touch Controller (PTC)) connected to **TMR** and **Acq_Engine**.
- TouchData** (Peripheral Library) connected to **Acq_Engine**.

A yellow star highlights the **TC3** component in the Project Graph, indicating it is the selected component.

For new start project only

Lab2 Touch Slider

API functions in Touch.c

uint8_t measurement_done_touch
touch_process ()

touch_init ()

touch_timer_config ()

touch_sensors_config ()

get_sensor_node_signal ()

update_sensor_node_signal ()

get_sensor_node_reference ()

update_sensor_node_reference ()

get_sensor_cc_val ()

update_sensor_cc_val ()

get_sensor_state ()

update_sensor_state ()

calibrate_node ()

get_scroller_state

get_scroller_position

Touch acquisition done flag

Main processing function of touch library.

This function initiates the acquisition, calls post processing after the acquisition complete and sets the flag for next measurement based on the sensor status.

Initialization PTC, timer, and datastreamer modules of touch library.

Initialization necessary RTC timer for Touch in touch_init()

Initialization of touch key sensors in touch_init()

Get sensor node acquisition signals.

Update sensor node acquisition signals.

Get sensor node channel reference.

Update sensor node channel reference.

Get sensor node compensation capacitor value.

Update sensor node compensation capacitor value.

Get sensor node state

Update sensor node state

Sensor node calibration

Get scroller sensor state

Get scroller position

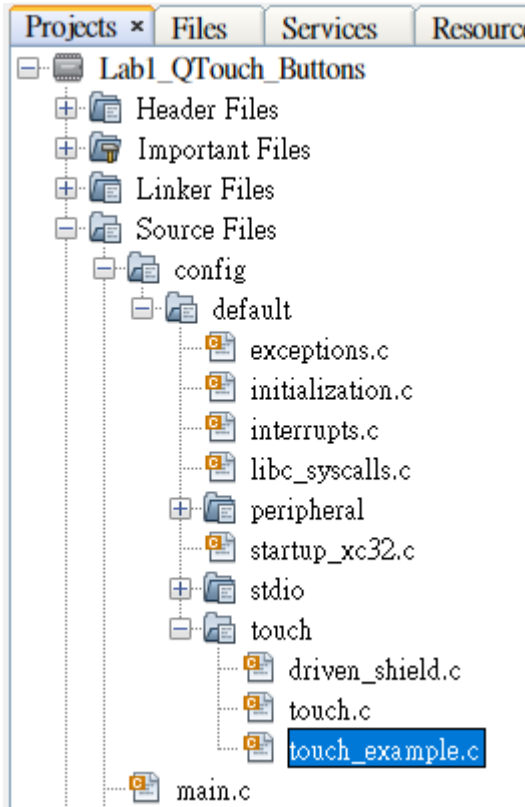
API functions in driven_shield.c

drivenshield_configure ()

Initialization Driven Shield for Touch in touch_init()

⚙️ Lab2 Touch Slider

- Example functions in `touch_example.c` depend to sensors you to configured.



```
#include "touch_example.h"

void touch_mainloop_example(void){

    /* call touch process function */
    touch_process();

    if(measurement_done_touch == 1u)
    {
        measurement_done_touch = 0u;
        // process touch data
    }
}
```

```
scroller_status = get_scroller_state(0);
scroller_position = get_scroller_position(0);
//Example: 8 bit scroller resolution. Modify as
scroller_position = scroller_position >> 5u;
//LED_OFF
if ( 0u != scroller_status) {
    switch (scroller_position) {
        case 0:
            //LED0_ON
            break;
        case 1:
            //LED1_ON
            break;
        ...

        case 7:
            //LED7_ON
            break;
        default:
            //LED_OFF
            break;
    }
}
```

```
/*=====
void touch_status_display(void)
-----
Purpose: Sample code snippet to demonstrate how to check the status
of the
        sensors
Input   : none
Output  : none
Notes   : none
=====*/
void touch_status_display(void)
{
    uint8_t key_status = 0u;
    uint8_t scroller_status = 0u;
    uint16_t scroller_position = 0u;
    key_status = get_sensor_state(0) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }

    key_status = get_sensor_state(1) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }

    key_status = get_sensor_state(2) & KEY_TOUCHED_MASK;
    if (0u != key_status) {
        //Touch detect
    } else {
        //Touch No detect
    }
}
```

Lab2 Touch Slider

- Implement below function in `main.c` to make sure your UART console is ready for use.

```
#include <string.h>
#include <stdio.h>
#include <stdarg.h>

extern volatile uint8_t measurement_done_touch;

#define SYS_CONSOLE_PRINT_BUFFER_SIZE 200
static char consolePrintBuffer[SYS_CONSOLE_PRINT_BUFFER_SIZE];
void myprintf(const char *format, ...)
{
    size_t len = 0;
    va_list args = {0};

    va_start(args, format);
    len = vsnprintf(consolePrintBuffer, SYS_CONSOLE_PRINT_BUFFER_SIZE, format, args);
    va_end(args);

    if ((len > 0) && (len < SYS_CONSOLE_PRINT_BUFFER_SIZE))
    {
        consolePrintBuffer[len] = '\0';
        SERCOM5_USART_Write((uint8_t*)consolePrintBuffer, len);
        while (SERCOM5_USART_WriteIsBusy());
    }
}

int main ( void )
{
    SYS_Initialize ( NULL );

    myprintf("\033[2J");
    myprintf("\033[1;1H");
    myprintf("\033[?25l");
    myprintf("-----\r\n");
    myprintf("  Touch Slider\r\n");
    myprintf("-----\r\n");
    myprintf("_____");
    ...
}
```

Lab2 Touch Slider

- Implement below function in **while loop** of **main.c** to make Touch Slider works.

```
int main ( void )
{
    uint16_t ScrollPos, PreScrollPos=256;

    SYS_Initialize ( NULL );
    ...

    while ( true )
    {
        touch_process();

        if( measurement_done_touch==1 && get_scroller_state(0) )
        {
            measurement_done_touch = 0;
            ScrollPos = get_scroller_position(0);
            if( ScrollPos != PreScrollPos )
            {
                PreScrollPos = ScrollPos;
                myprintf("\033[4;1H");
                myprintf("_____");
                myprintf("\033[4;%dH", (ScrollPos*17/255)+1);
                myprintf("0");
                myprintf("\033[5;1H");
                myprintf("Slider Pos = %03d", ScrollPos);

                switch( ScrollPos*4/255 )
                {
                    case 0: LED1_Set(); LED2_Clear(); LED3_Clear(); break;
                    case 1: LED1_Set(); LED2_Set(); LED3_Clear(); break;
                    case 2: LED1_Clear(); LED2_Set(); LED3_Clear(); break;
                    case 3: LED1_Clear(); LED2_Set(); LED3_Set(); break;
                    case 4: LED1_Clear(); LED2_Clear(); LED3_Set(); break;
                }
            }
        }
        ...
    }
}
```

Lab2 Touch Slider

- Implement manual Touch Calibration feature.

```
int main ( void )
{
    uint16_t ScrollPos, PreScrollPos=256;
    ...

    while ( true )
    {
        touch_process();

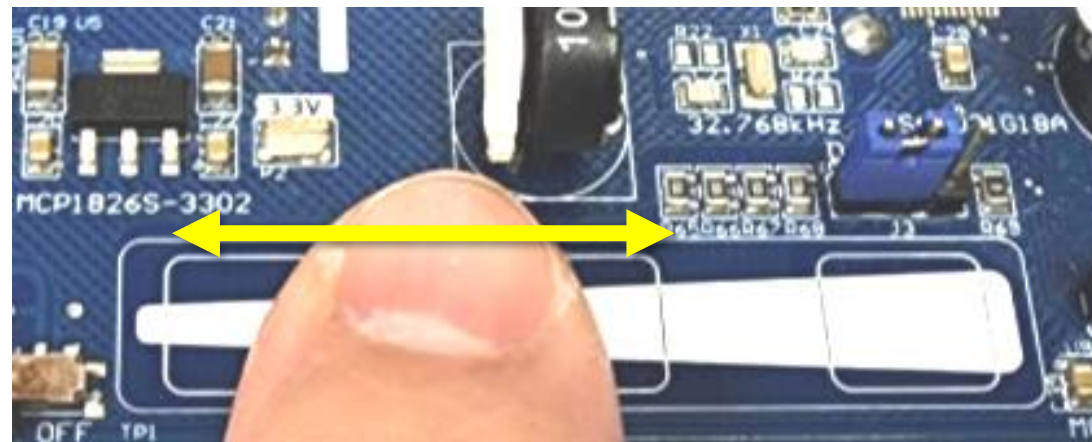
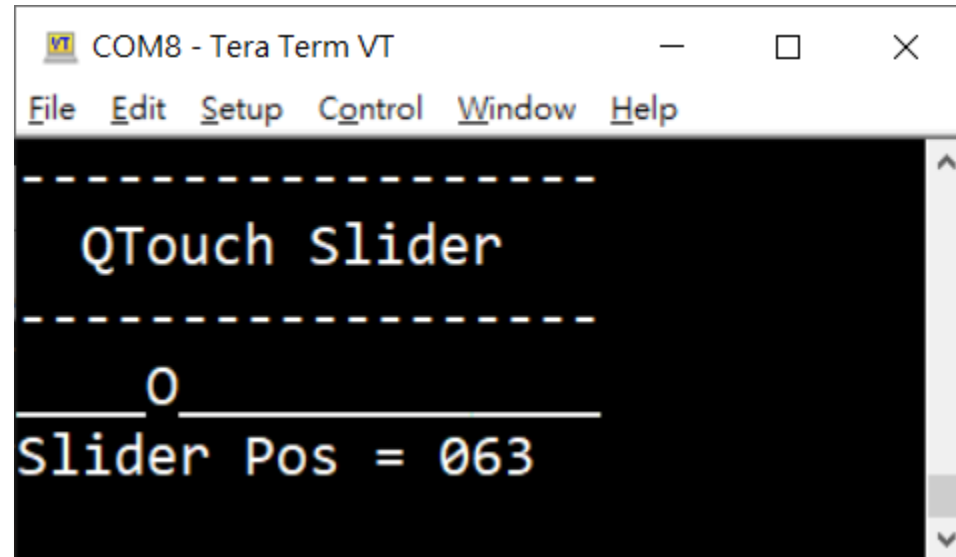
        if(BT1_Get()==0)
        {
            while(BT1_Get()==0);
            calibrate_node(0);
            calibrate_node(1);
            calibrate_node(2);
        }

        if( measurement_done_touch==1 && (get_scroller_state(0) & KEY_TOUCHED_MASK) )
        {
            measurement_done_touch = 0;
            ScrollPos = get_scroller_position(0);
            if( ScrollPos != PreScrollPos )
            {
                PreScrollPos = ScrollPos;
                myprintf("\033[4;1H");
                myprintf("_____");
                myprintf("\033[4;%dH", (ScrollPos*17/255)+1);
                myprintf("0");
                myprintf("\033[5;1H");
                myprintf("Slider Pos = %03d", ScrollPos);

                switch( ScrollPos*4/255 )
                {
                    case 0: LED1_Set(); LED2_Clear(); LED3_Clear(); break;
                    case 1: LED1_Set(); LED2_Set(); LED3_Clear(); break;
                    case 2: LED1_Clear(); LED2_Set(); LED3_Clear(); break;
                    ...
                }
            }
        }
    }
}
```

✖ Lab2 Touch Slider

- Result of Touch Slider(Scroller) implementation.



Lab3 Data Visualizer for Touch

Lab3 Data Visualizer for Touch

- **MPLAB® Data Visualizer** could support to Touch Visualization and Tuning through UART.

The screenshot displays the MPLAB X IDE environment with the Lab3_QTouch_DataVisualizer project open. The interface is organized into several key sections:

- Menu Bar:** File, Edit, MHC, View, Navigate, Source, Refactor, Production, Debug, Team, Tools, Window, Help.
- Toolbar:** Includes icons for file operations (New, Open, Save, etc.) and a search bar.
- Left Sidebar:** Contains project components like 'Lab3_QTouch_DataVisualizer', 'COM3 on COM3', and various widgets like 'FrameCounter', 'Signal0', 'Reference0', and 'Delta0'.
- Top Panel:** Shows the 'Time Plot' and 'Dashboard' tabs. The 'Time Plot' tab is active, displaying a graph of sensor data over time.
- Central Plot Area:** A time plot showing multiple channels of data (Signal0, Reference0, Delta0, etc.) over a time range from 7072s to 7090s. A legend at the bottom identifies the channels.
- Right Sidebar:** Contains data tables for 'Button Data', 'Sensor Data', 'Slider & Wheel Data', and 'Debug Data'. It also includes a 'Compensation' section with a text box for the compensation value.

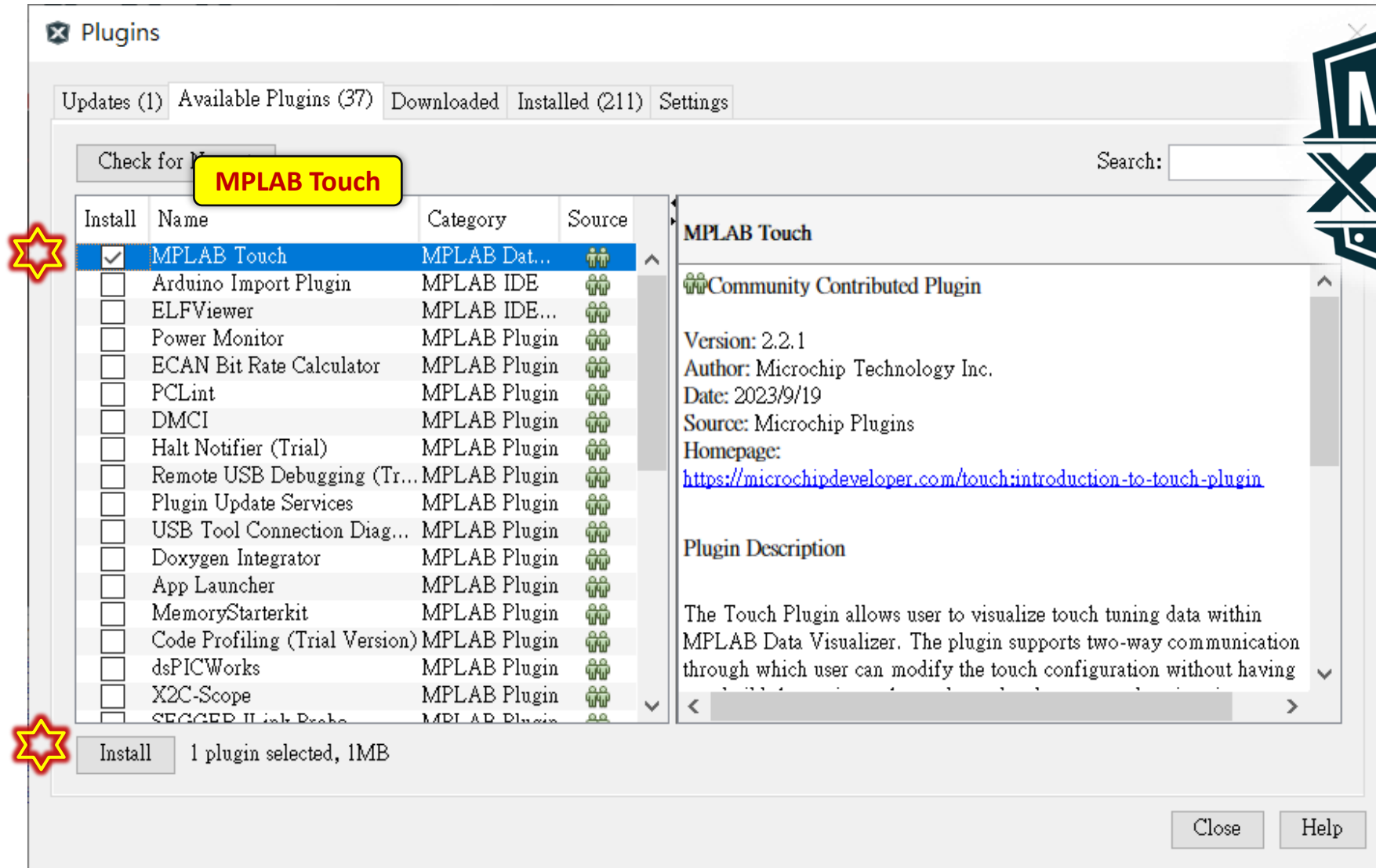
The 'Time Plot' tab shows a graph of sensor data over time. The x-axis represents time in seconds (7072s to 7090s), and the y-axis represents data values (0 to 200). The legend at the bottom identifies the channels: Signal0 on Lab3_QTouch_DataVisualizer, Reference0 on Lab3_QTouch_DataVisualizer, Delta0 on Lab3_QTouch_DataVisualizer, Delta1 on Lab3_QTouch_DataVisualizer, Delta2 on Lab3_QTouch_DataVisualizer, Threshold0 on Lab3_QTouch_DataVisualizer, Threshold1 on Lab3_QTouch_DataVisualizer, Threshold2 on Lab3_QTouch_DataVisualizer, SWDelta0 on Lab3_QTouch_DataVisualizer, and SWThreshold0 on Lab3_QTouch_DataVisualizer.

The 'Dashboard' tab shows a summary of the sensor data, including a table of 'Button Data' and a table of 'Sensor Data'. The 'Button Data' table has columns for Channel ID, Sensor Type, State, Delta, and Threshold. The 'Sensor Data' table has columns for Channel ID, Sensor Type, Signal, Reference, Delta, and Compensation pF.

The 'Compensation' section includes a text box for the compensation value, which is currently set to 0. The text below the box states: 'Compensation: Represents PTC compensation circuit value which is equivalent to sensor capacitance. "Compensation" value can be used to check whether sensor is saturated. Refer to User Guide'.

⚙️ Lab3 Data Visualizer for Touch

- Before use **Data Visualizer** for touch tuning, it's need install extra Touch plugin firstly.



⚙️ Lab3 Data Visualizer for Touch

- Return Harmony Touch Configurator, **if you are continuing from previous Lab2.** In **Tune** Tab, Please switch to **Enable Touch Tuning Data connection** and choice **Touch Visualization (unidirectional, requires UART Tx)**

The screenshot shows the 'Touch Configuration' window with the 'Tune' tab selected. The window has a title bar with standard OS controls and a blue information icon. Below the title bar are tabs for 'Create', 'Configure', 'Tune' (active), and 'Summary'. The main content area contains text explaining 'Touch Visualization' and its requirements. It includes a toggle switch for 'Enable Touch Tuning Data connection' (labeled with a yellow star '2') and three radio button options for visualization (labeled with a yellow star '3'). The first option is 'Touch Tuning (bidirectional, requires UART Tx and Rx)'. The second option, 'Touch Visualization (unidirectional, requires UART Tx)', is selected and highlighted with a yellow box. The third option is 'Touch Tuning - 2D Surface (bidirectional, requires UART Tx and Rx)'. At the bottom, there is a section for 'Charge Time Autotune' with a toggle switch.

1 Touch Configuration

Create Configure **Tune** Summary

Touch Visualization enables to visualize relevant touch data on screen in real time. The communication is established using the UART capabilities of the touch MCU requiring **only TX** connected from Touch MCU to UART-USB bridge (such as [Microchip Touch Bridge](#) or [MCP2221A breakout module](#)). Use MPLAB Data Visualizer with [Auto-Configure](#) in Variable Streamers tab or [Atmel Data visualizer](#). MPLAB Data Visualizer exists within MPLAB X IDE as well as a standalone application.

This should be enabled and used for project development and validation and be disabled for production code to remove overhead, power consumption and flash/RAM footprint..

Enable Touch Tuning Data connection **2** **Enable Touch Tuning Data connection**

3 ☐ Touch Tuning (bidirectional, requires UART Tx and Rx)

☒ **Touch Visualization (unidirectional, requires UART Tx)** **Touch Visualization (unidirectional, requires UART Tx)**

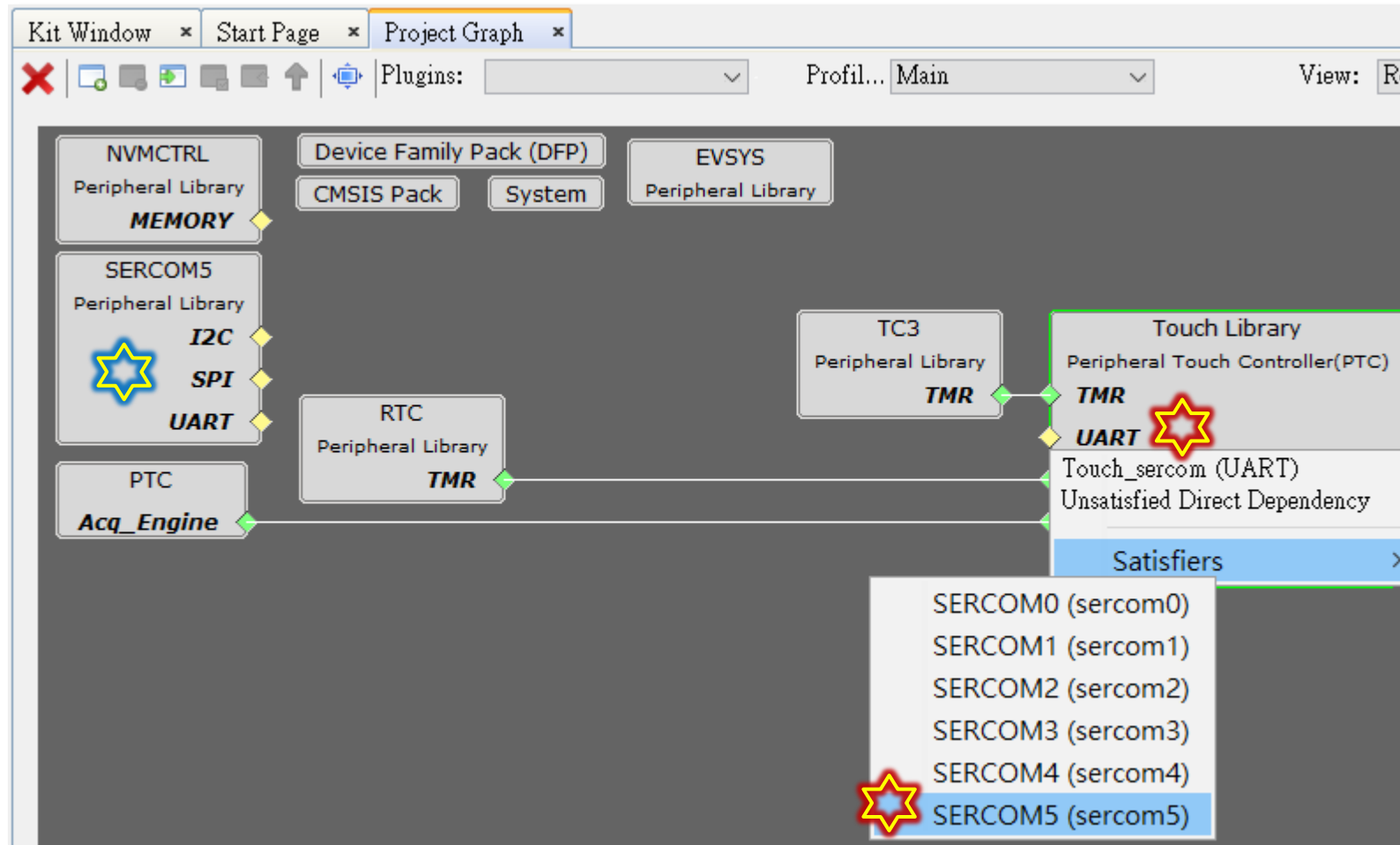
☐ Touch Tuning - 2D Surface (bidirectional, requires UART Tx and Rx)

Charge Time Autotune feature aids the tuning of sensor charge time. Manual tuning process is explained in [Microchip Developer Help page](#)
!!! Users must enable this feature ONLY during development and configure the recommended settings in touch.h file and disable this option in production application.

Enable Charge Time Autotune

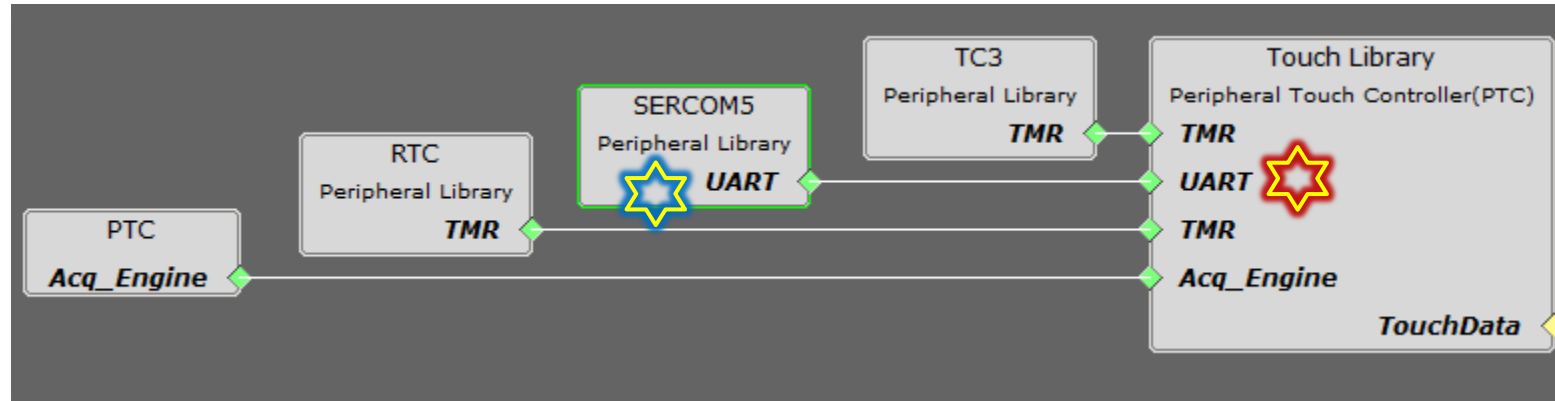
⚙️ Lab3 Data Visualizer for Touch

- A new **UART** connection interface will appear on **Touch Library** module after enabled the Tuning feature in Touch Configurator.
- Assign PLIB **SERCOM5** to **UART** interface of **Touch Library**.



⚙️ Lab3 Data Visualizer for Touch

- Configure PLIB **SERCOM5-UART** as below, **if you are not continuing from previous Lab2.**



For new start project only

Pin Number	Pin ID	Custom Name	Function
37	PB22		SERCOM5_PAD2
38	PB23		SERCOM5_PAD3

SERCOM5

Select SERCOM Operation Mode: USART with internal Clock

Operating Mode: Blocking mode **Use Blocking Mode**

Receive Enable: ☒

Transmit Enable: ☒

Frame Format: USART frame

Baud Rate in Hz: 115,200

Parity Mode: No Parity

Character Size: 8 Bits

Stop Bit Mode: One Stop Bit

Receive Pinout: SERCOM PAD[3] is used for data reception

Transmit Pinout: PAD[2] = TxD; PAD[3] = XCK

Enable Run in Standby: ☐

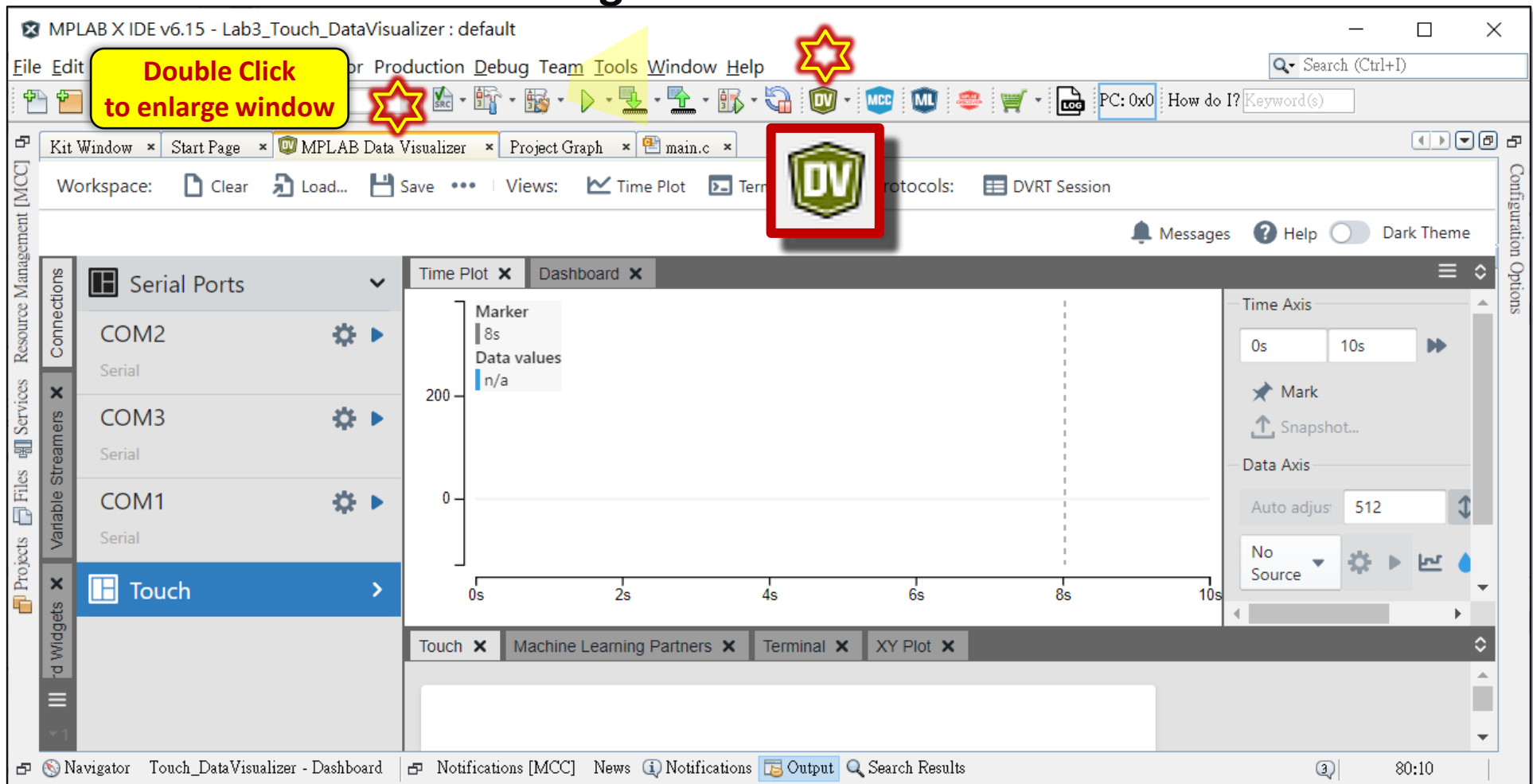
⚙️ Lab3 Data Visualizer for Touch

- Modify `myprintf()` to use `TransmitComplete()` function of SERCOM Blocking Mode.
- **Common out** all debug `myprintf()` in while loop of main.c.

```
...  
void myprintf(const char *format, ...)  
{  
    ...  
    if ((len > 0) && (len < SYS_CONSOLE_PRINT_BUFFER_SIZE))  
    {  
        consolePrintBuffer[len] = '\0';  
        SERCOM5_USART_Write((uint8_t*)consolePrintBuffer, len);  
        //while (SERCOM5_USART_WriteIsBusy());  
        while( !SERCOM5_USART_TransmitComplete() );  
    }  
}  
  
int main ( void )  
{  
    ...  
  
    while ( true )  
    {  
        touch_process();  
  
        if( measurement_done_touch==1 && get_scroller_state(0) )  
        {  
            ...  
            if( ScrollPos != PreScrollPos )  
            {  
                PreScrollPos = ScrollPos;  
                //myprintf("\033[4;1H");  
                //myprintf(" ");  
                //myprintf("\033[4;%dH", (ScrollPos*17/255)+1);  
                //myprintf("0");  
                //myprintf("\033[5;1H");  
                //myprintf("Slider Pos = %03d", ScrollPos);  
  
                switch( ScrollPos*4/255 )
```

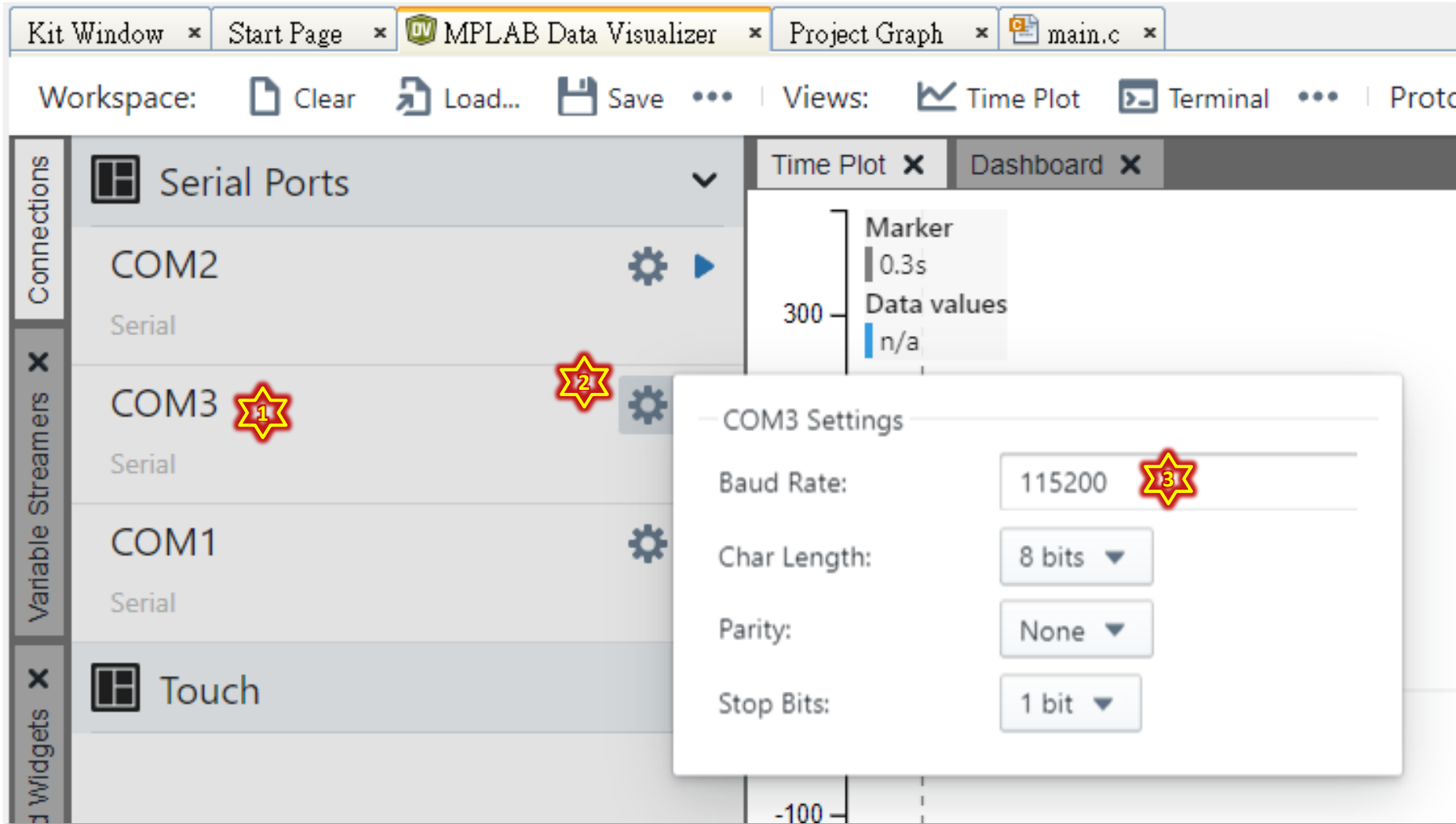
⚙️ Lab3 Data Visualizer for Touch

- Open **MPLAB® Data Visualizer** by simple click on icon  , you could **double click** on **MPLAB Data Visualizer** tab to enlarge it to full window as below.



⚙️ Lab3 Data Visualizer for Touch

- In Connections Tab, click on the **UART COM Port** you to output debug message from **SERCOM5** and configure to **115200-n-8-1**.



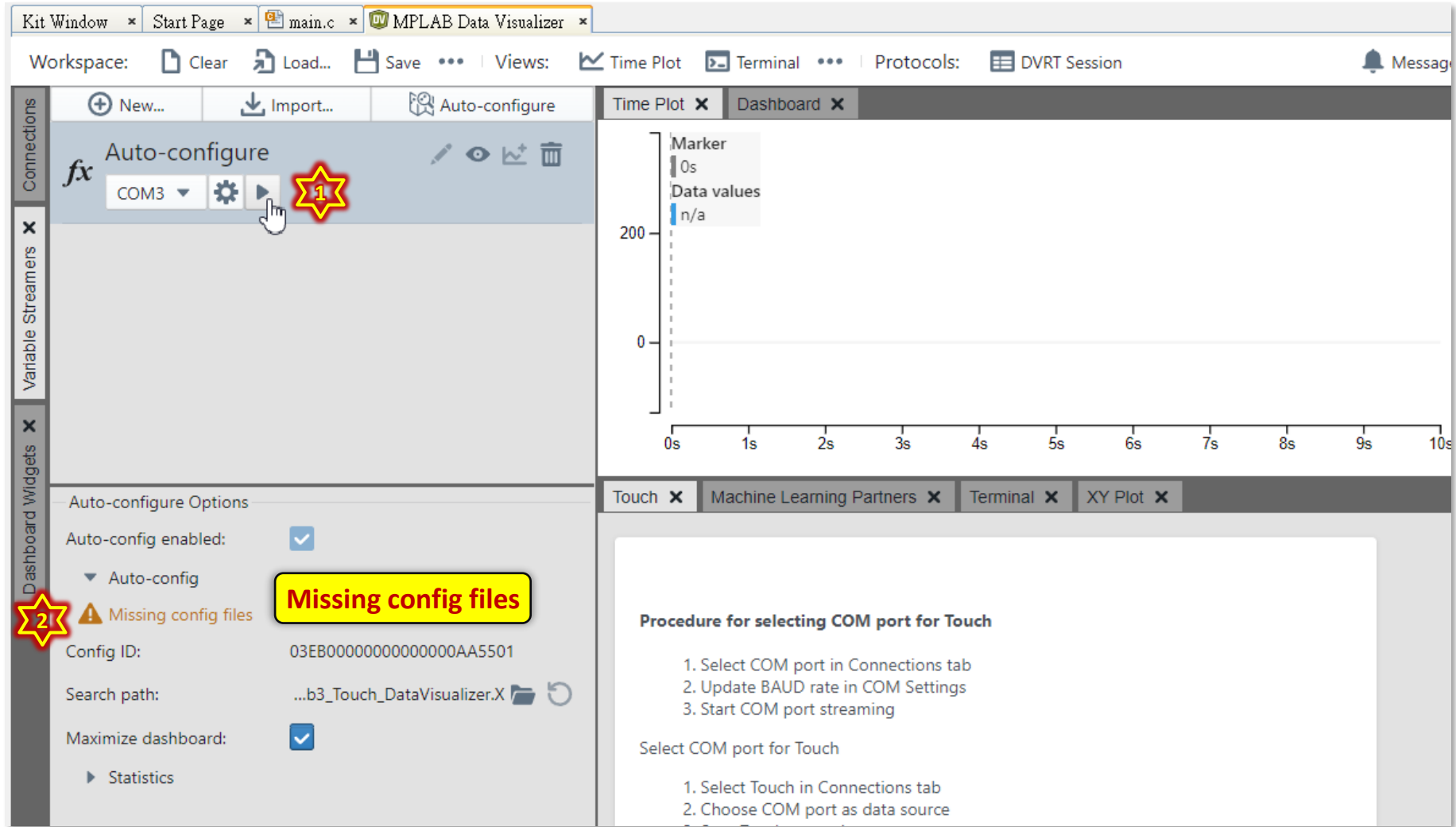
⚙️ Lab3 Data Visualizer for Touch

- Click on **Variable Streamers** tab, then click the **Auto-configure** to add a new Streamer.
- Select the same **COM Port** as Auto-configuration source.

The screenshot displays the MPLAB Data Visualizer software interface. The top toolbar includes buttons for 'New...', 'Import...', 'Auto-configure', 'Time Plot', 'Terminal', and 'Protocols'. The left sidebar contains tabs for 'Connections', 'Variable Streamers', and 'Dashboard Widgets'. The 'Variable Streamers' tab is active, showing the 'Auto-configure' dialog. A red star with the number '1' is placed over the 'Auto-configure' button. The dialog has a 'No Source' dropdown menu, which is open, showing a list of options: 'No Source', 'Touch Session on Touch', 'COM1', 'COM2', and 'COM3'. A red star with the number '2' is placed over the 'Auto-configure' button, and a red star with the number '3' is placed over the 'COM3' option in the dropdown. The 'Auto-configure Options' section below the dialog shows 'Auto-config enabled' checked, 'Config ID' as 'Not decoded', 'Search path' as '...b3_Touch_DataVisualizer.X', 'Recursive search' checked, and 'Maximize dashboard' checked. The right pane shows a 'Time Plot' with a y-axis labeled 'Marker' and 'Data values' ranging from 0 to 200, and an x-axis labeled '0s' to '10s'. The 'Touch' tab is selected in the bottom right pane, displaying a 'Procedure for selecting COM port for Touch' with three steps: 1. Select COM port in Connections tab, 2. Update BAUD rate in COM Settings, and 3. Start COM port streaming. Below this, it says 'Select COM port for Touch' followed by two steps: 1. Select Touch in Connections tab and 2. Choose COM port as data source.

⚙️ Lab3 Data Visualizer for Touch

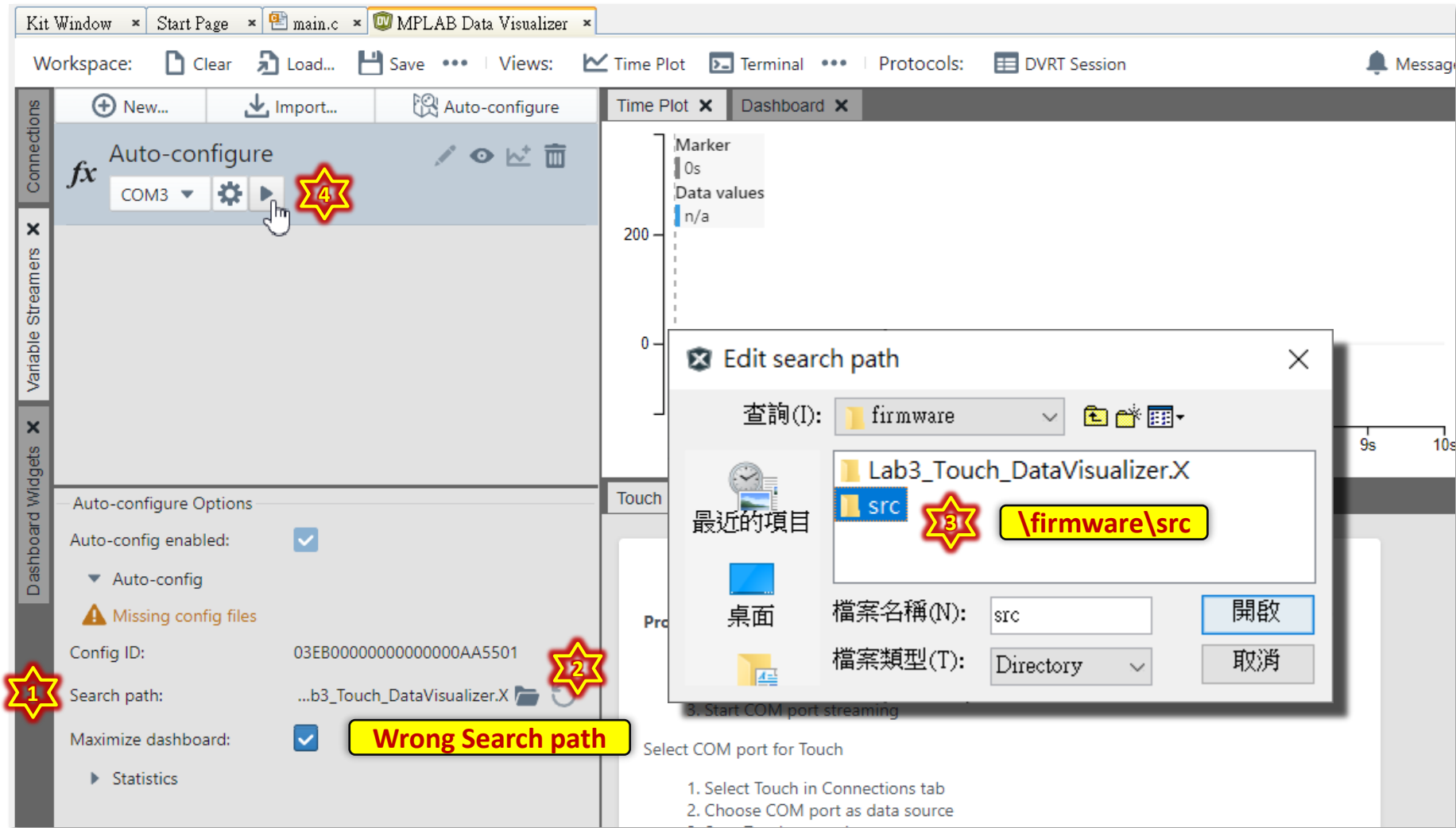
- Click on  to start communicate then you will see a **Miss configure files** message.



The screenshot displays the MPLAB Data Visualizer software interface. The top toolbar includes buttons for 'New...', 'Import...', and 'Auto-configure'. The 'Auto-configure' panel shows 'COM3' selected with a play button icon highlighted by a red star labeled '1'. The left sidebar contains 'Connections', 'Variable Streamers', and 'Dashboard Widgets'. The 'Dashboard Widgets' section shows 'Auto-configure Options' with 'Auto-config enabled' checked. A yellow star labeled '2' points to a warning icon and the text 'Missing config files'. A yellow callout box with the text 'Missing config files' is also present. The main area shows a 'Time Plot' with 'Marker' at '0s' and 'Data values' at 'n/a'. The bottom right panel contains a 'Procedure for selecting COM port for Touch' with three steps: 1. Select COM port in Connections tab, 2. Update BAUD rate in COM Settings, 3. Start COM port streaming. Below this, it says 'Select COM port for Touch' with two steps: 1. Select Touch in Connections tab, 2. Choose COM port as data source.

⚙️ Lab3 Data Visualizer for Touch

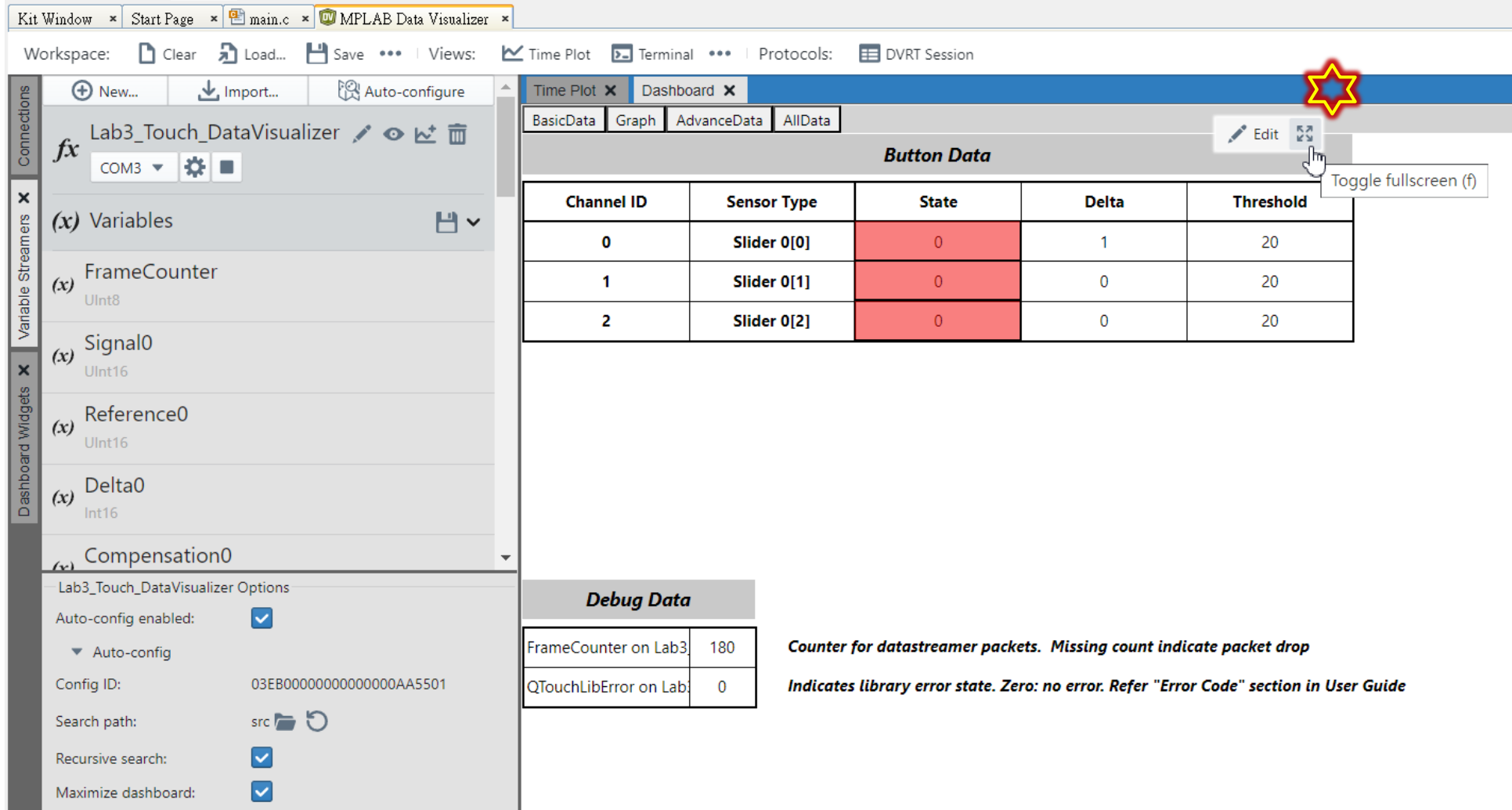
- The default Auto-configuration Options **Search Path** is wrong, please click on Icon  to change folder to **\firmware\src** then click  again.



The screenshot shows the MPLAB Data Visualizer interface. The 'Auto-configure' panel is active, displaying 'COM3' as the selected port. A red star with the number '4' is placed over the play button icon in the 'Auto-configure' section. Below this, the 'Auto-configure Options' section shows 'Auto-config enabled' checked. The 'Search path' is listed as '...b3_Touch_DataVisualizer.X', with a red star and the number '2' next to it. A yellow box with the text 'Wrong Search path' is overlaid on this section. A red star with the number '1' is placed over the folder icon next to the search path. An 'Edit search path' dialog box is open, showing a list of recent items. The path '\firmware\src' is highlighted with a red star and the number '3'. The '檔案名稱(N):' field contains 'src' and the '檔案類型(T):' is set to 'Directory'. The '開啟' (Open) button is visible.

⚙️ Lab3 Data Visualizer for Touch

- If all configuration are correctly, you will see a **Touch Dashboard** to start up.
- Click on Icon  to enlarge Dashboard window.



Kit Window x Start Page x main.c x MPLAB Data Visualizer x

Workspace: Clear Load... Save ... Views: Time Plot Terminal ... Protocols: DVRT Session

Connections

Lab3_Touch_DataVisualizer

COM3

Variable Streamers

- (x) Variables
- (x) FrameCounter (UInt8)
- (x) Signal0 (UInt16)
- (x) Reference0 (UInt16)
- (x) Delta0 (Int16)
- (x) Compensation0

Dashboard Widgets

Lab3_Touch_DataVisualizer Options

Auto-config enabled: ☒

Auto-config

Config ID: 03EB000000000000AA5501

Search path: src

Recursive search: ☒

Maximize dashboard: ☒

Time Plot x Dashboard x

BasicData Graph AdvanceData AllData

Edit

Toggle fullscreen (f)

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Slider 0[0]	0	1	20
1	Slider 0[1]	0	0	20
2	Slider 0[2]	0	0	20

Debug Data

FrameCounter on Lab3	180
QTouchLibError on Lab3	0

Counter for datastreamer packets. Missing count indicate packet drop

Indicates library error state. Zero: no error. Refer "Error Code" section in User Guide

Lab3 Data Visualizer for Touch

- **Basic Data** window of Touch Dashboard.



BasicData | Graph | AdvanceData | AllData

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Slider 0[0]	0	2	20
1	Slider 0[1]	1	226	20
2	Slider 0[2]	0	2	20

Slider & Wheel Data

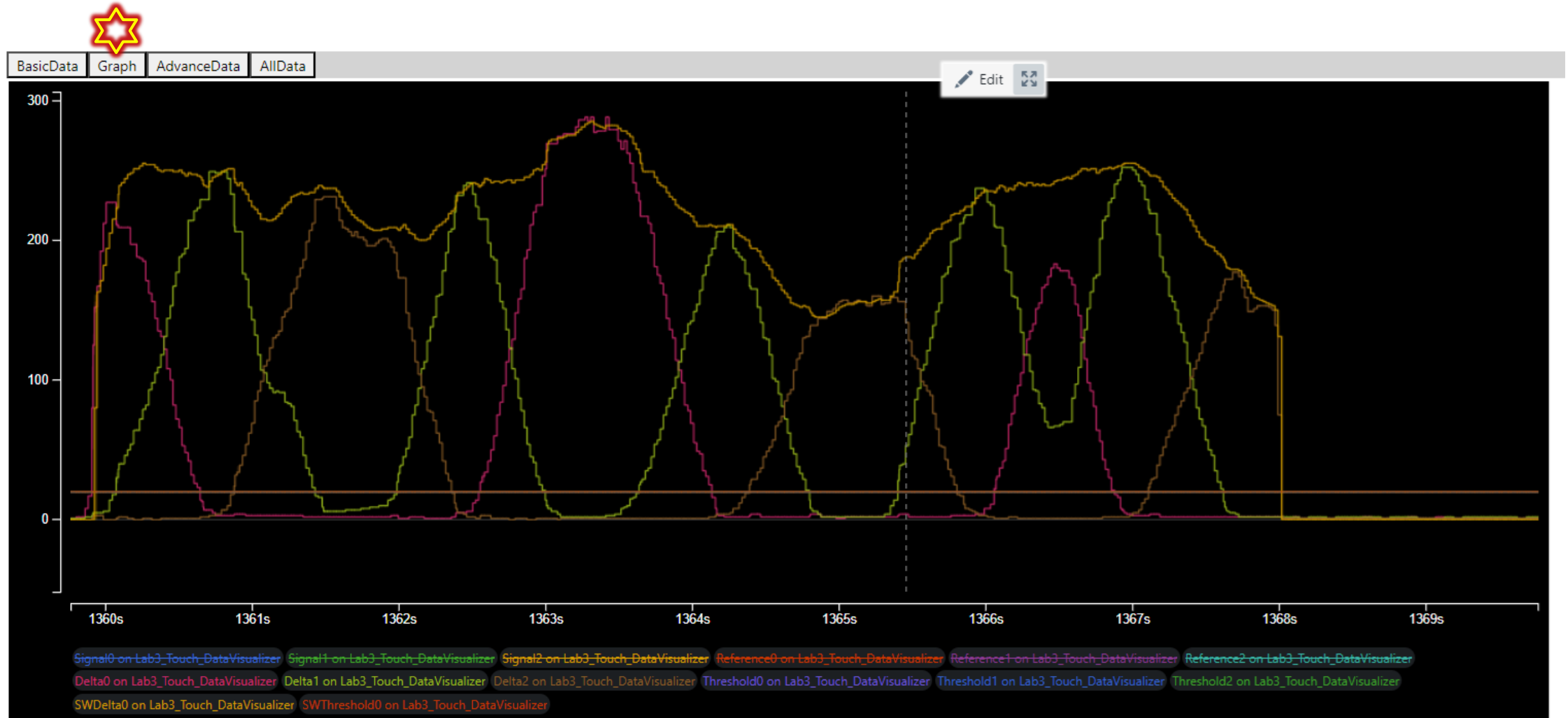
Sensor	State	SW Delta	SW Threshold	Position
Slider 0	1	232	20	128

Debug Data

FrameCounter on Lab3	128	<i>Counter for datastreamer packets. Missing count indicate packet drop</i>
QTouchLibError on Lab	0	

⚙️ Lab3 Data Visualizer for Touch

- **Graph** window of Touch Dashboard.



* Vertical Zoom: Ctrl Key + Mouse Scroll (uncheck "Automatically fit Y" option).

* Horizontal Zoom: Left-Shift Key + Mouse Scroll.

* Click on "Legend" to enable or disable plot.

Lab3 Data Visualizer for Touch

- **Advance Data** window of Touch Dashboard.



BasicData | Graph | **AdvanceData** | AllData

 Edit 

Sensor Data

Channel ID	Sensor Type	Signal	Reference	Delta	Compensation pF
0	Slider 0[0]	508	508	0	17.39475
1	Slider 0[1]	507	506	1	17.93475
2	Slider 0[2]	507	507	0	21.78225

Compensation: Represents PTC compensation circuit value which is equivalent to sensor capacitance

"Compensation" value can be used to check whether sensor is saturated. Refer to User Guide

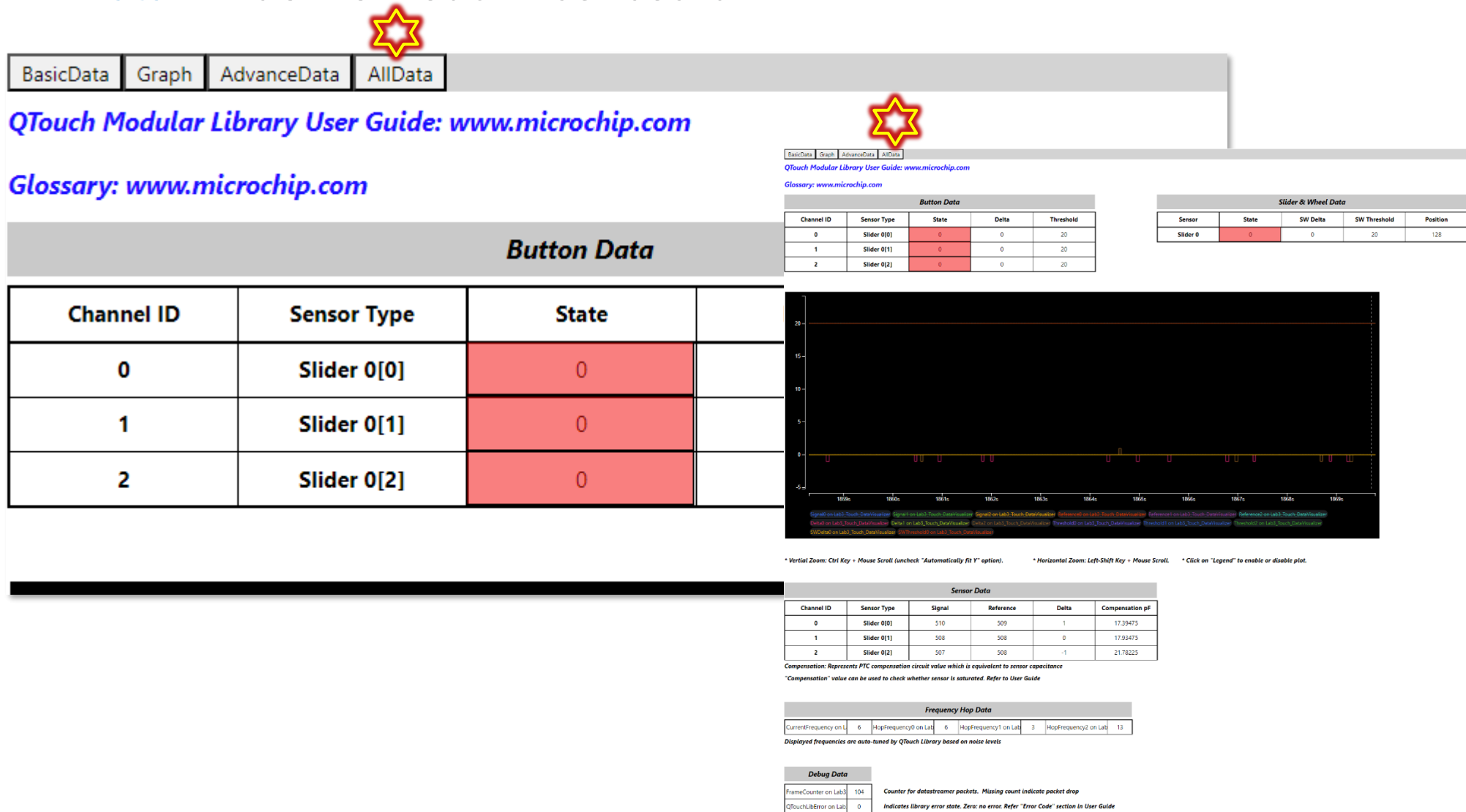
Frequency Hop Data

CurrentFrequency on L	10	HopFrequency0 on Lab	9	HopFrequency1 on Lab	10	HopFrequency2 on Lab	0
-----------------------	----	----------------------	---	----------------------	----	----------------------	---

Displayed frequencies are auto-tuned by QTouch Library based on noise levels

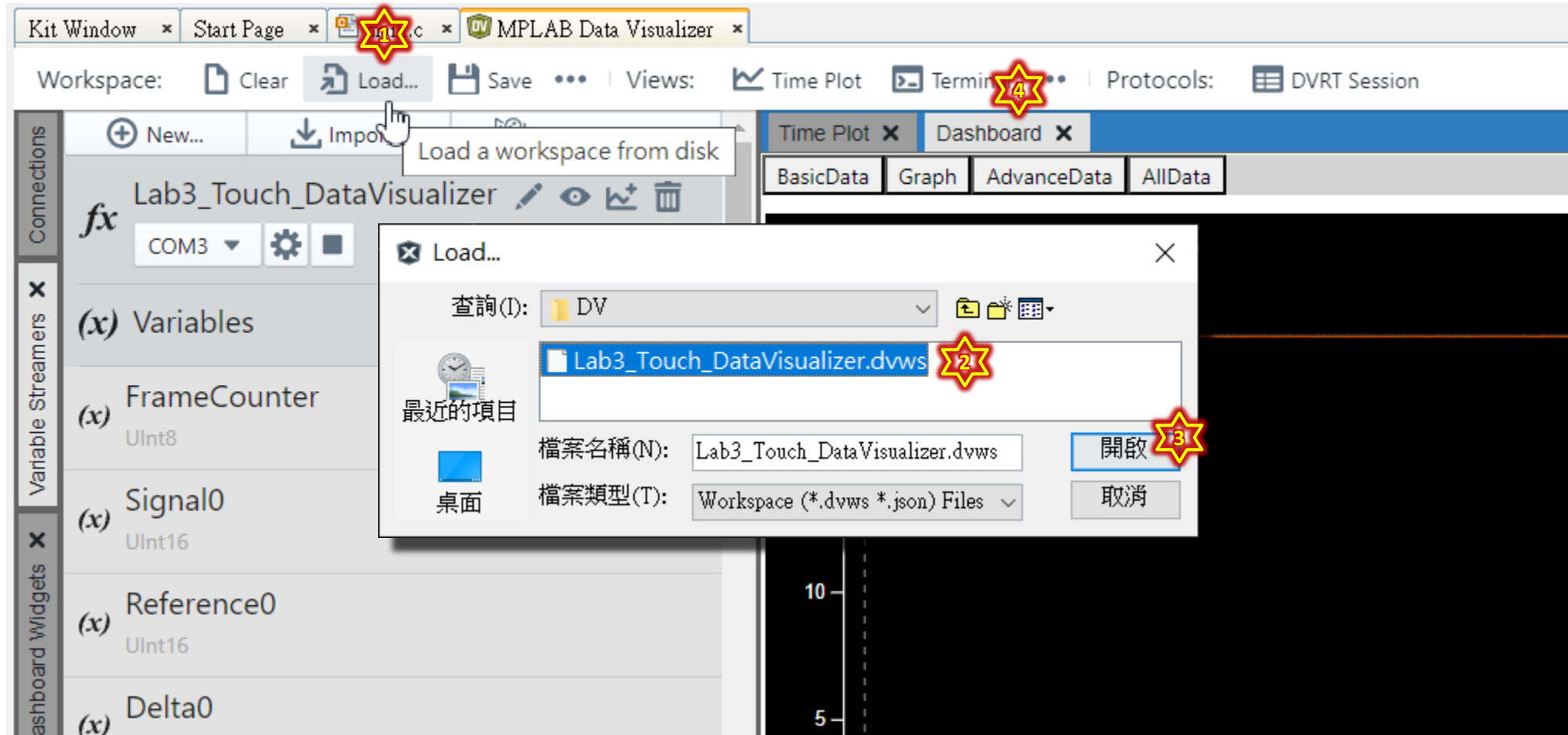
Lab3 Data Visualizer for Touch

- All Data window of Touch Dashboard.



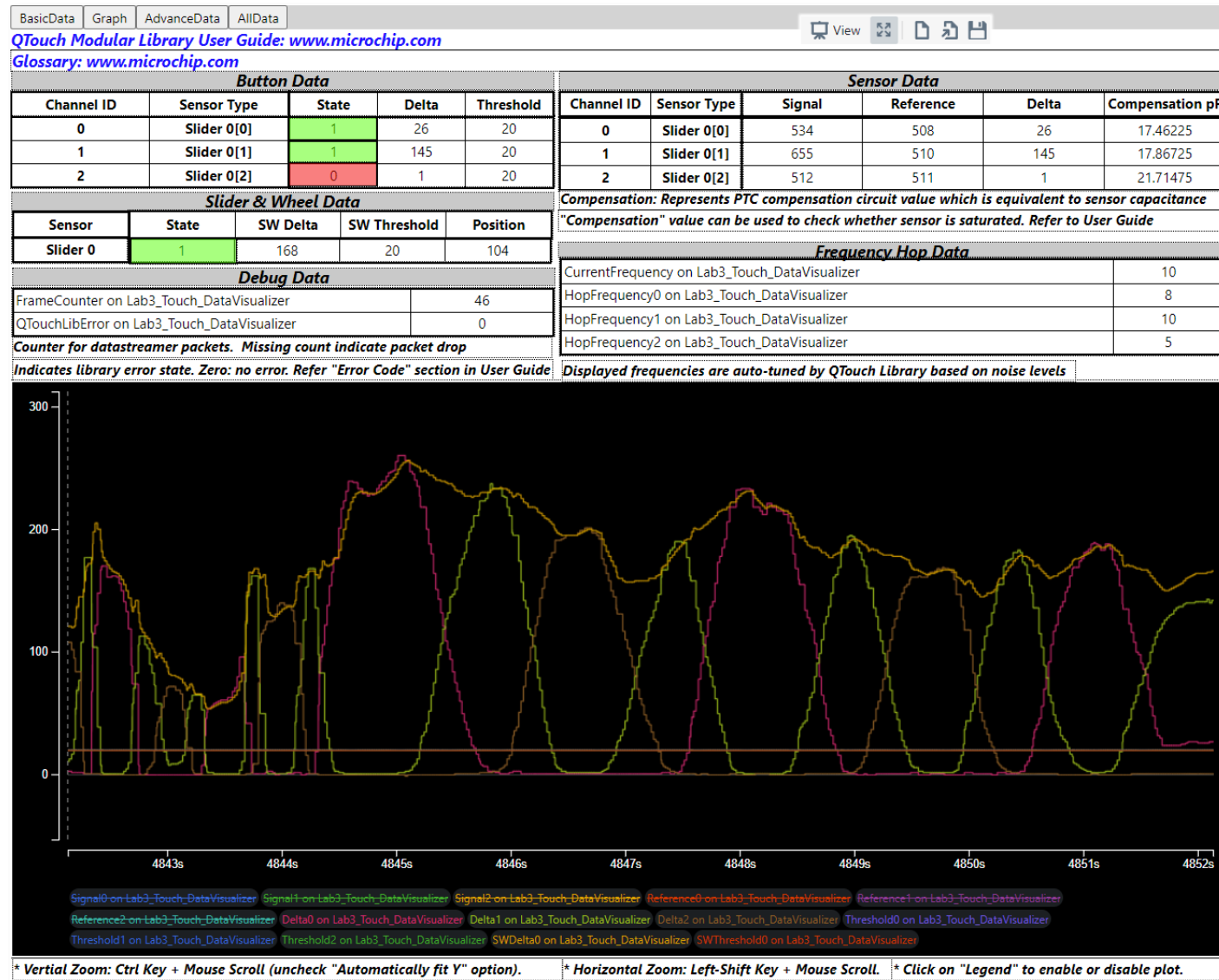
⚙️ Lab3 Data Visualizer for Touch

- Load a customized Dashboard **Lab3_Touch_DataVisualizer.dvws** from **\DV** folder of project to show compact **All Data**.



✖ Lab3 Data Visualizer for Touch

- The customized **All Data** of Dashboard



USB Fundamentals

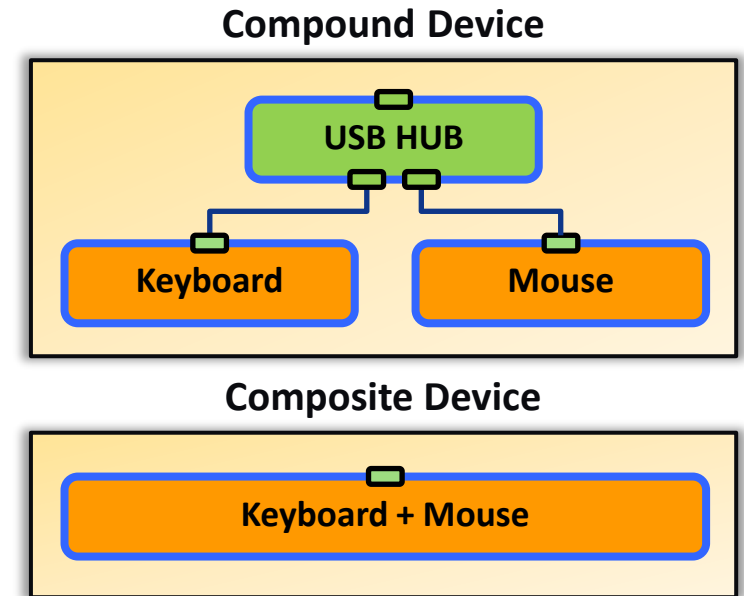
USB, Universal Serial Bus

- USB was co-developed by a group of companies, Compaq, Intel, Microsoft and NEC etc. Who wanted to make it much easier to add/remove peripheral devices from PCs
- History
 - USB 0.7 – 1994 -> USB 1.0 - Jan. 1996 -> USB 1.1 - Sep. 1998 ->
 - USB 2.0 - Apr. 2000
 - USB 1.0 - > Low Speed 1.5 Mbps
 - USB 1.1 - > Full Speed 12 Mbps
 - USB 2.0 - > High Speed 480 Mbps
 - USB 3.0 - Nov. 2008 -> USB 3.1 - Jul. 2013 ->
 - USB 3.2
 - USB 3.0 -> USB 3.2 Gen 1 – SuperSpeed, 5Gbps
 - USB 3.1 -> USB 3.2 Gen 2 – SuperSpeed 10G, 10Gbps
 - USB 3.2 Gen 2x2 - SuperSpeed 20G, 20Gbps
 - USB 4.0 - Sep. 2019
 - 40 Gbps

On-The-Go Supplement 1.3 - Dec.2006
Battery Charging Specification 1.0
(Max. 1.5A) - Mar. 2007
USB Power Delivery - Jul. 2012
Wireless USB

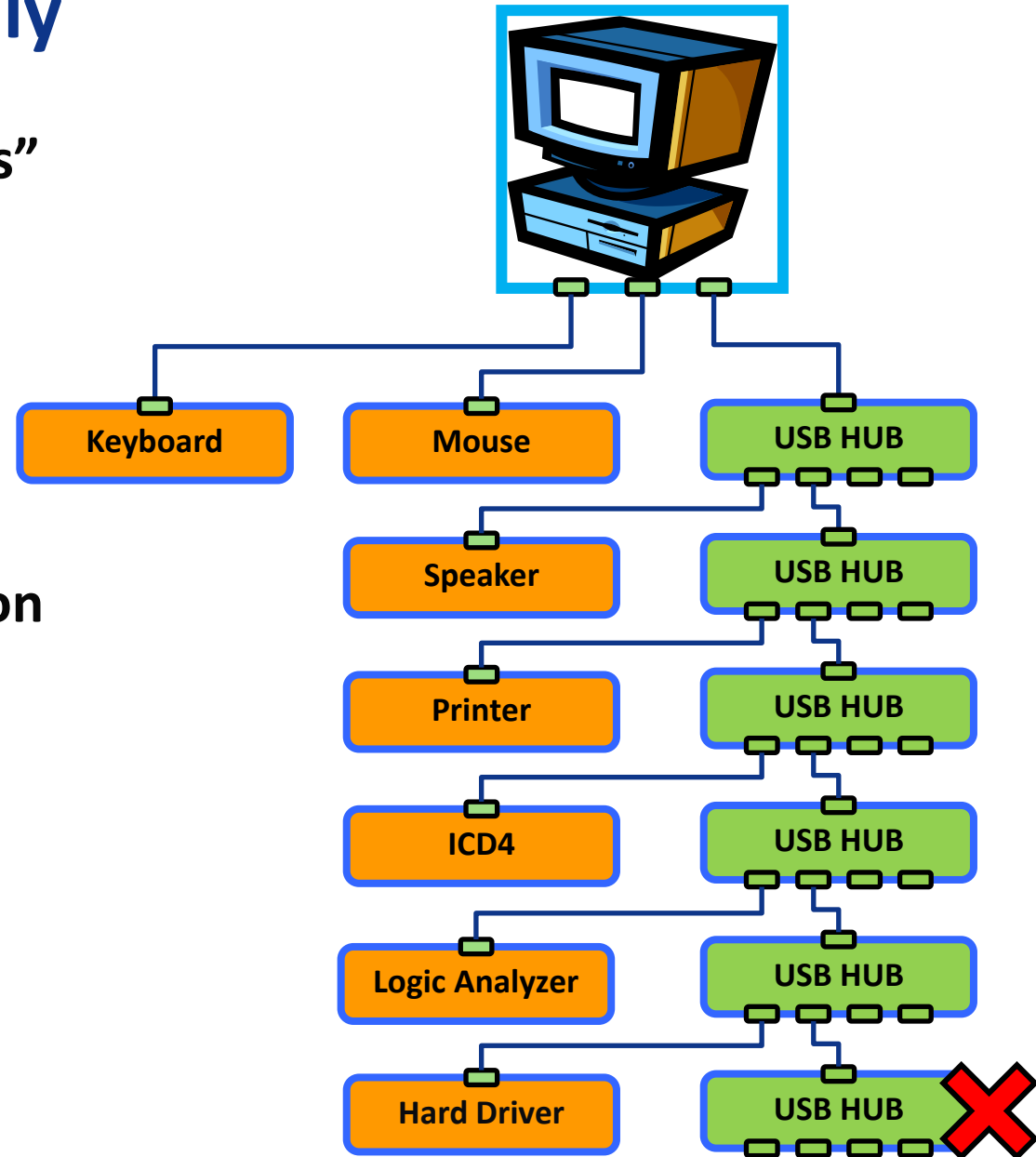
USB Device Types

- **Peripheral (also called “Function”)**
 - Provides a functionality (capability) to the host
 - E.g. Keyboard, mouse, ICD4
- **USB HUB**
 - Repeats traffic (both directions), Extends bus
 - Provides extra power and manages power
- **Compound Device**
 - Contains a hub and 1 or more peripheral
 - Host treats hub and peripheral function separately (each has its own address)
 - E.g. USB keyboard and mouse with hub
- **Composite Device**
 - Has multiple interfaces active at the same time
 - Host loads a driver for each interface
 - E.g. USB keyboard and mouse set (both keyboard & mouse interfaces active)



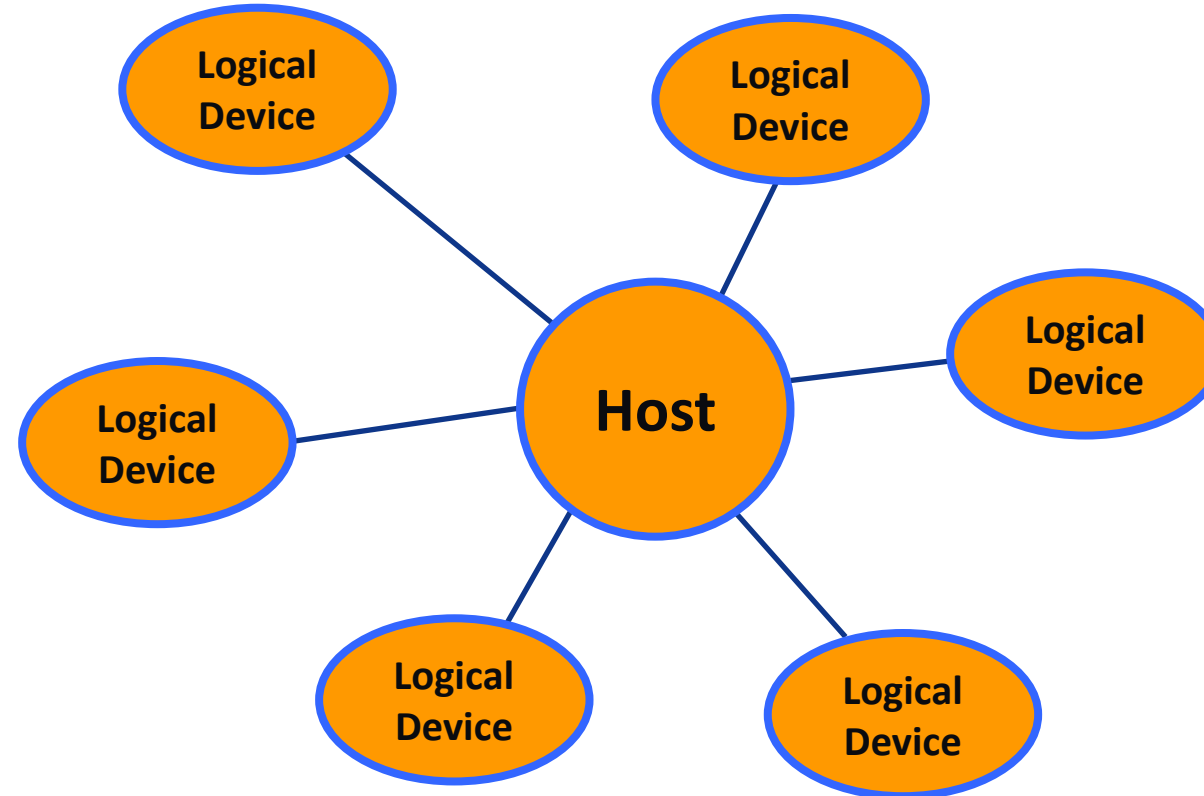
Tiered Star Topology, Physically

- USB is a “Single Master, Multiple Slaves” polled bus.
Master Initiates all data transfer and Schedules traffic.
- 5 chained hub limit.
- Up to 127 devices at bus.
- 100 mA Guaranteed for un-configuration device, bus powered up to 500 mA for configuration device.



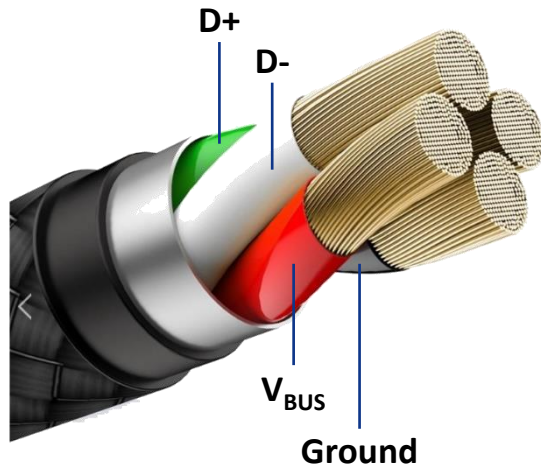
Star Topology, Logical

- Logically, Not a tiered-star!
- Host software communicates to each “logical” device as if it were directly connected to the root hub.



USB 2.0 cable and Physical Interface

- USB cables are four-wire connection.
- D+ and D- are used for half duplex communication with NRZI encoding.
 V_{BUS} can be used to supply power to the peripheral.
- The peripheral needs to be able to work with V_{BUS} between 4.40 and 5.25V, and can require up to 100mA (guaranteed)
- The peripheral can require up to 500mA but through a negotiation. If peripheral require more than 500mA, it needs an external power supplier (Self-Power).



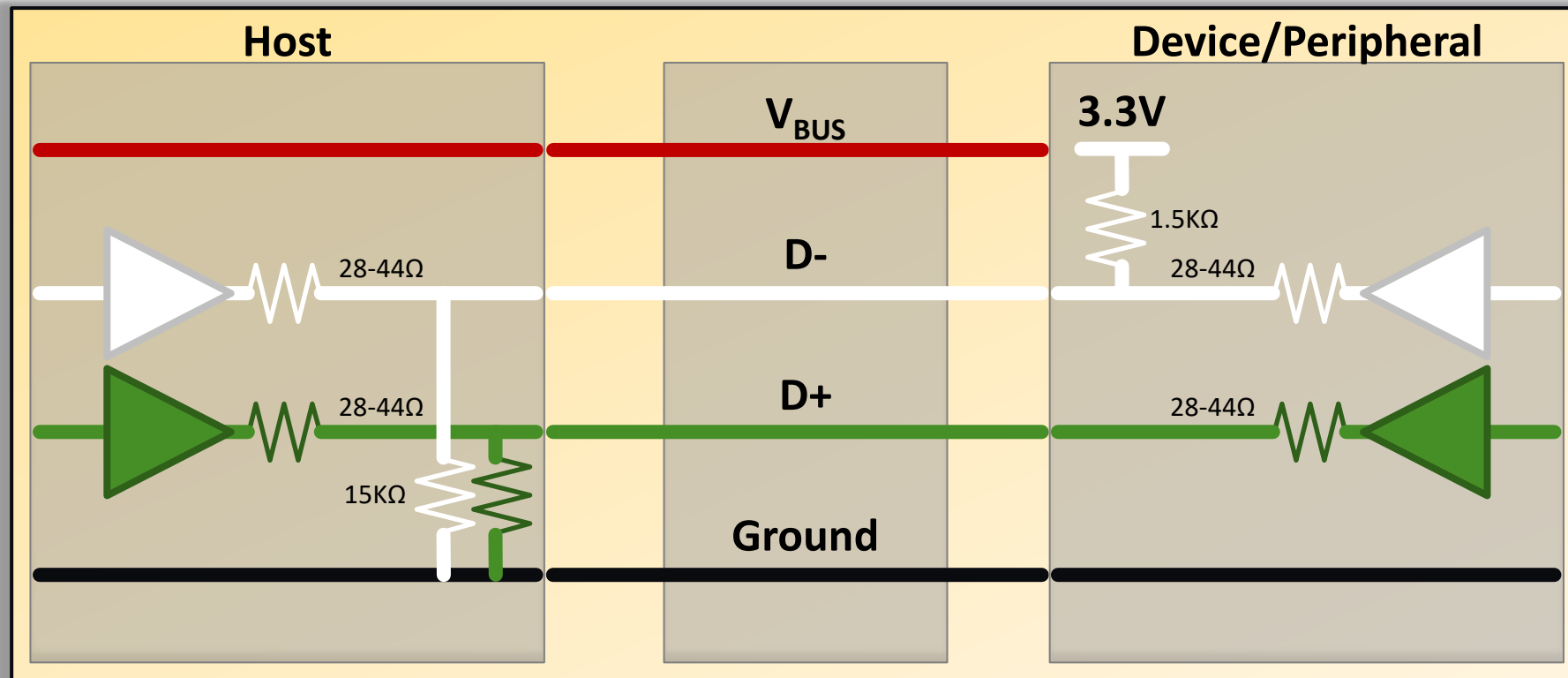
<https://www.baseusworld.com/products/baseus-double-fast-charging-usb-cable-usb-for-type-c-5a-1m>

VBUS	~ 5.0V
D+/D-	~ 3.3V

Half Duplex with NRZI (No Return to Zero Invert) Data Encoding,
Bus Power to each device:
 V_{BUS} Between 4.40V - 5.25V
Guaranteed 100 mA, 500 mA maximum through negotiation

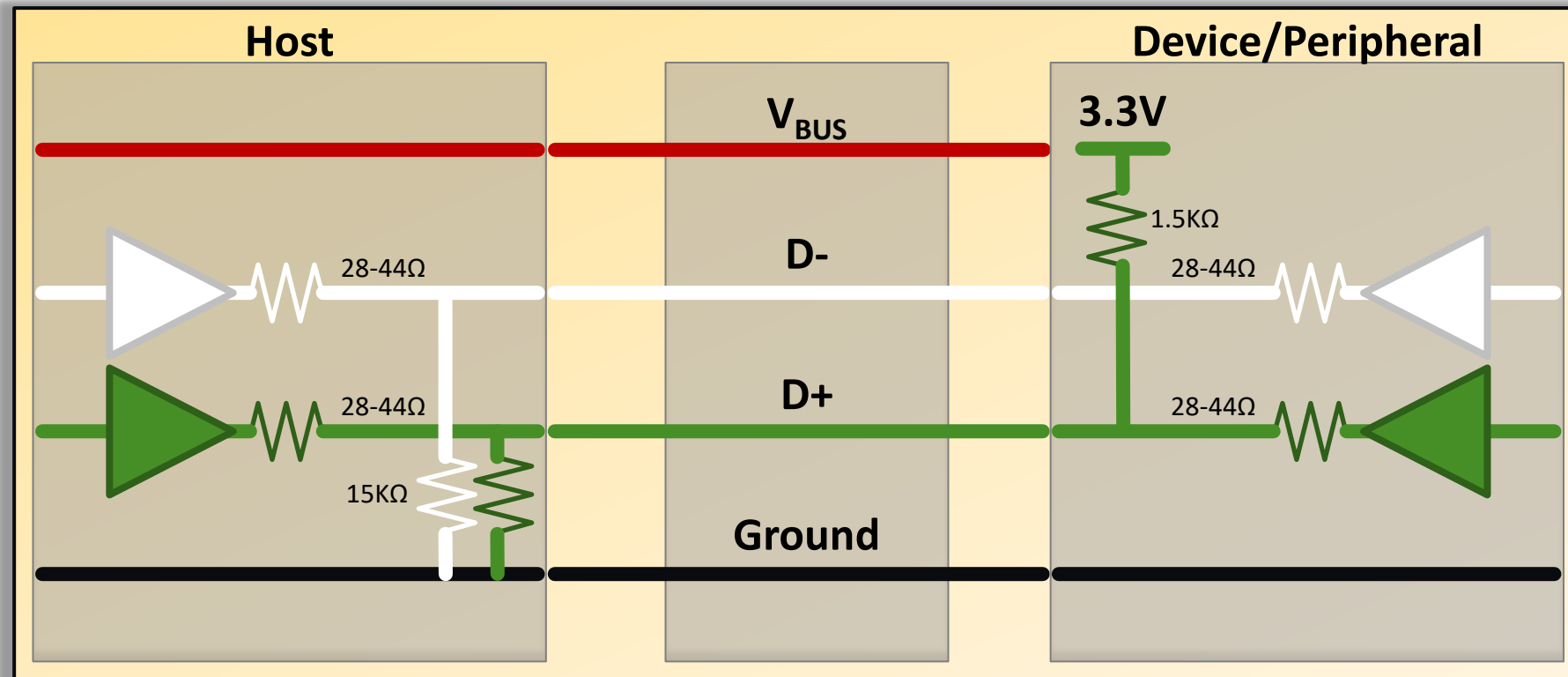
Attached : Low-Speed

- USB host use pull up resistors to recognizes the device speed.
- The auto-detection is achieved moving D lines after cable connection
- If the device pull-up D-, the host recognizes that a “Low speed” device has been connected.



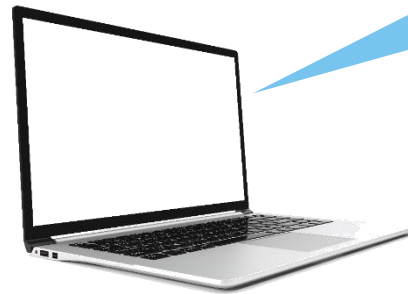
Attached : Full/High-Speed

- If the device pull-up D+, the host recognizes that a “Full speed” device has been connected.
- Each “High speed” device initially starts “Full speed” mode when attached to the bus and then negotiates to switch to high speed.



Enumeration

- USB was created to be a self-describing bus.
- Each peripheral when plugged in will tell the host what it is and how it works.
- When a device is plugged in, the host will say “Hi, who are you?”
- The device will respond with information about itself that the host needs.
- The host then tells the device if it has been accepted to the bus or not. This process called “Enumeration”.



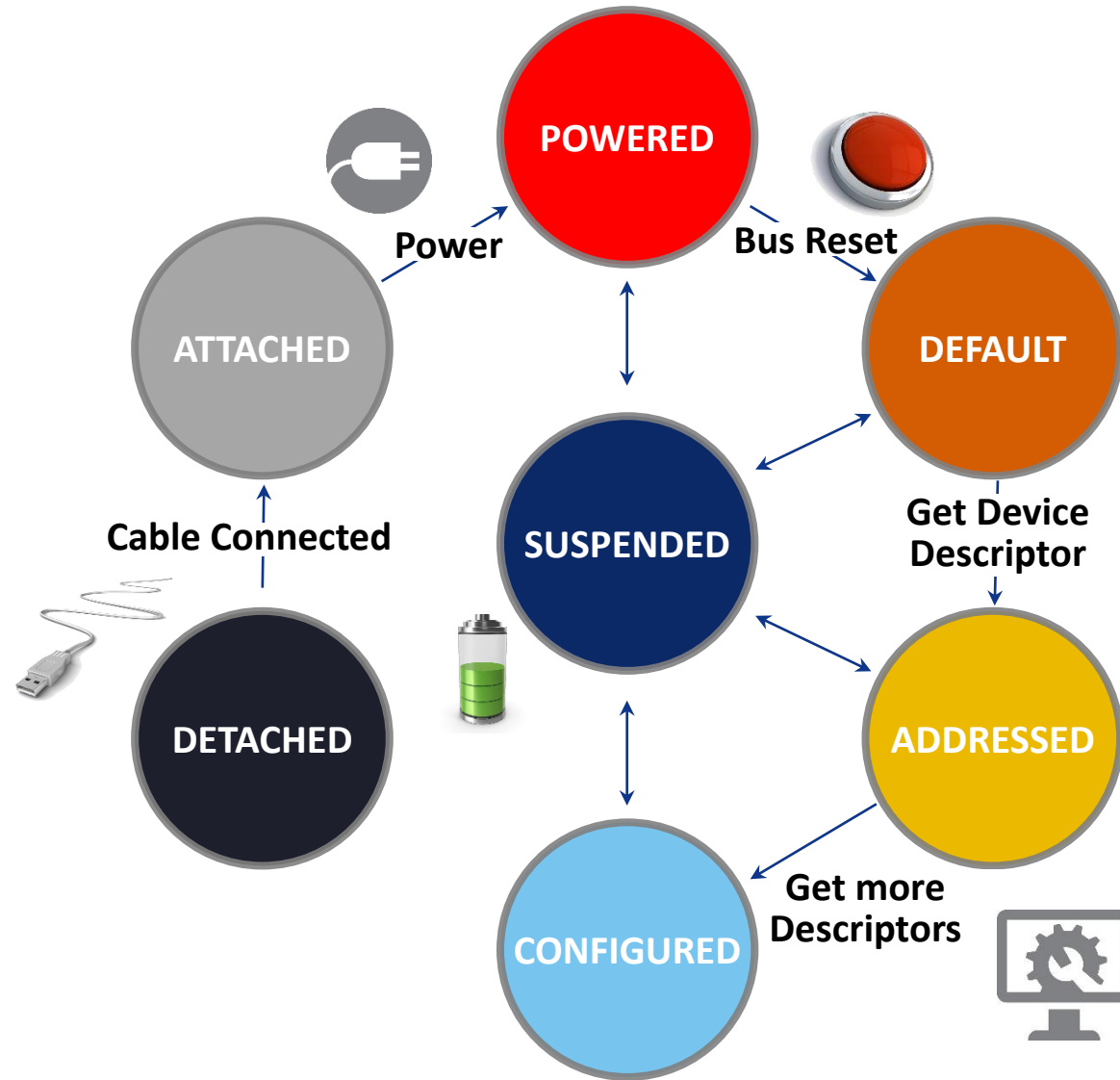
Hi, who are you?



Hi! I'm a mouse.

We have a driver for you.
Welcome to the system.

Enumeration Process



Descriptors

- **Device Descriptor**

- Overview the device, VID, PID, Num of Configurations ?, Num of Strings ?

- **Configuration Descriptor**

- The details of the device, Power mode and needs ?, Num of Interface ?

- **Interface Descriptor**

- Function Detail,
Num of Endpoint ?

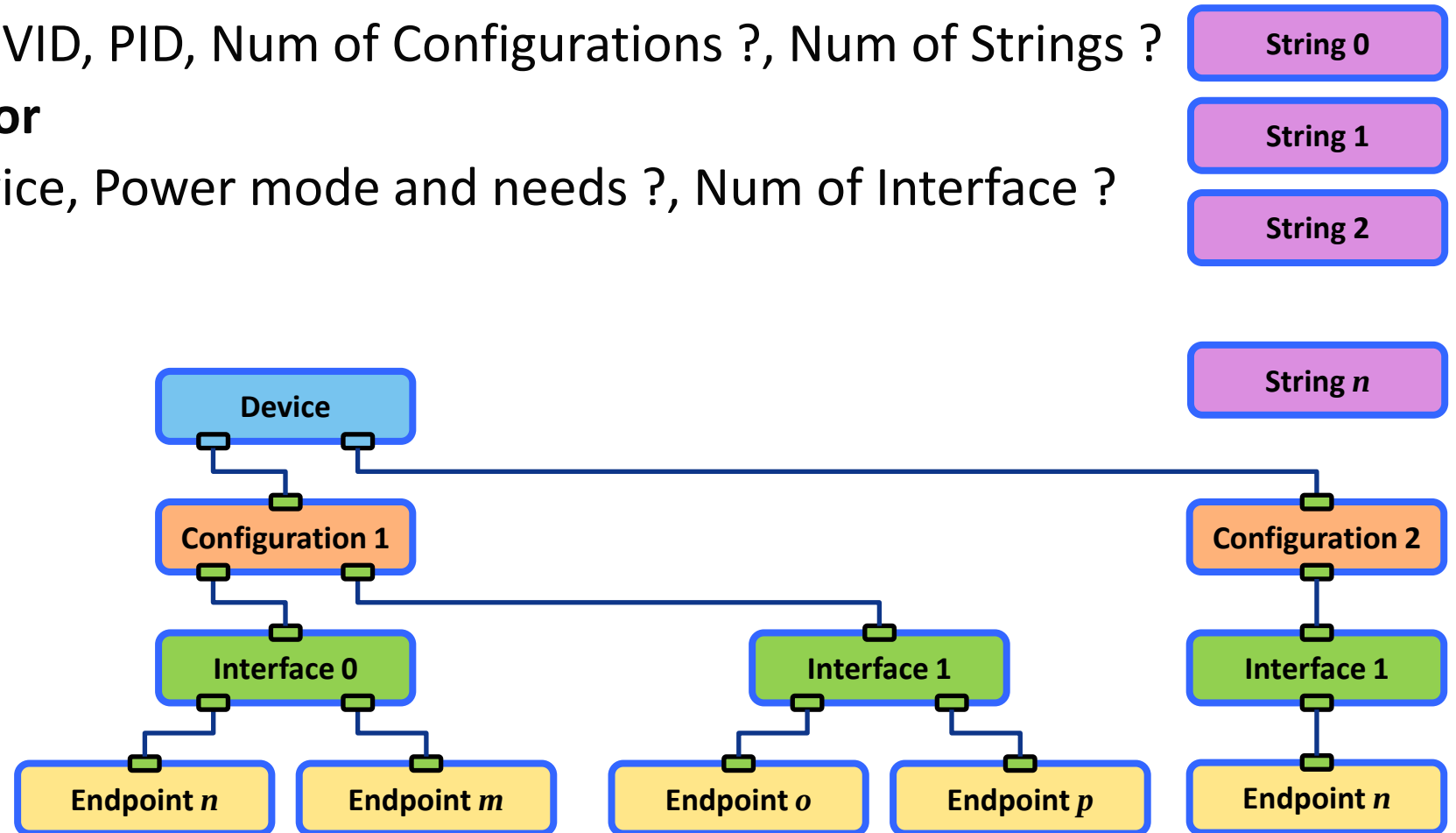
- **Endpoint Descriptor**

- Transfer Type

- **String Descriptor**

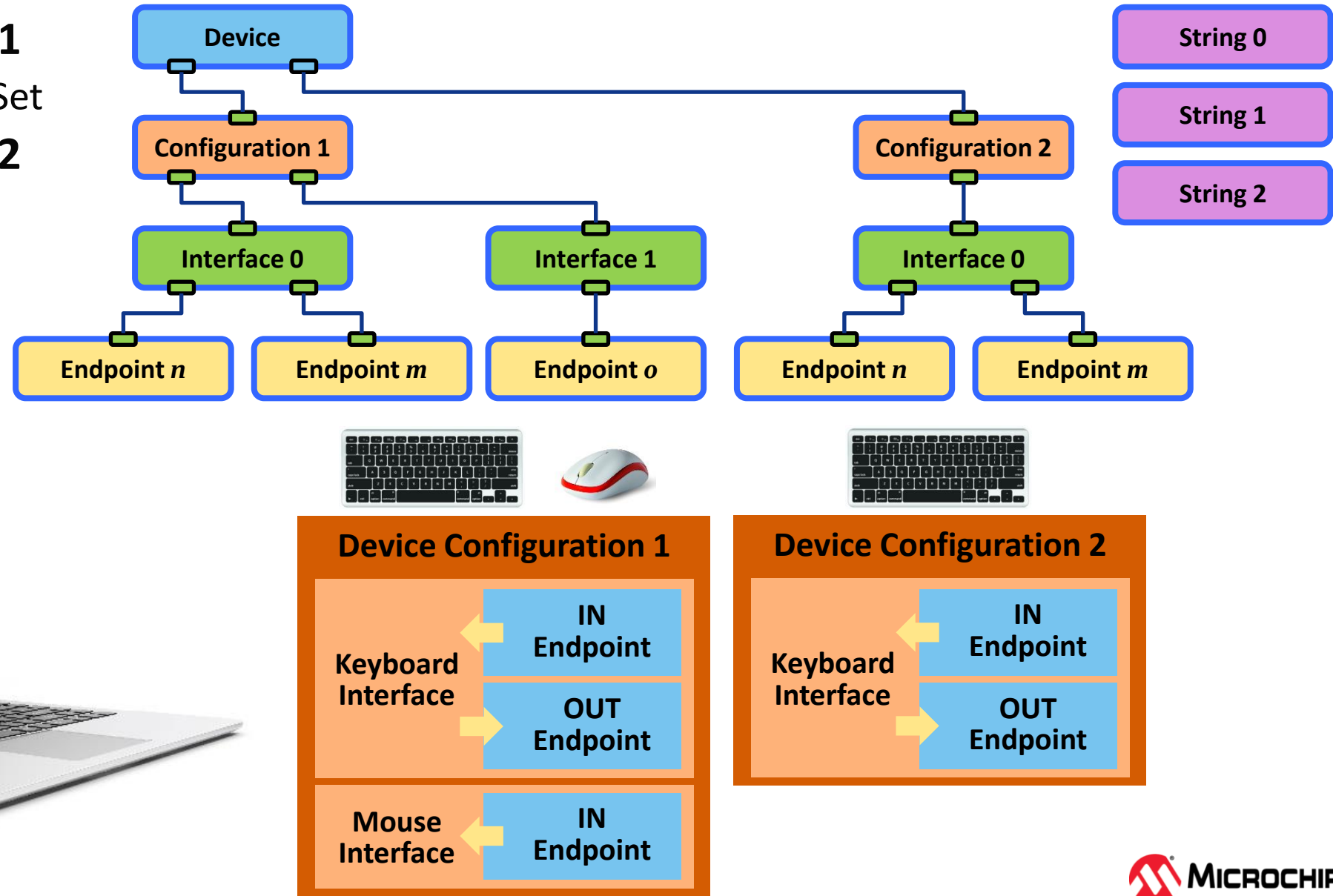
- **Driver Class Specific**

- **Many more...**



Configuration Descriptor Example

- **Device Configuration 1**
 - Keyboard with Mouse Set
- **Device Configuration 2**
 - Keyboard only



Data Transfer Types

- **Interrupt Transfer**

- Periodic Transfer.
- Guaranteed latency, Guaranteed delivery.
- Low throughput.

- **Isochronous Transfer**

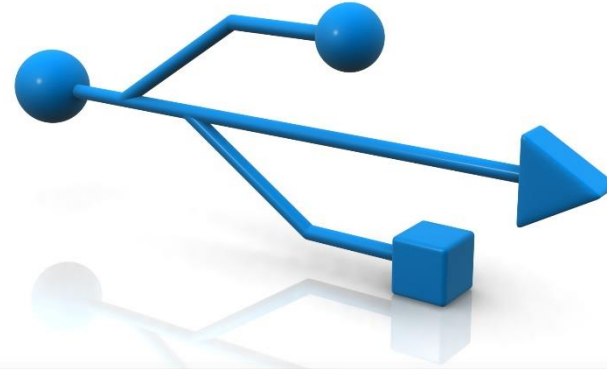
- Periodic Transfer, Guaranteed latency.
- Delivery not Guaranteed.
- Moderate/high throughput.

- **Bulk Transfer**

- Indeterminant latency, Guaranteed delivery.
- Variable throughput, typically high.

- **Control Transfer**

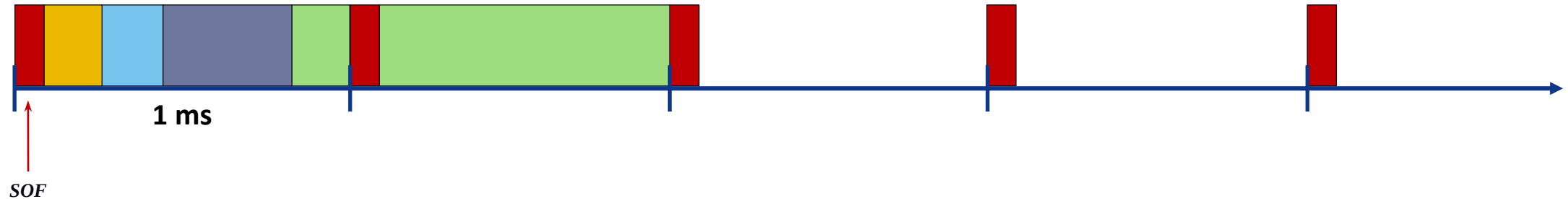
- Used to configure the device.
- All devices use control transfers.
- Guaranteed delivery and Low throughput.



<u>Transfer/ Endpoint Type</u>	<u>Polling Interval</u>	<u>% Reserved BW/Frame for all transfers of this type</u>	<u>Max. # Data Bytes/Frame/Endpoint (Max# transactions per frame @ Max Ep Size)*</u>	<u>Data Integrity</u>
Interrupt	Fixed, Periodic	10	64 (1 x 64)	Yes
Isochronous	Fixed, Periodic	90	1,023 (1 x 1023)	No
Bulk	Variable, Uses Free Bandwidth	0	1,216 (19 x 64)	Yes
Control	Variable	10	832 (13 x 64)	Yes

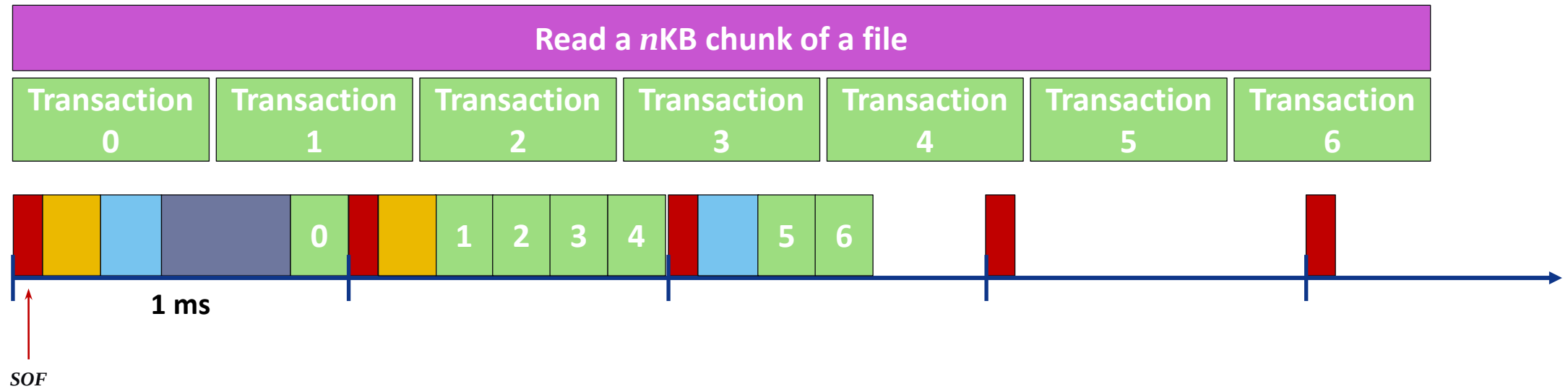
Data Transfer Scheduling

- 10% reserved for non-periodic (**control** and **bulk**)
- 90% reserved for periodic transfers (**interrupt** and **isochronous**)
- Any extra given to **bulk**

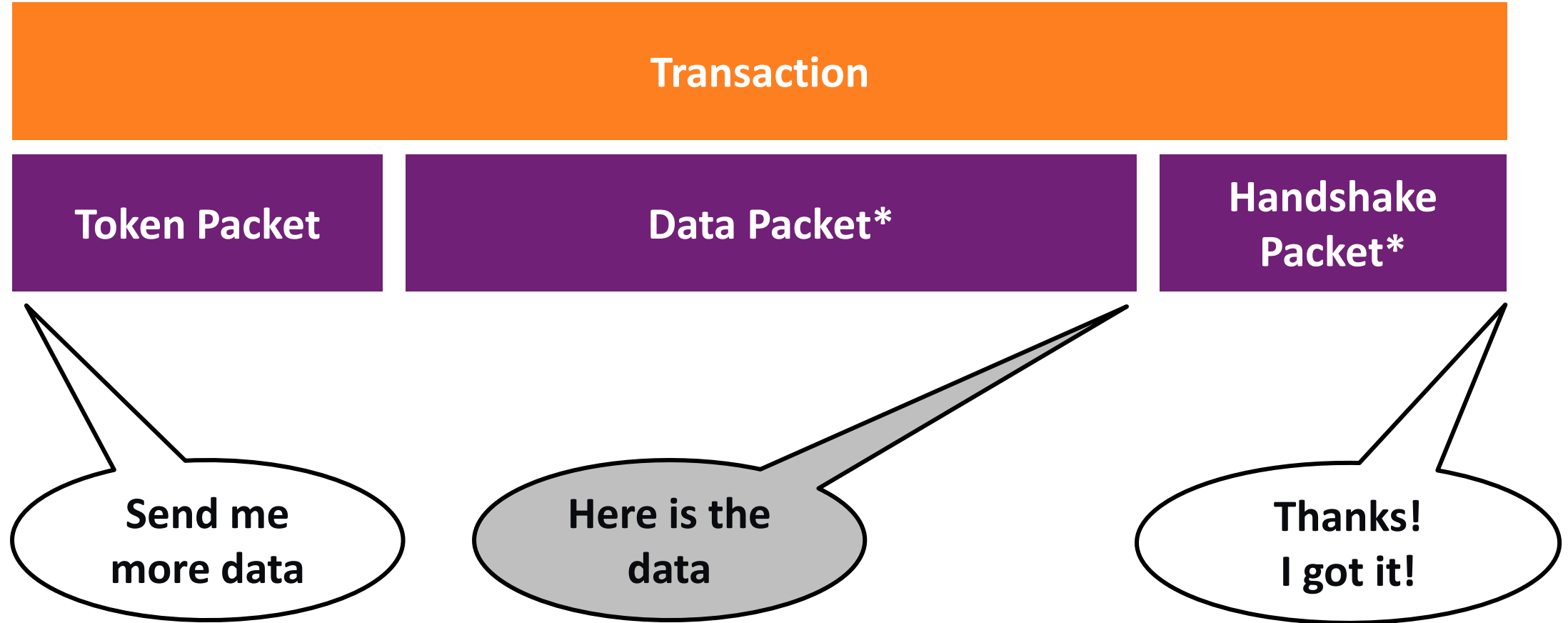


USB Transactions

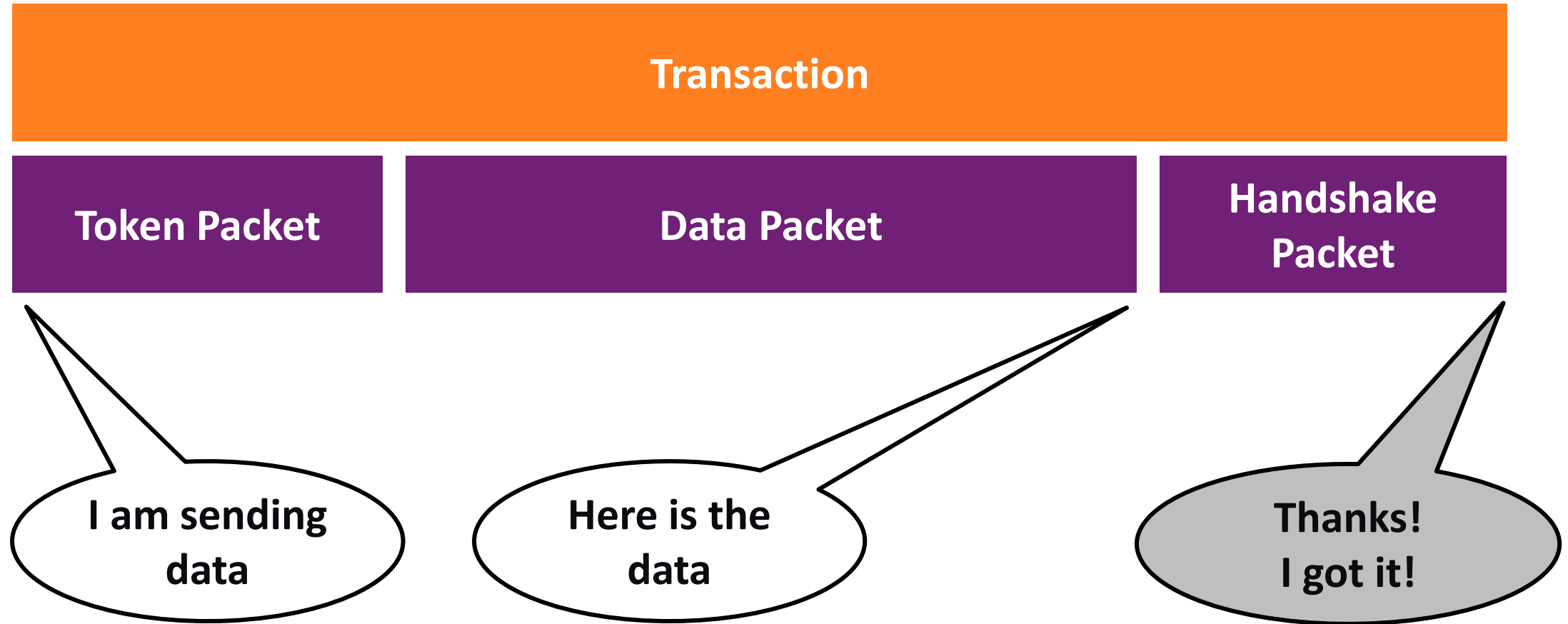
- Specification defined maximum packet size for endpoint.



USB Transactions - Require Data



USB Transactions - Send Data



USB Transactions, Token & Handshake

Token Packet

IN
OUT
SETUP
PING

IN : Require Data
OUT : Send Data
SETUP : Control Operation
PING : Confirm Device Ready (2.0)

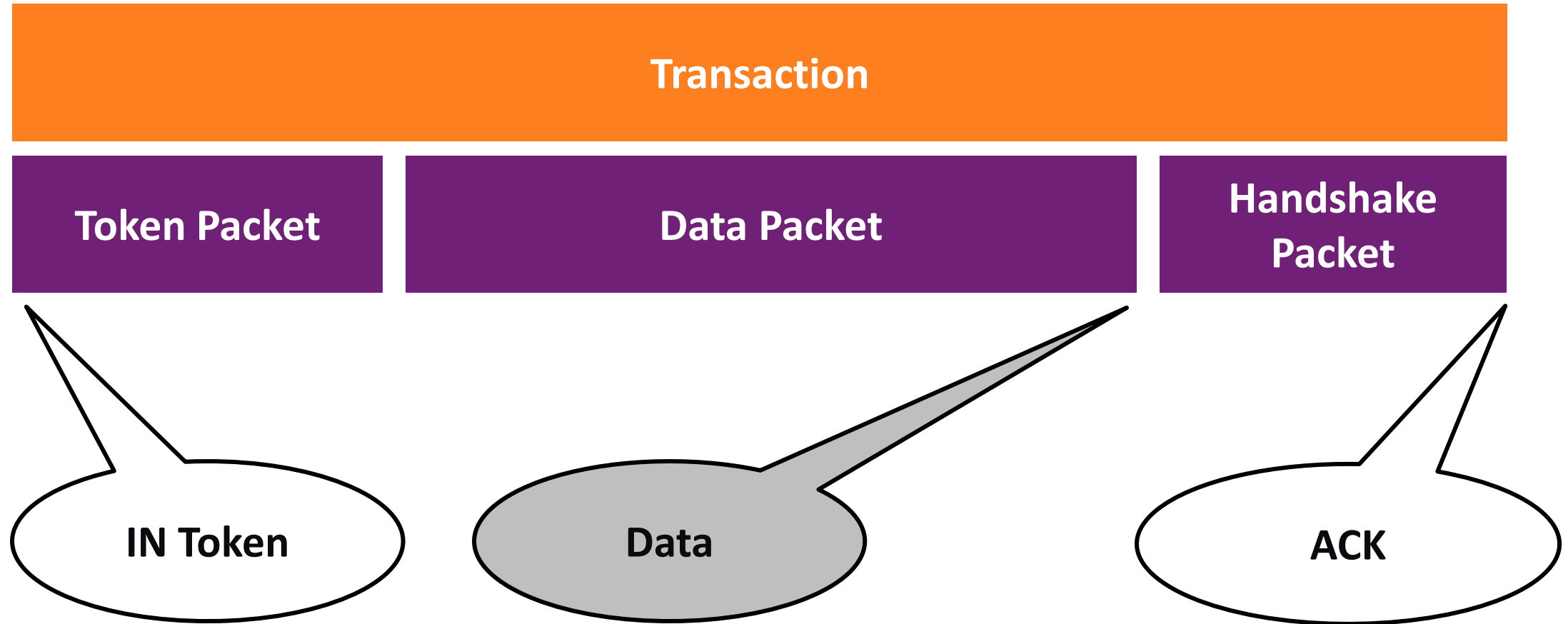
Data Packet

**Handshake
Packet**

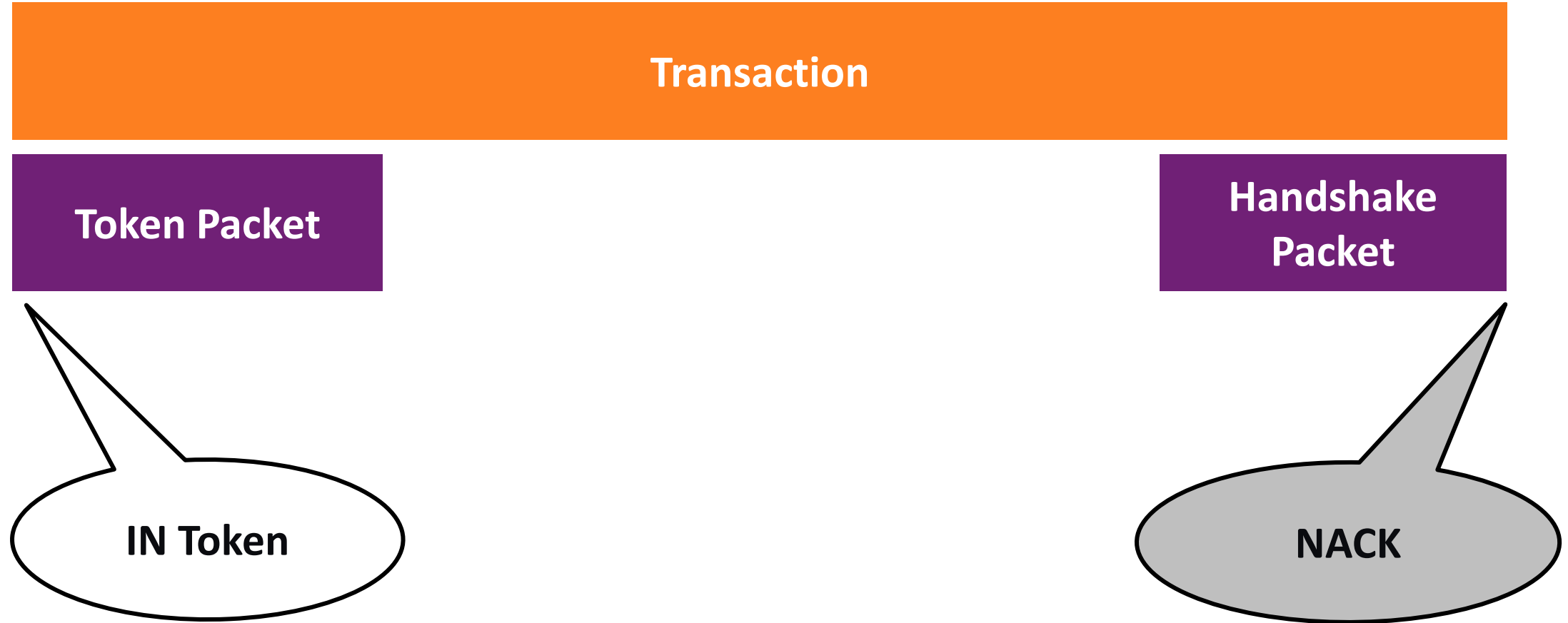
ACK
NACK
STALL
NYET
ERR

ACK : Accept data
NACK : Can't accept data
STALL : Request is not supported
NYET : Not Ready
ERR : Split Transaction Error (for HUB)

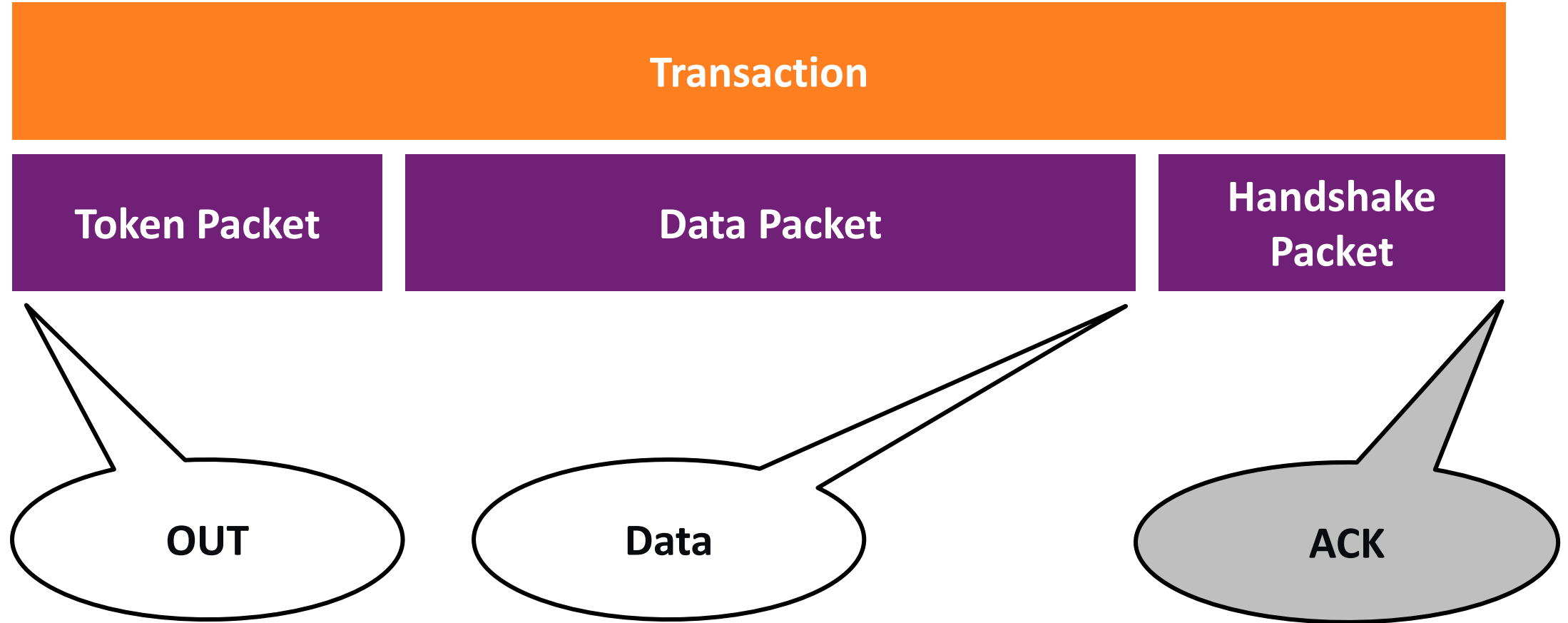
USB Transactions - Require Data, IN - ACK



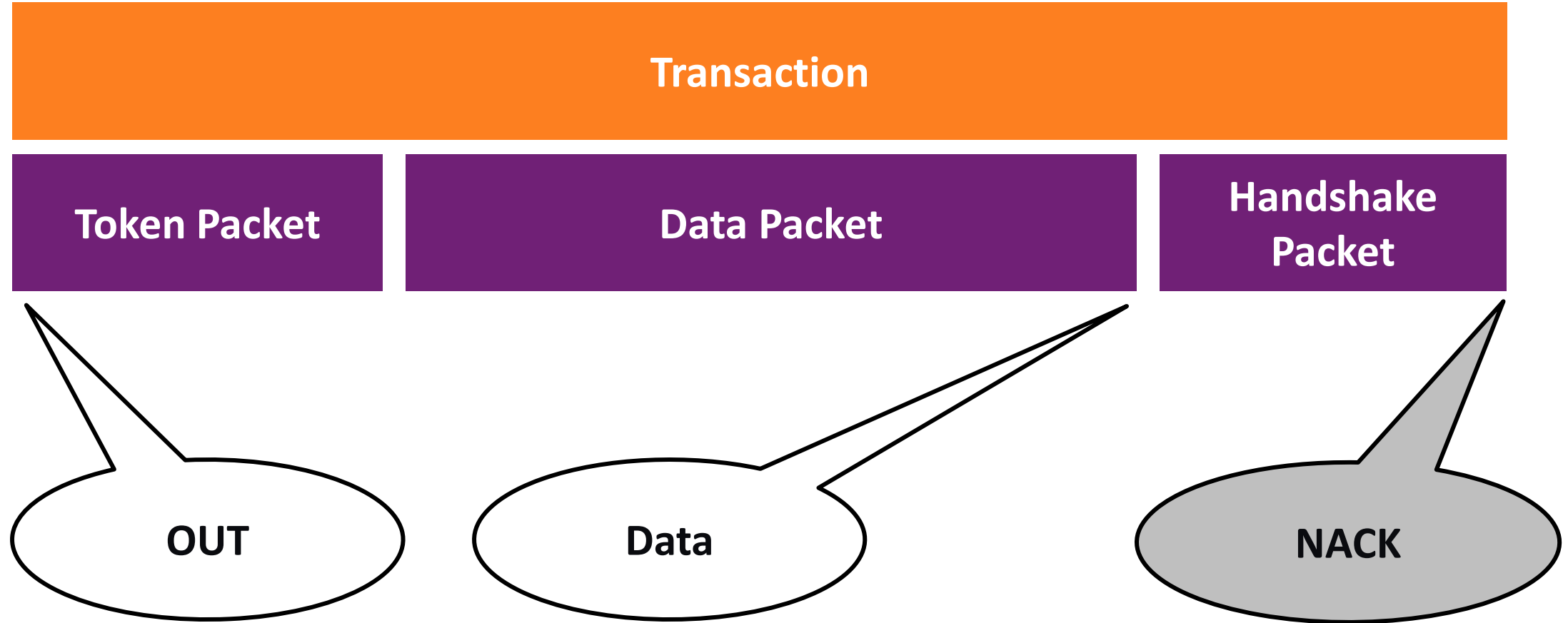
USB Transactions - Require Data, IN - NACK



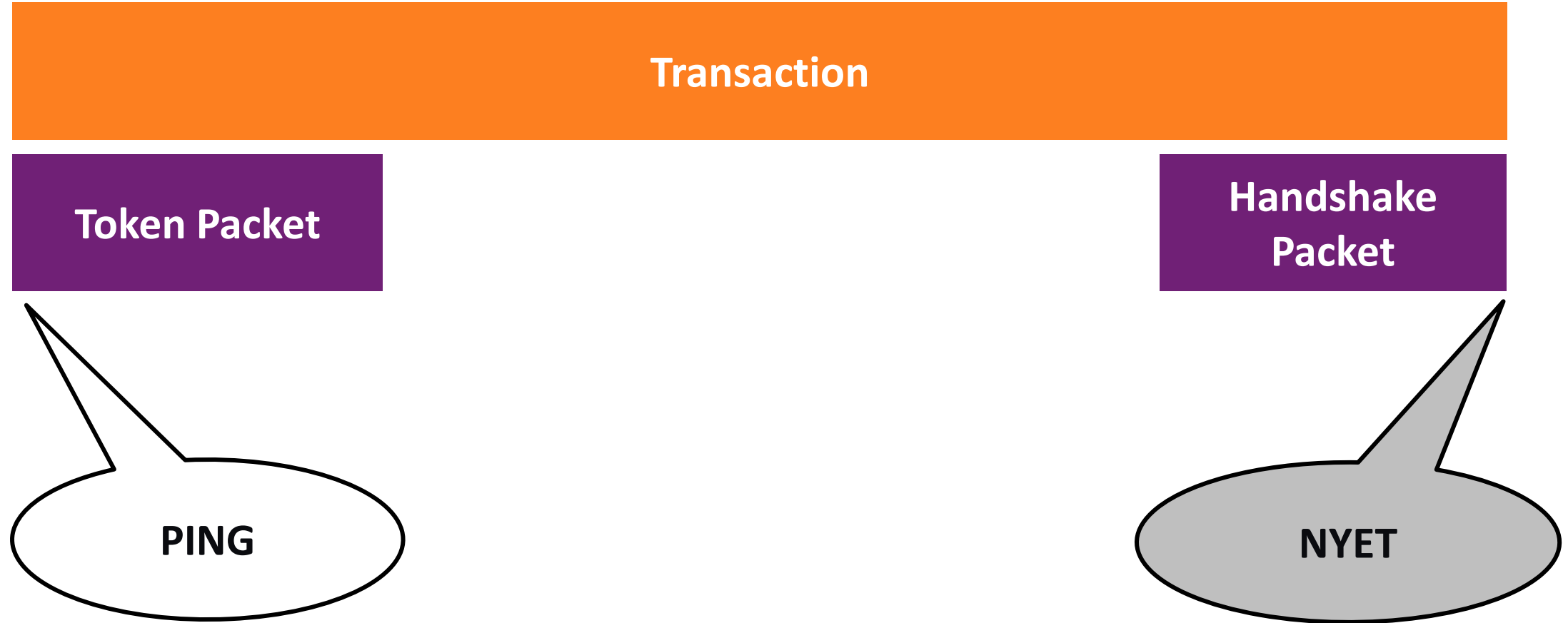
USB Transactions - Send Data, OUT - ACK



USB Transactions - Send Data, OUT - NACK

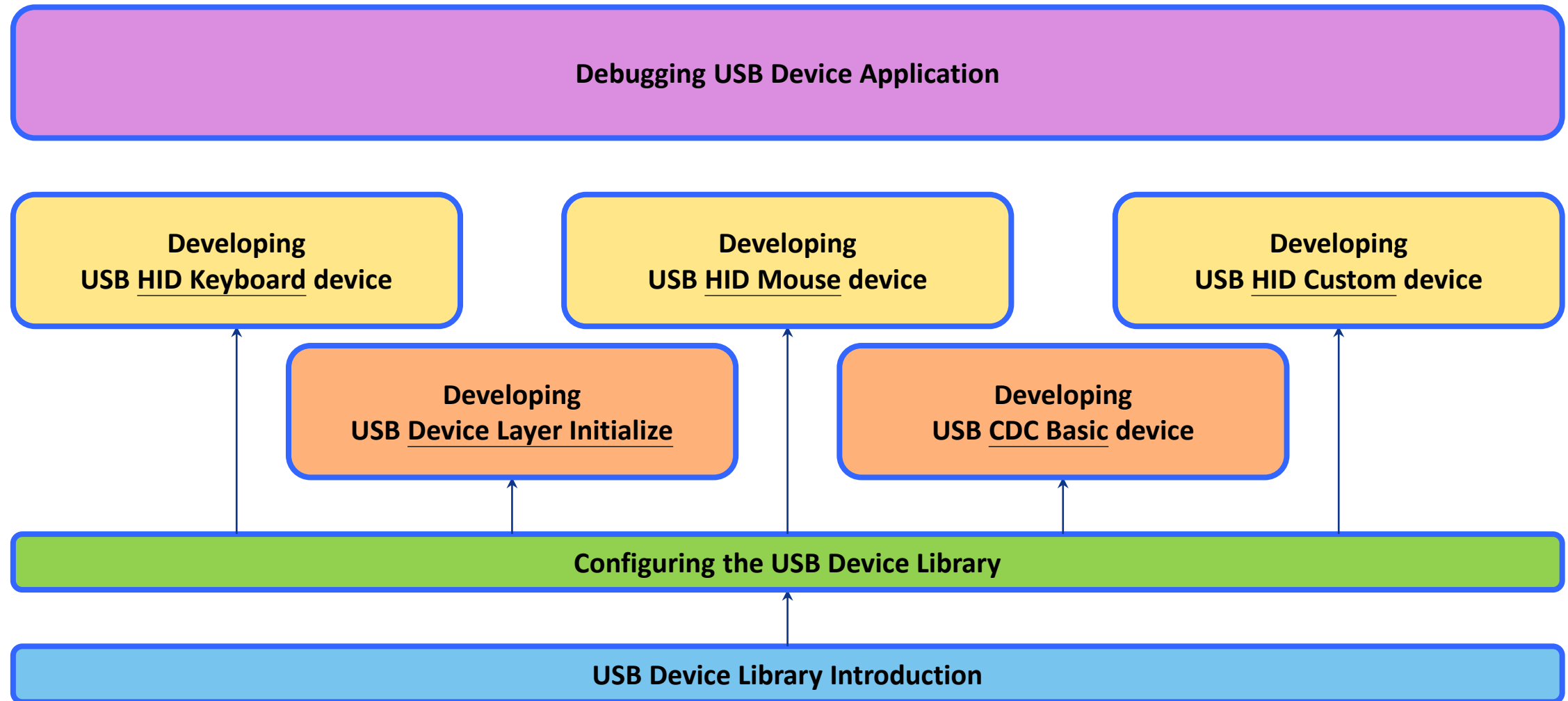


USB Transactions - Send Data, PING - NYET

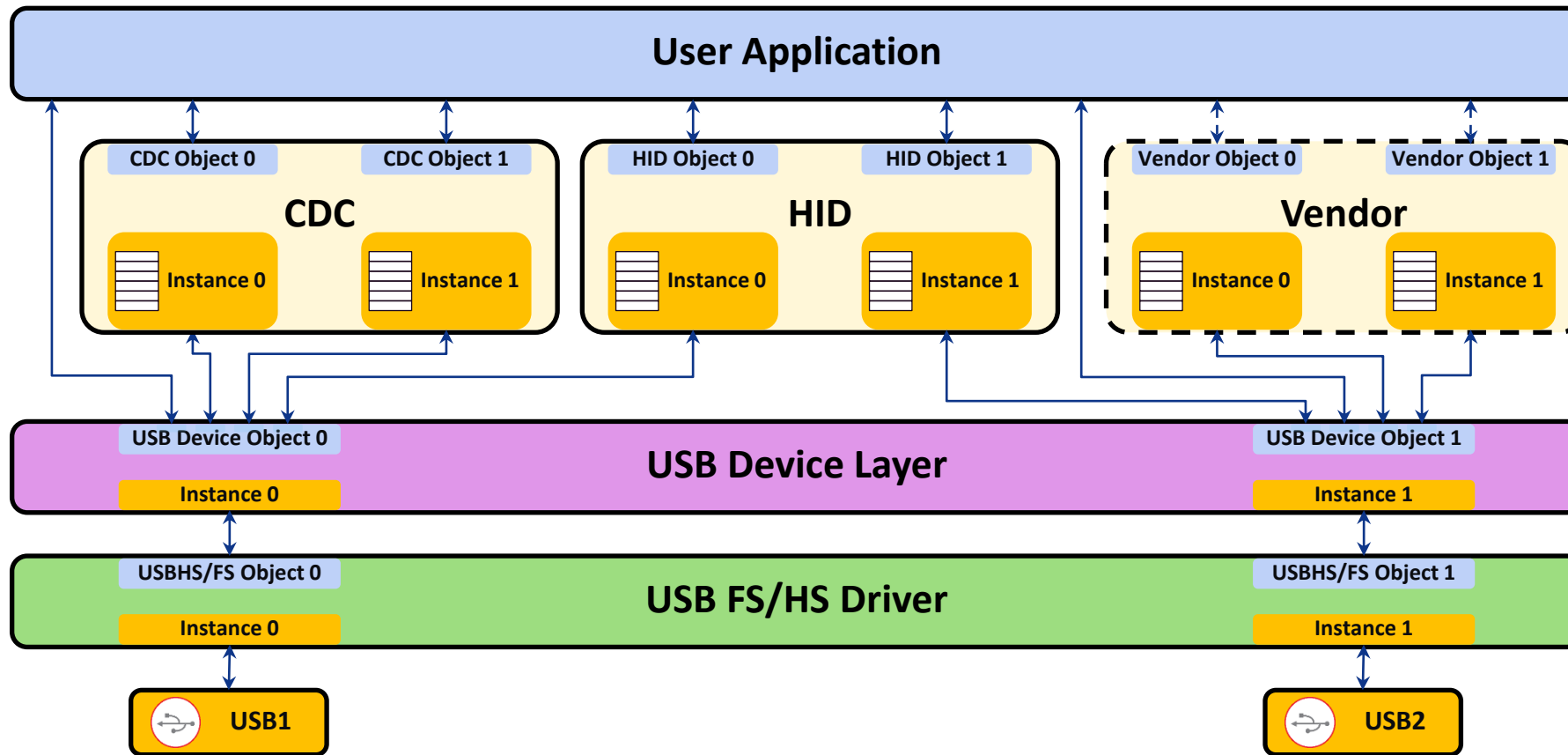


USB Device Library (USB Device Architecture)

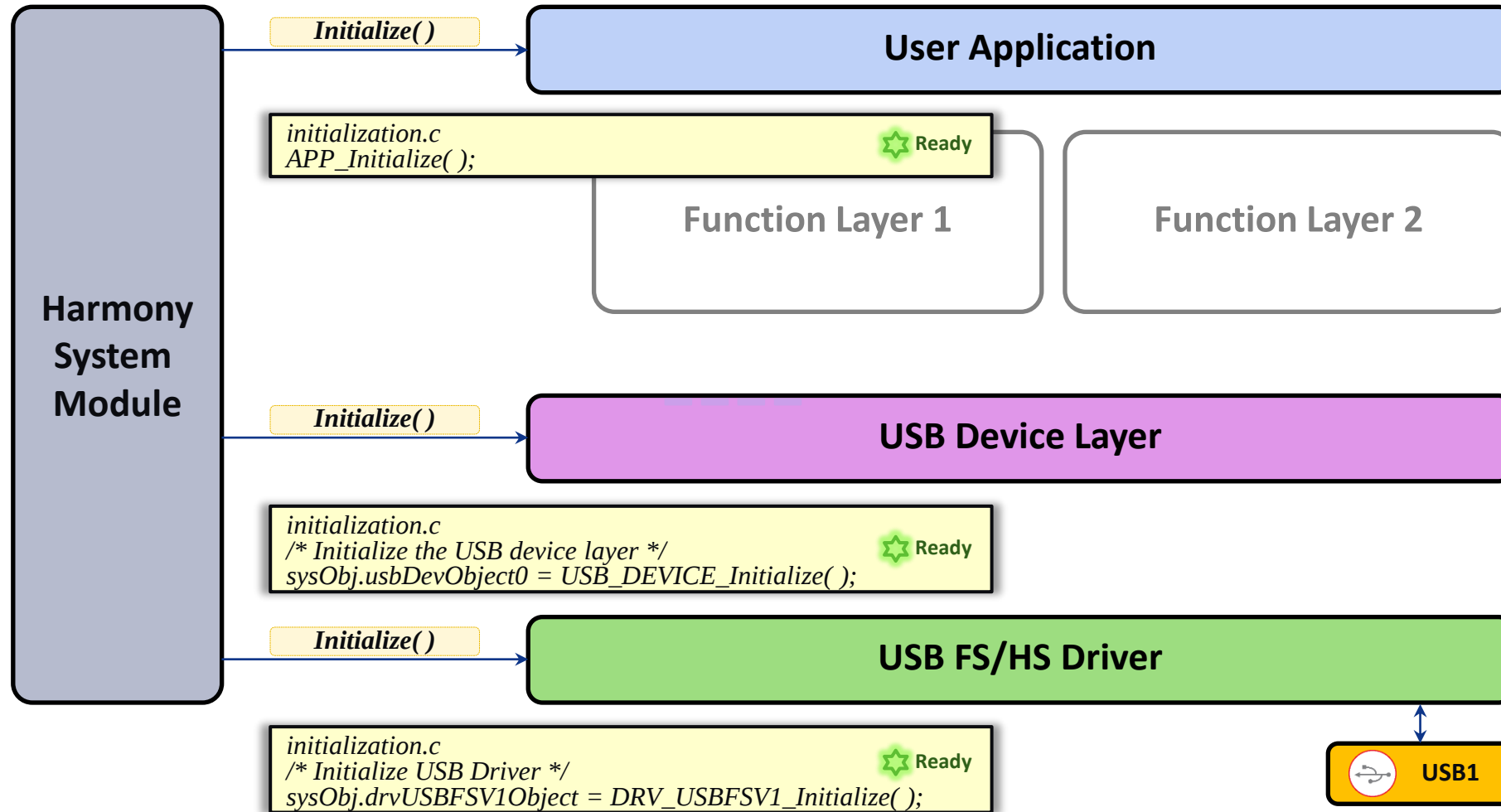
Course Structure for USB Device Library



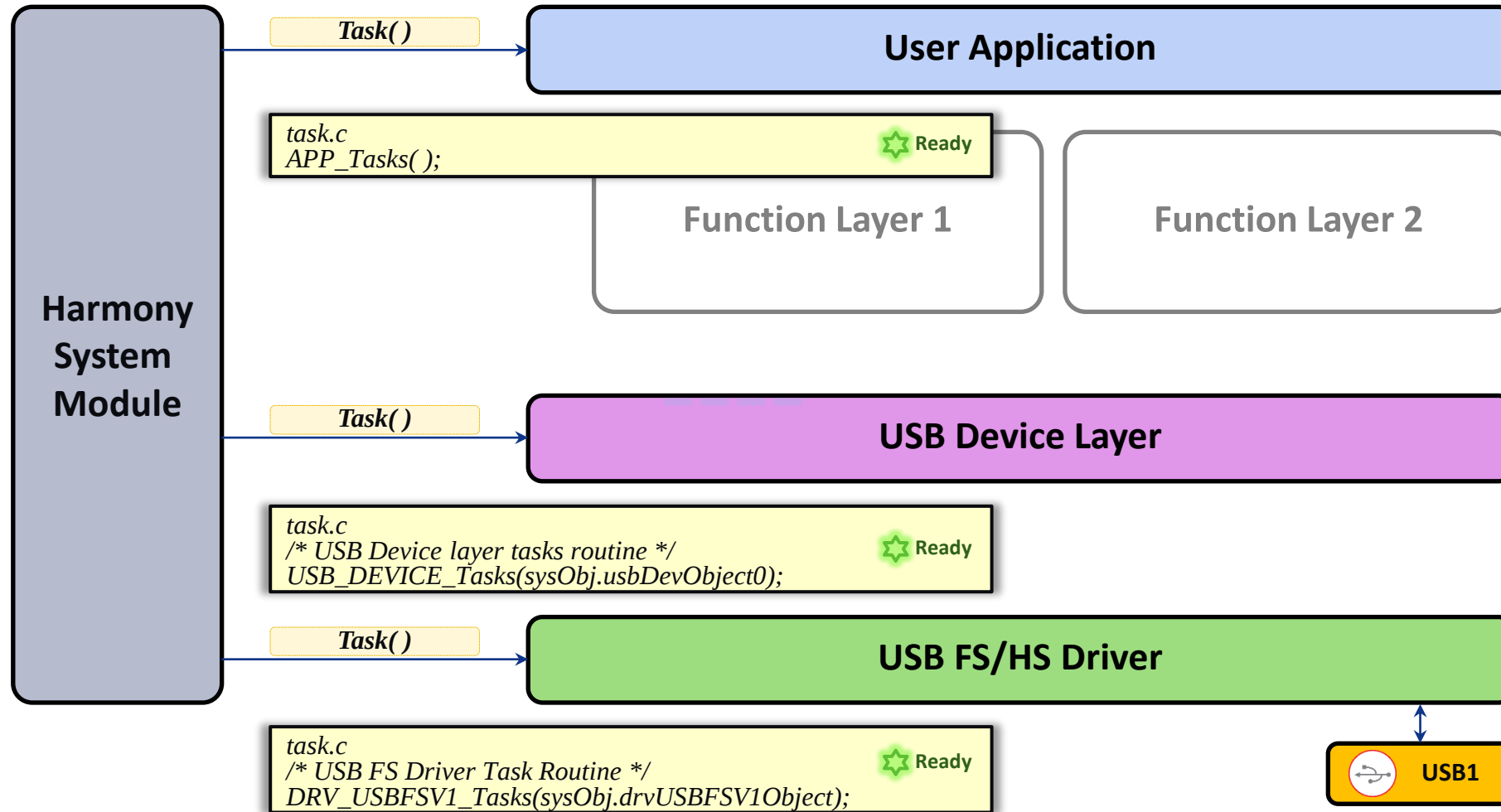
USB Device Library Architecture



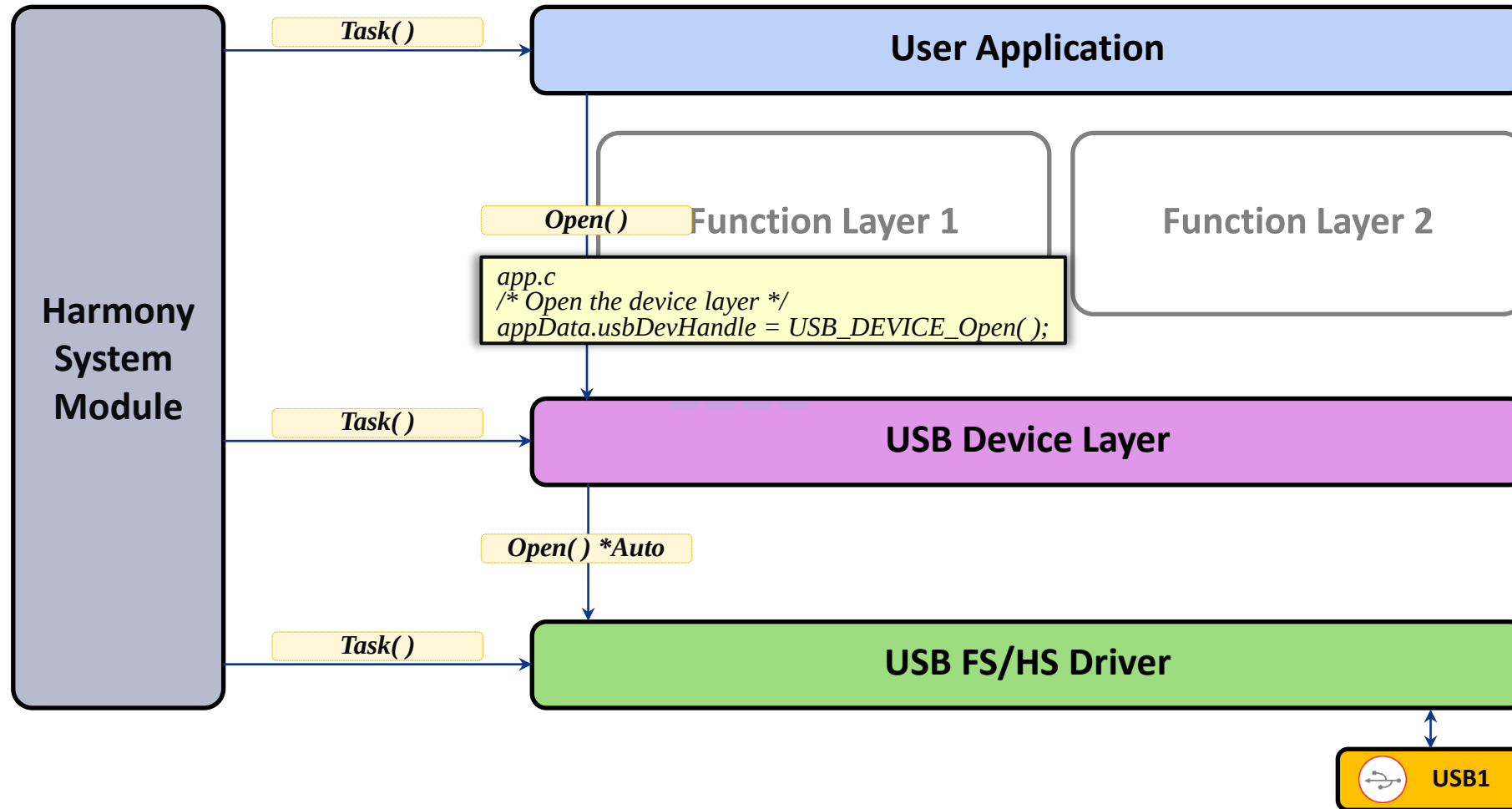
USB Stack Abstraction Model - Initialize & Open



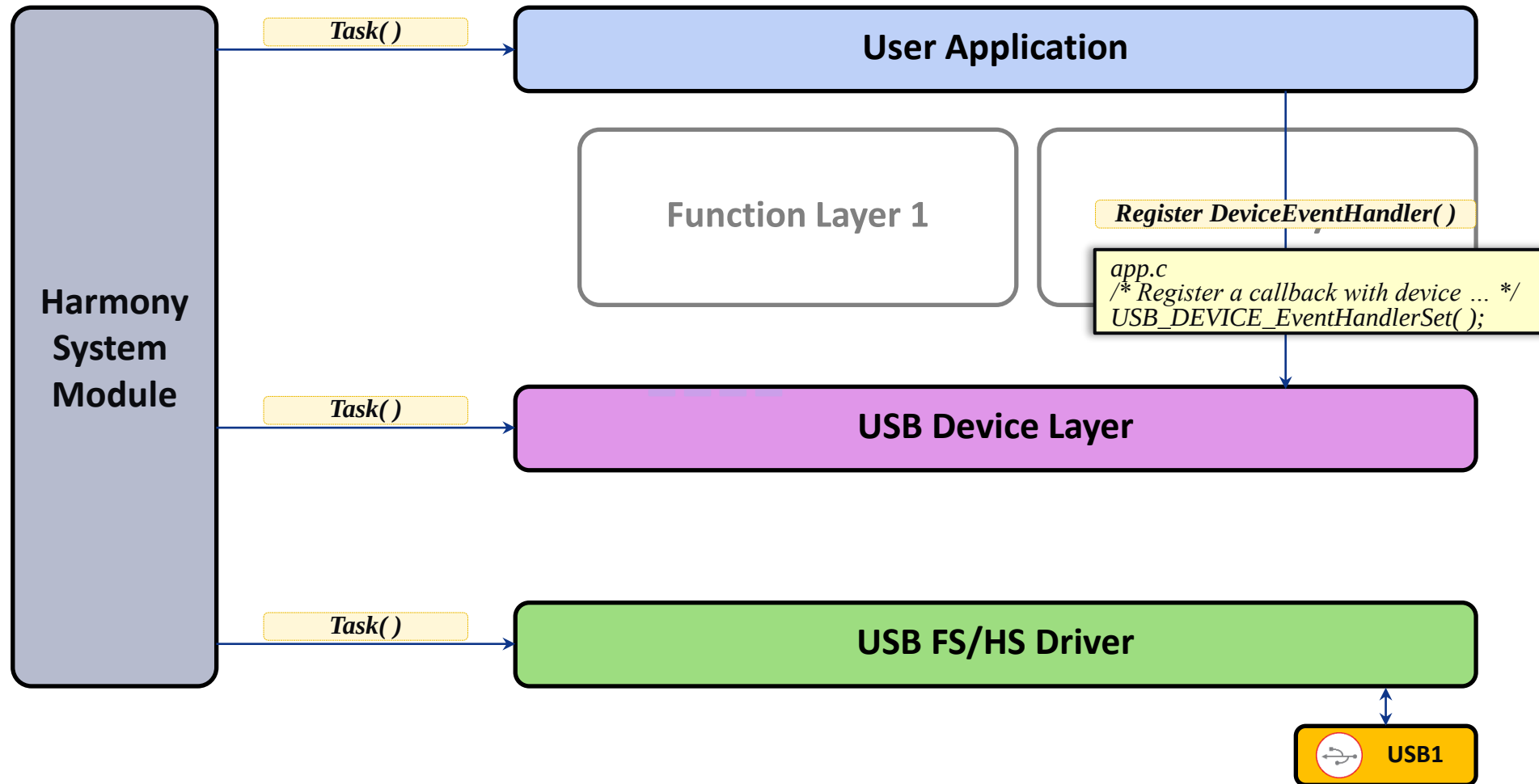
USB Stack Abstraction Model - Initialize & Open



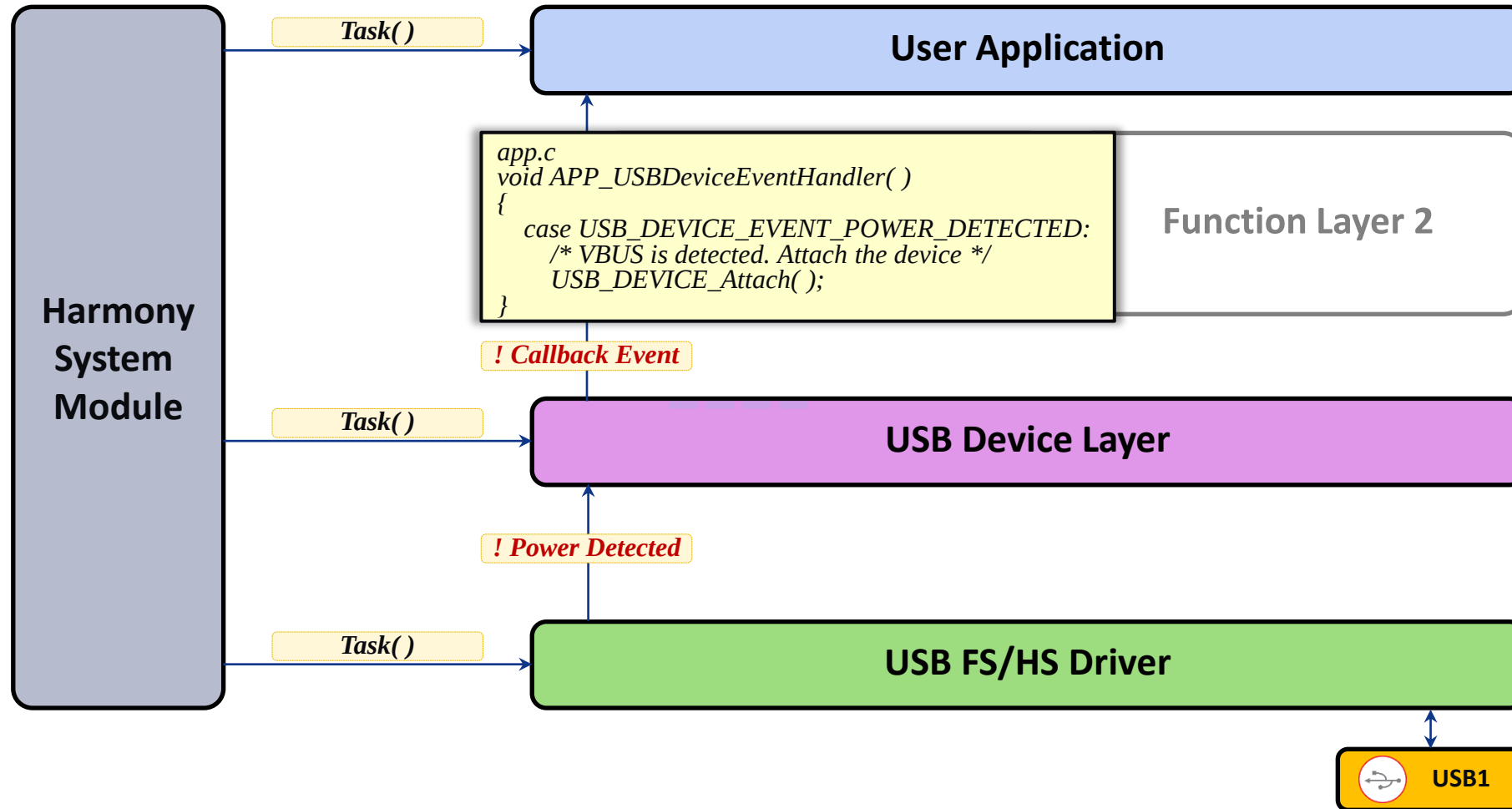
USB Stack Abstraction Model - Initialize & Open



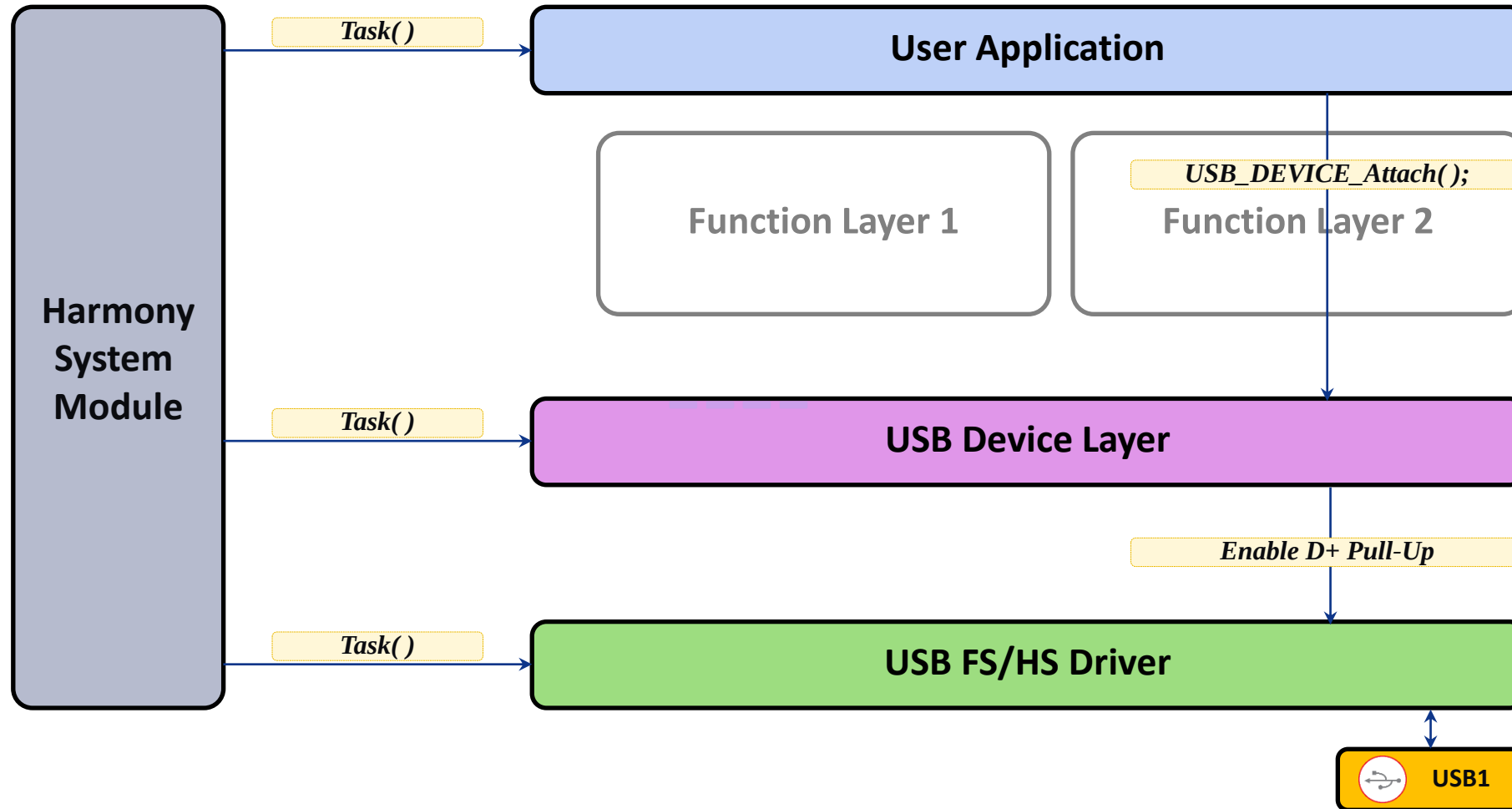
USB Stack Abstraction Model - Register Device Event Handler



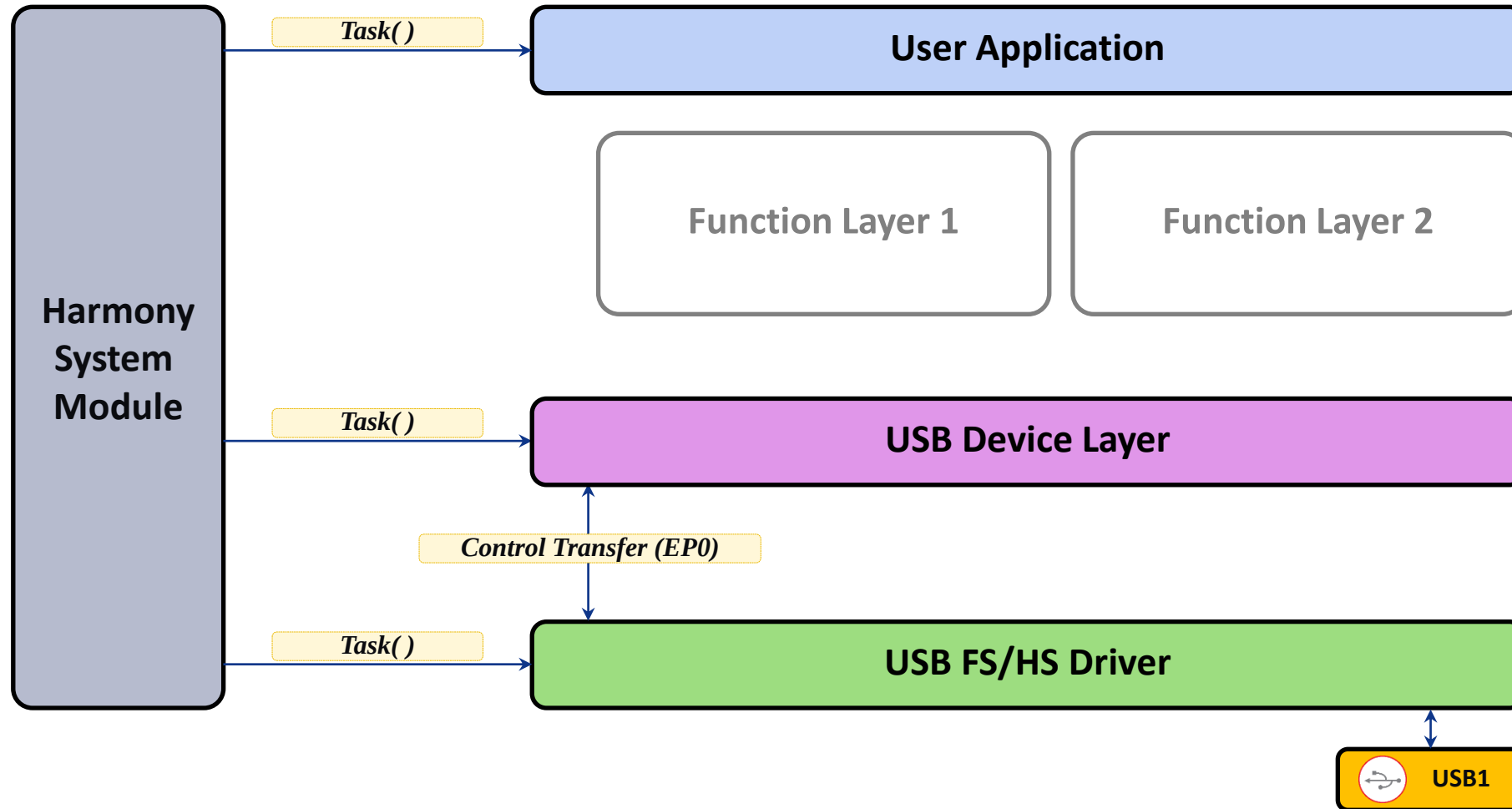
USB Stack Abstraction Model - Powered



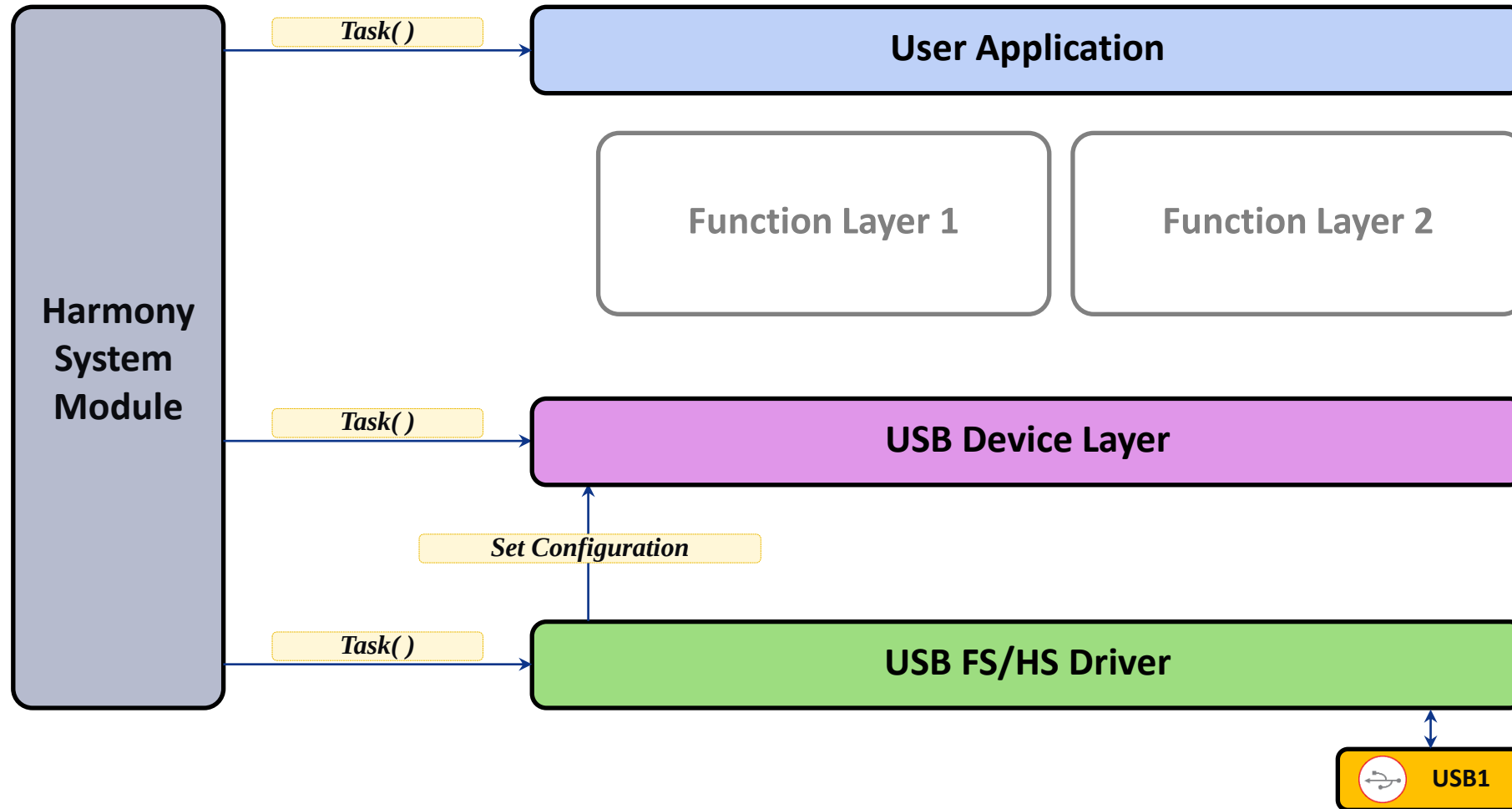
USB Stack Abstraction Model - Attach



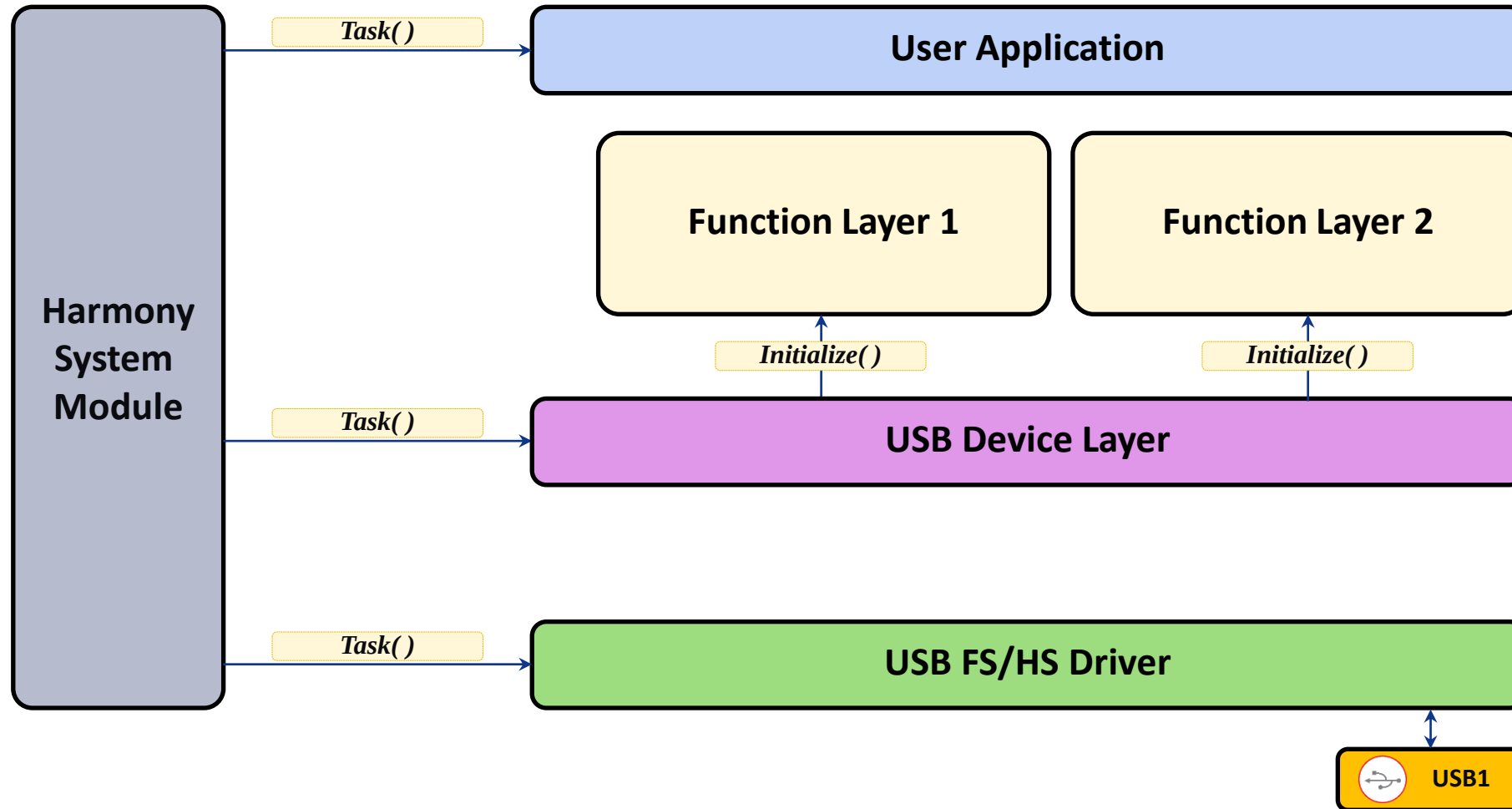
USB Stack Abstraction Model – Control Transfer



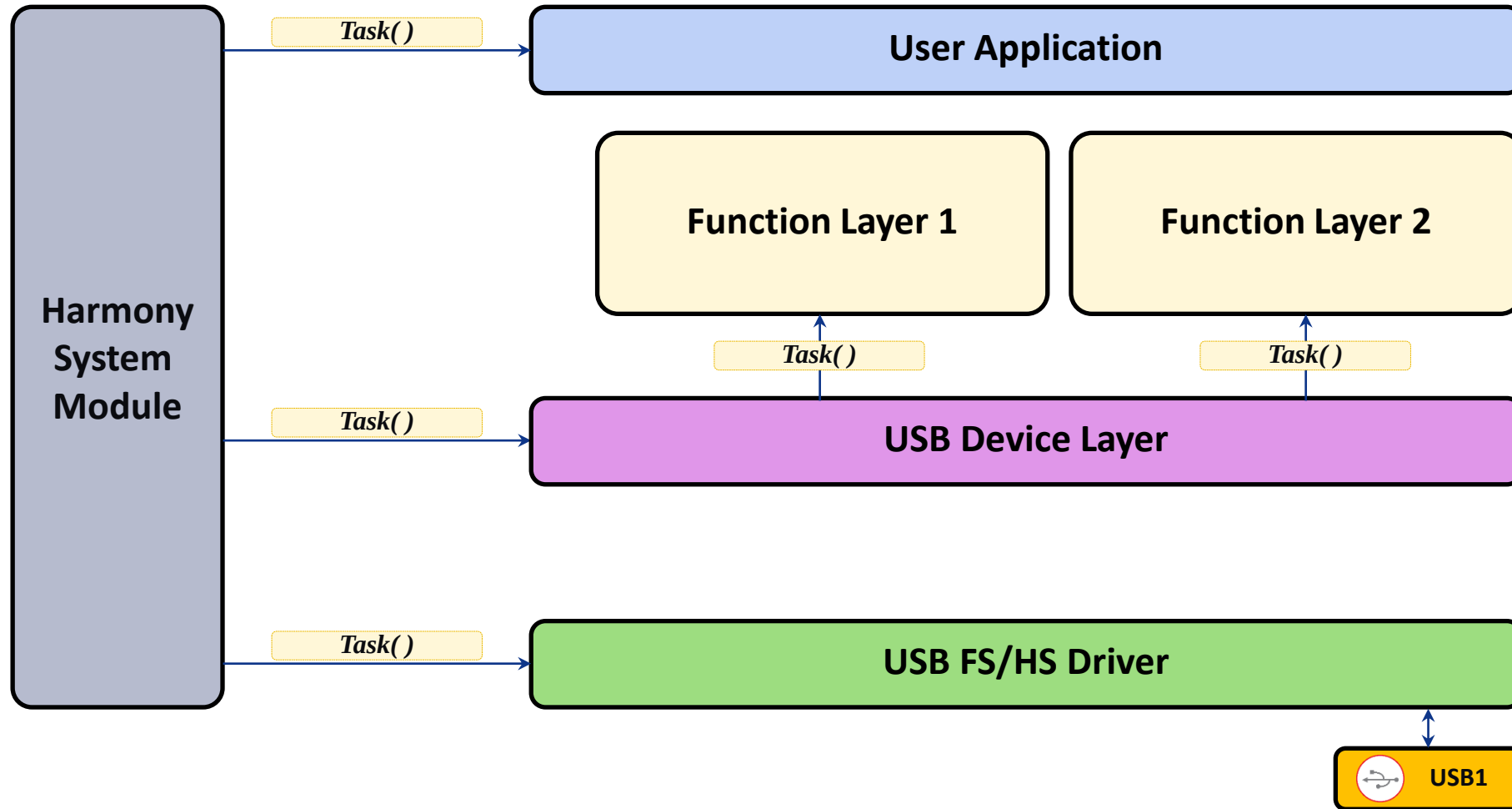
USB Stack Abstraction Model - Set Configuration



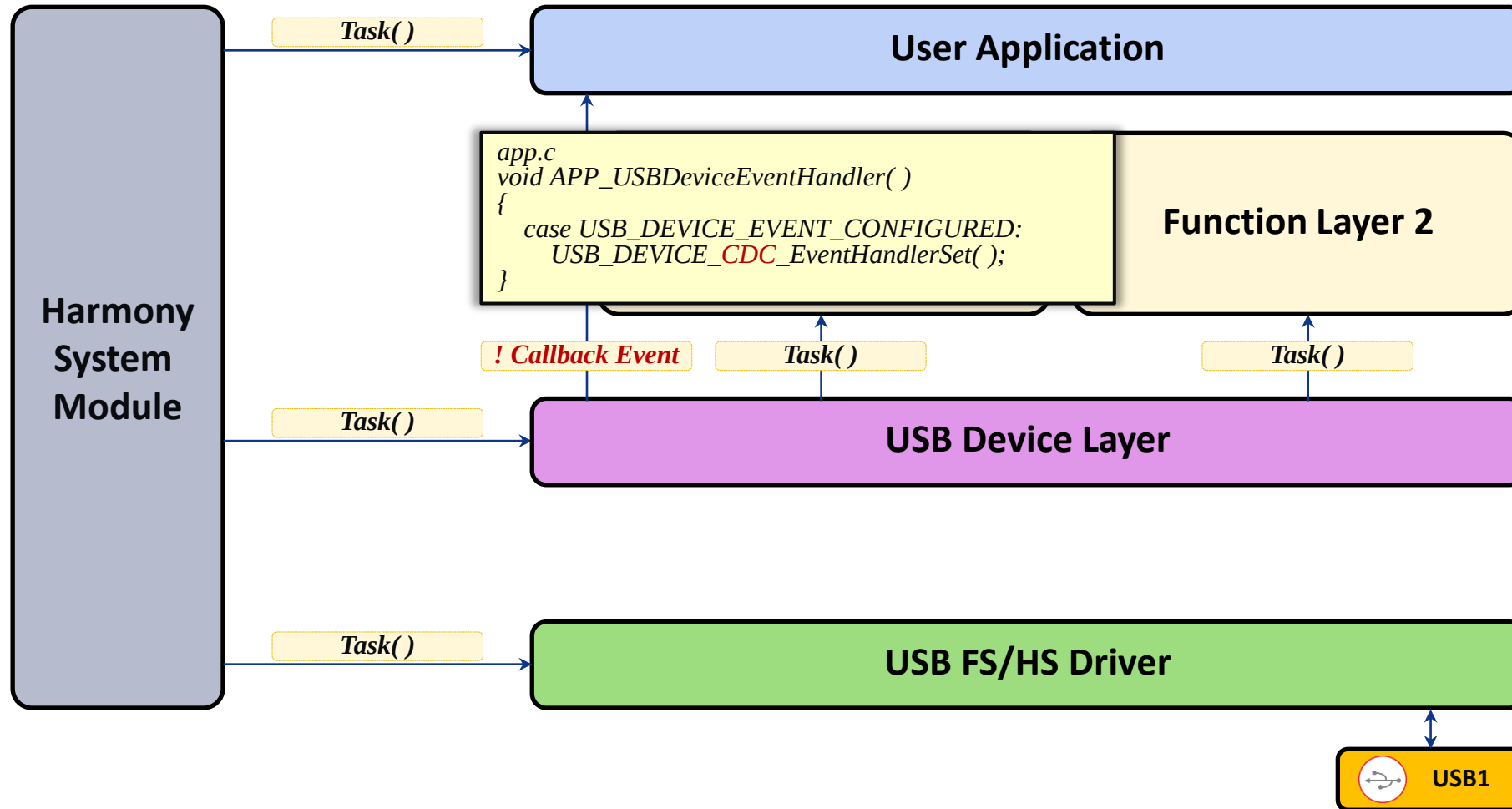
USB Stack Abstraction Model - Function Initialize & Open



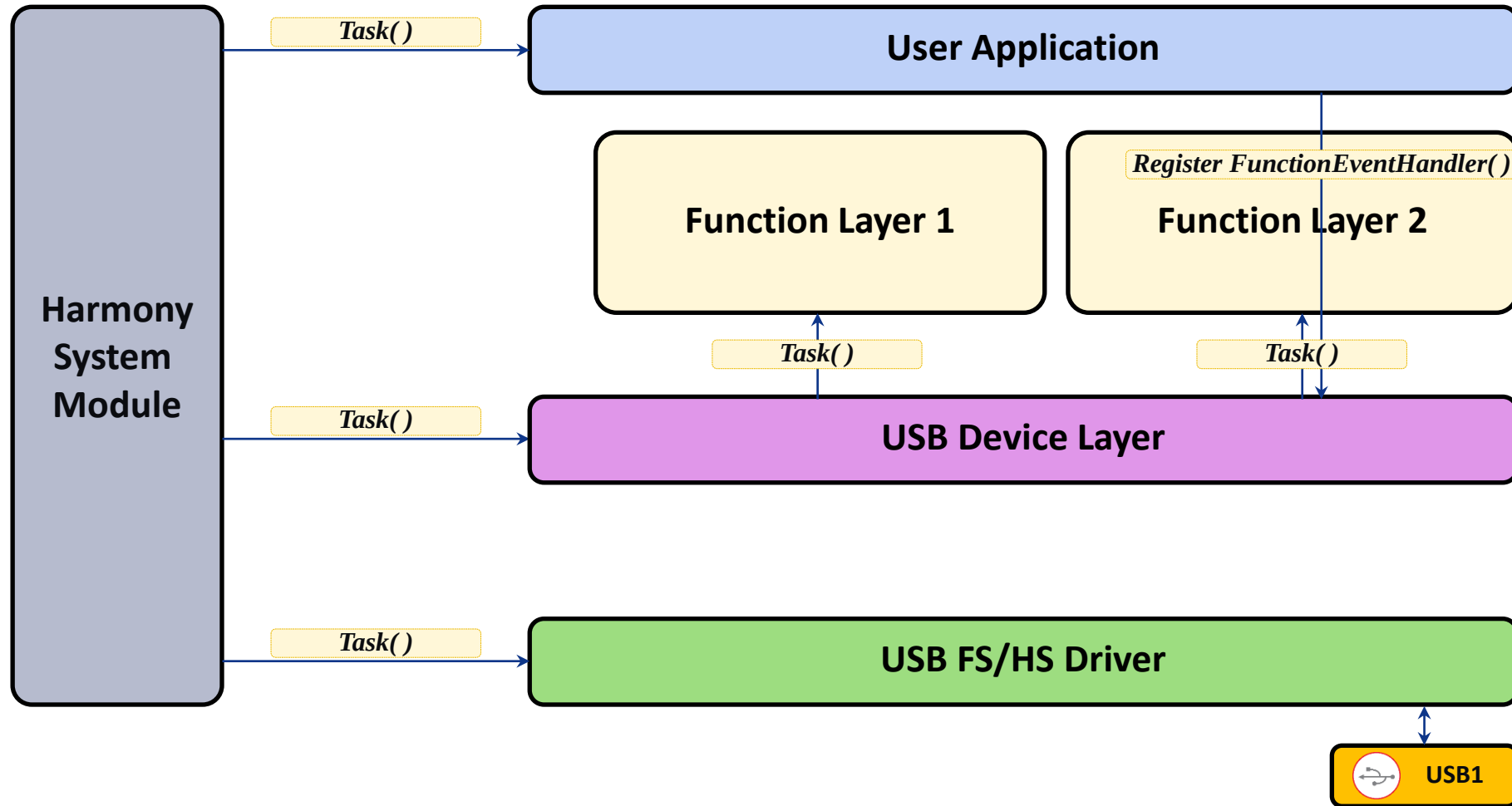
USB Stack Abstraction Model - Function Initialize & Open



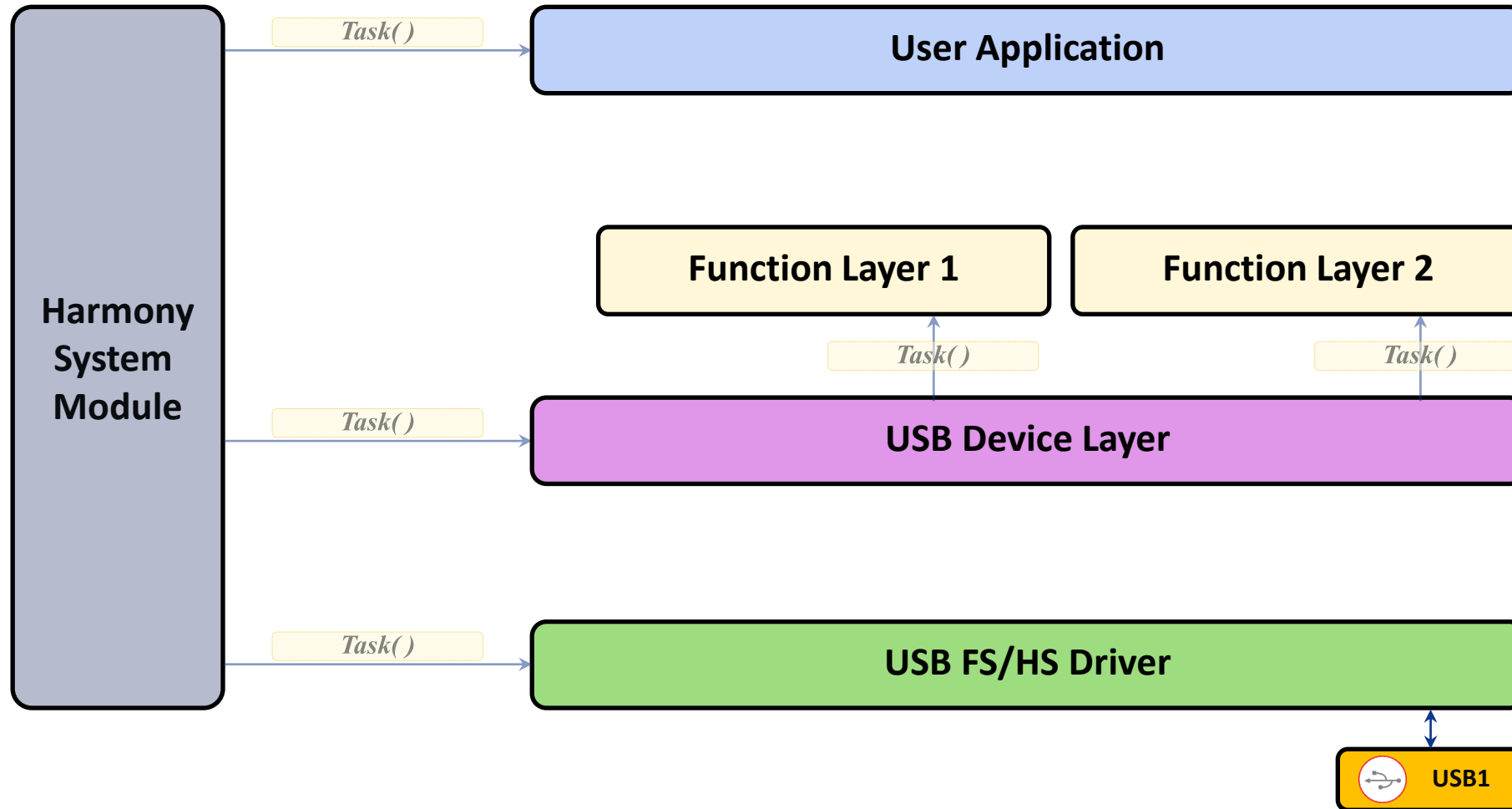
USB Stack Abstraction Model - Register Function Event Handler



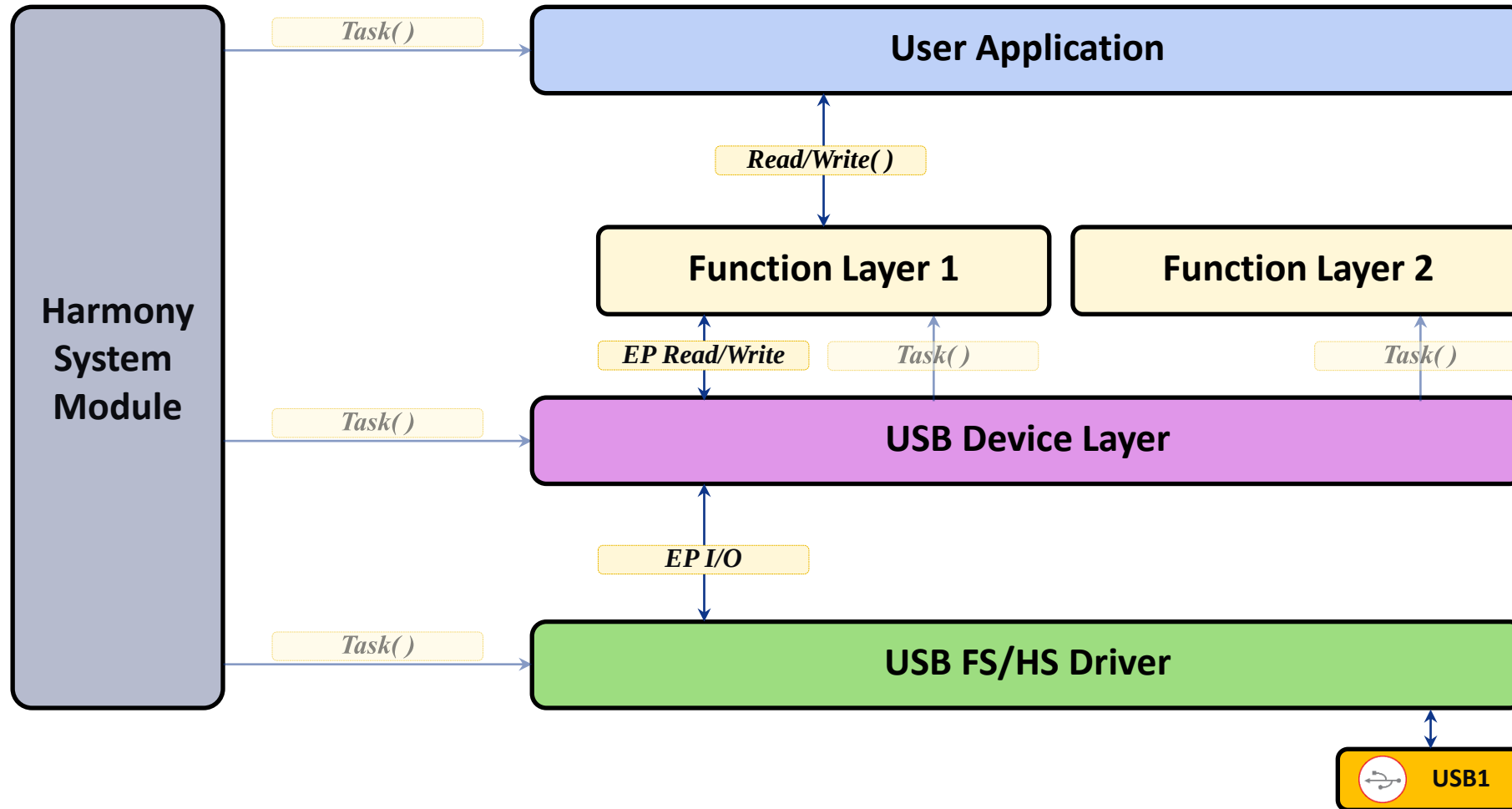
USB Stack Abstraction Model - Register Function Event Handler



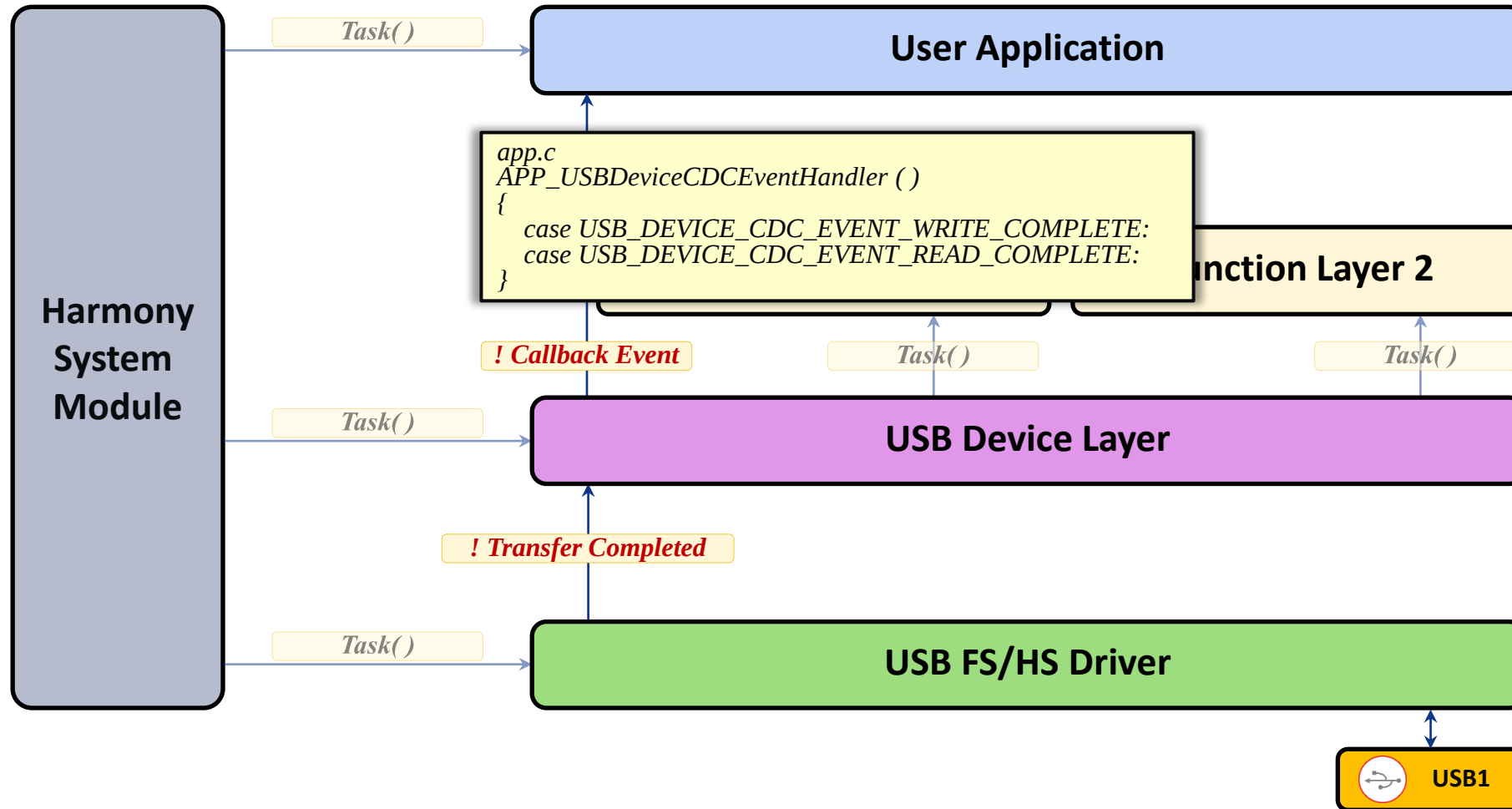
USB Stack Abstraction Model - Function Service Ready



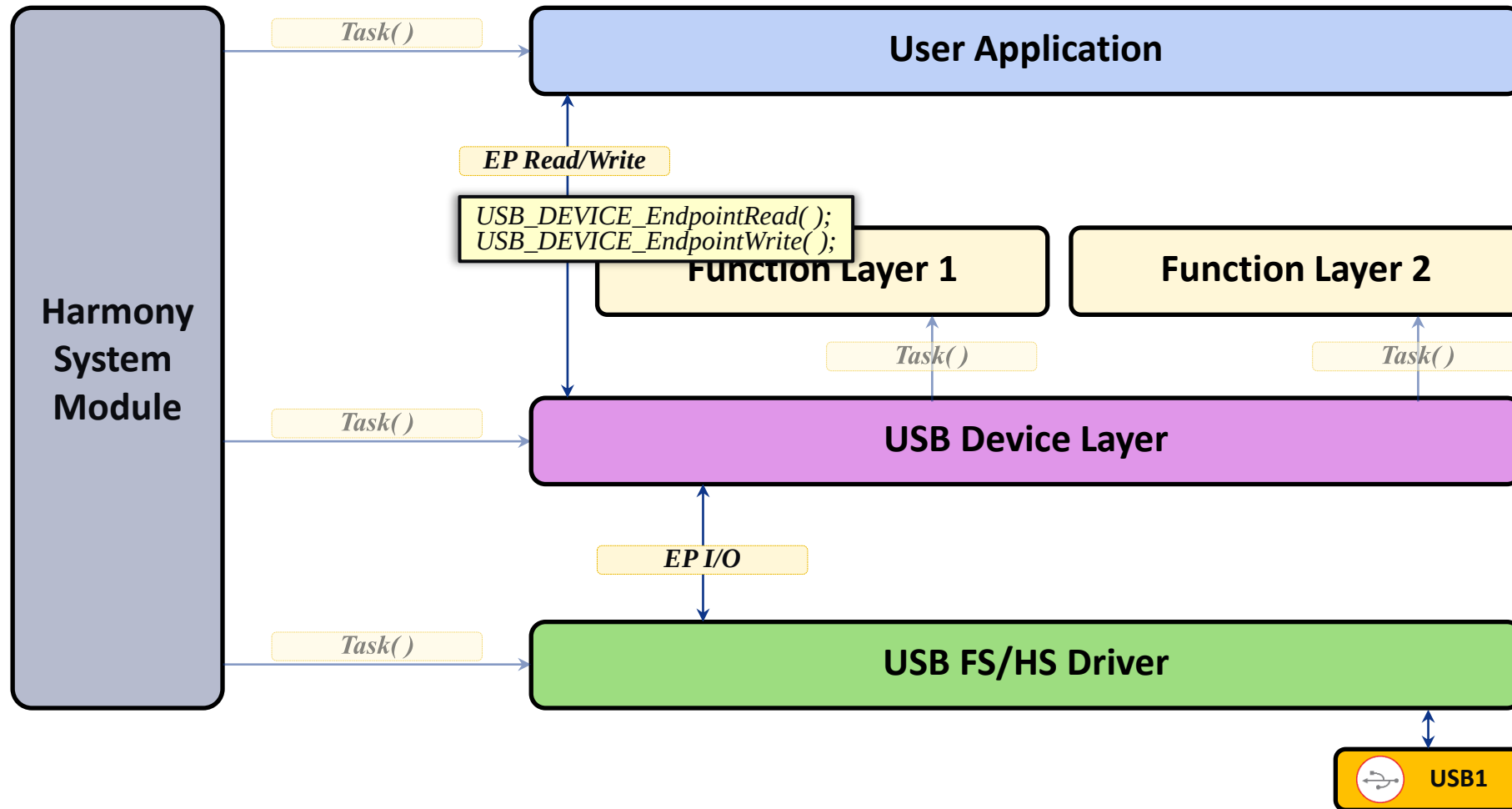
USB Stack Abstraction Model - Functionally Read/Write



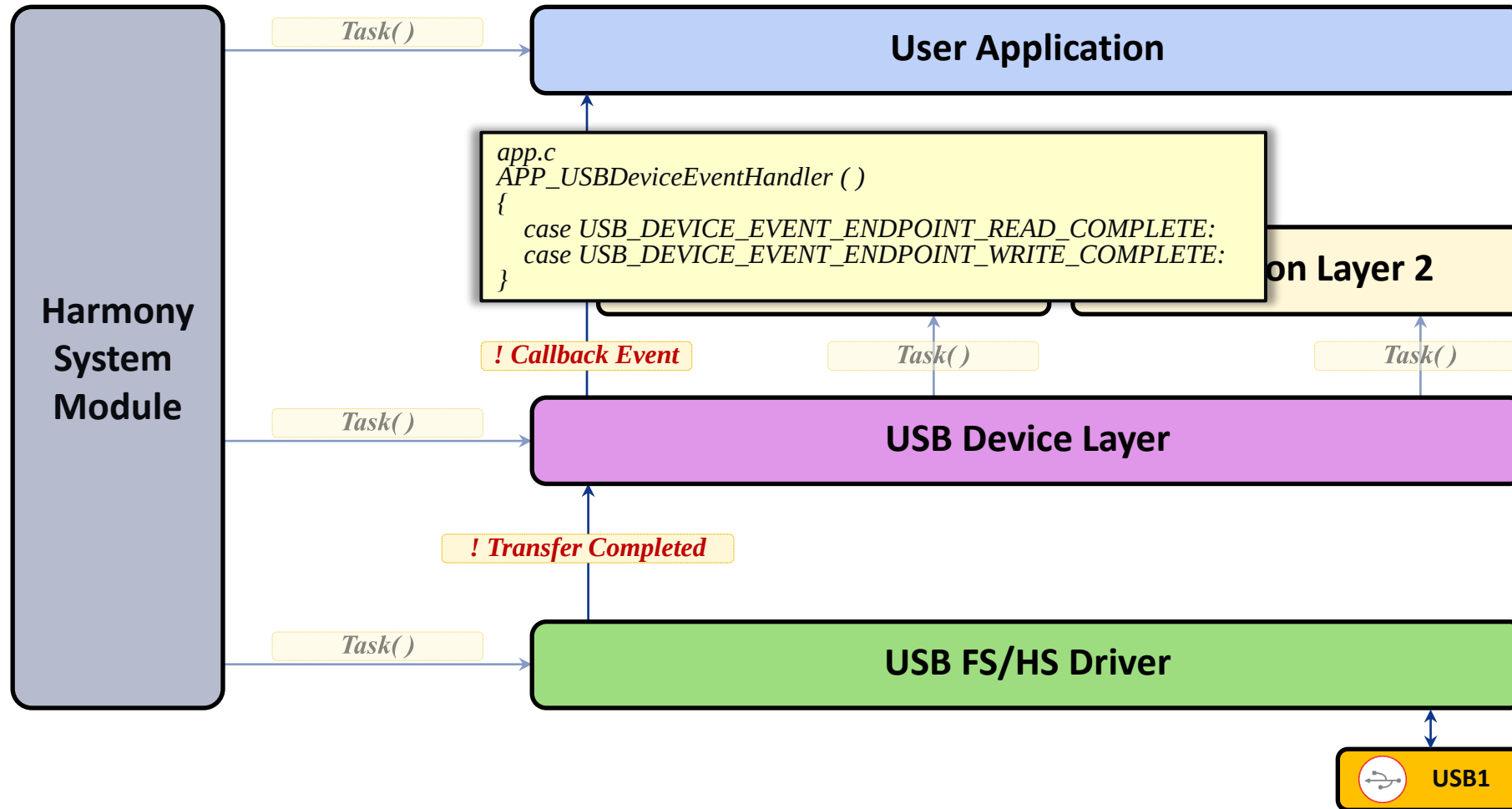
USB Stack Abstraction Model - Functionally Read/Write



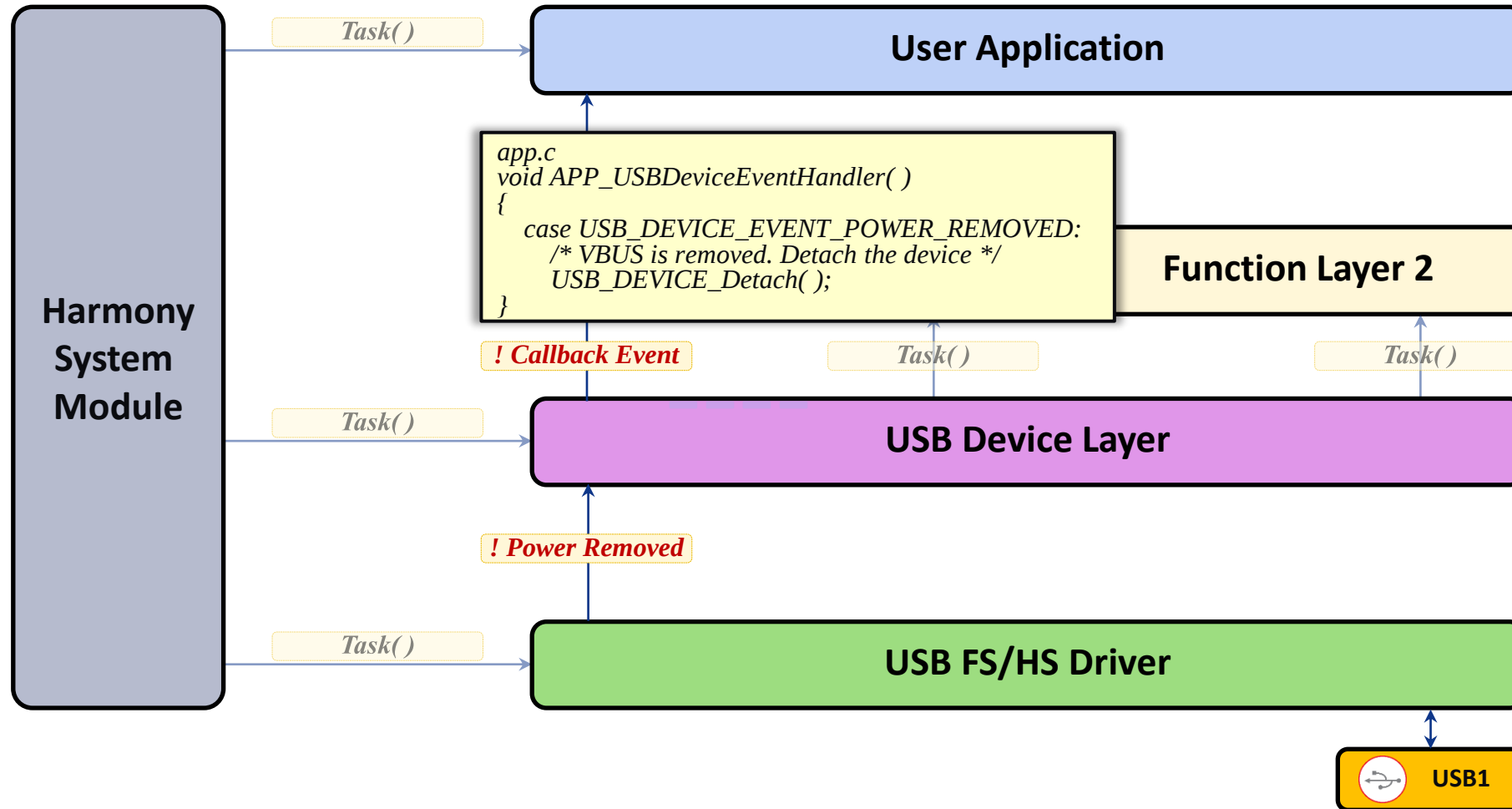
USB Stack Abstraction Model - Device Layer Read/Write



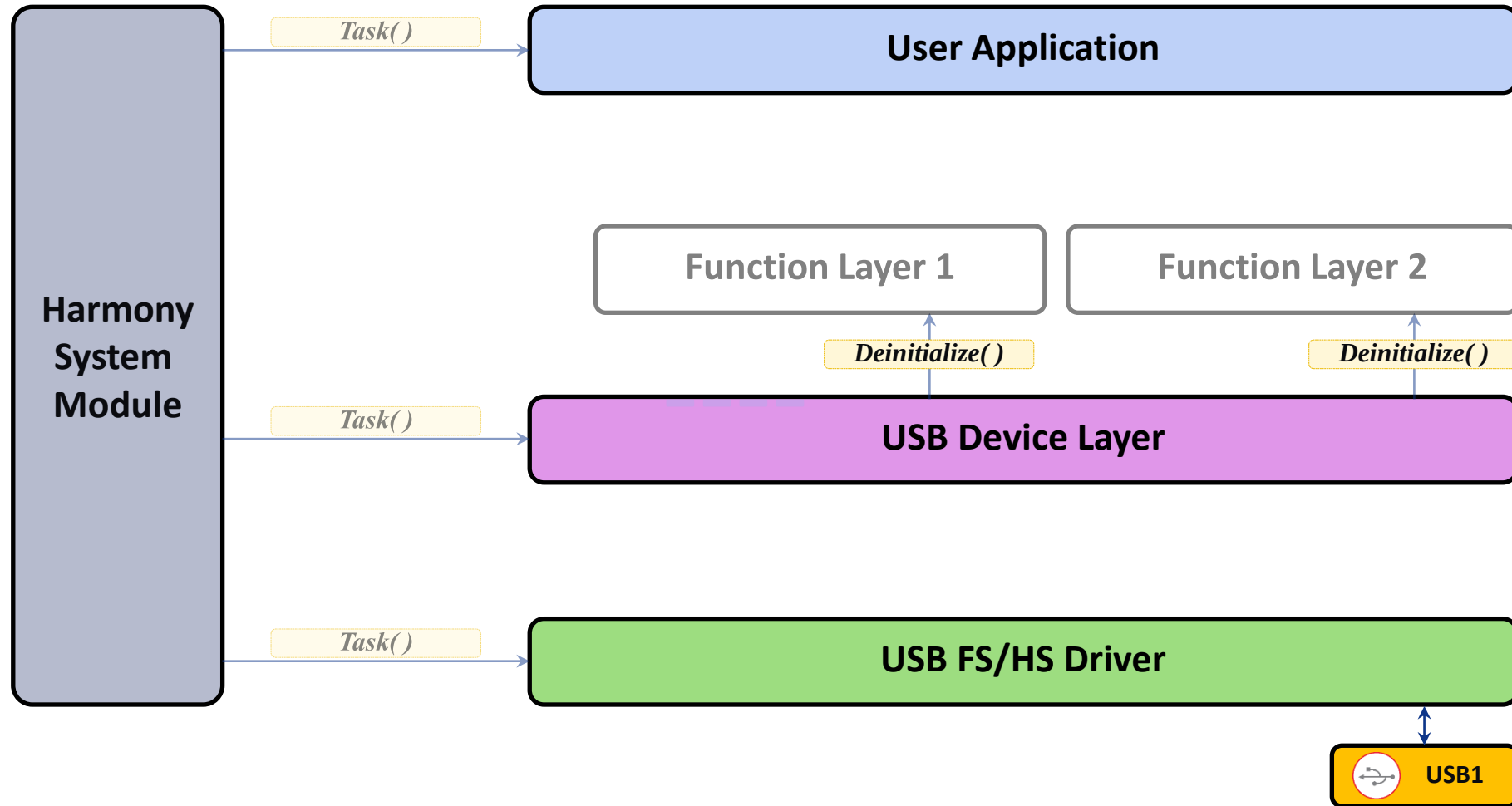
USB Stack Abstraction Model - Device Layer Read/Write



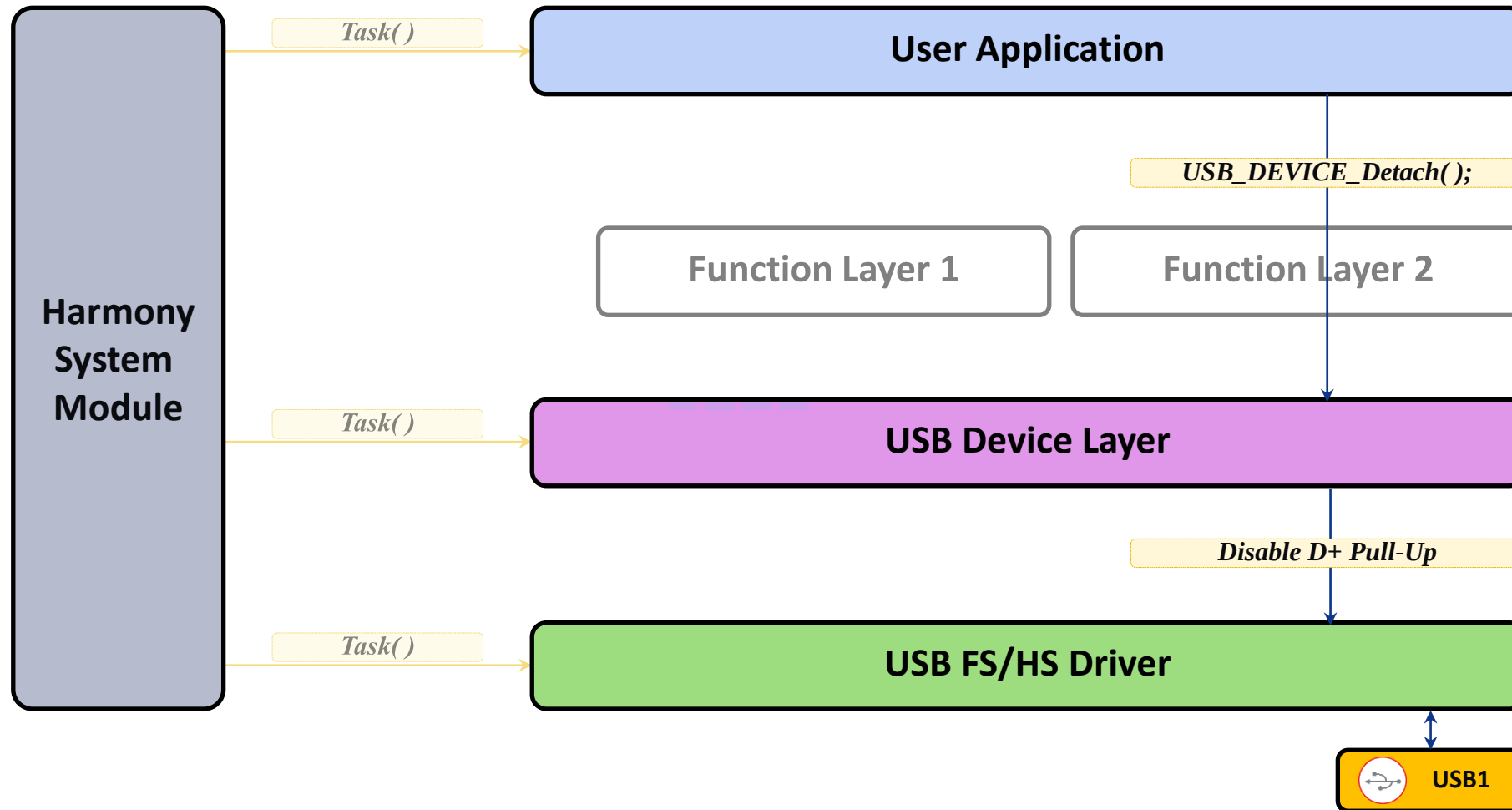
USB Stack Abstraction Model - Power Removed



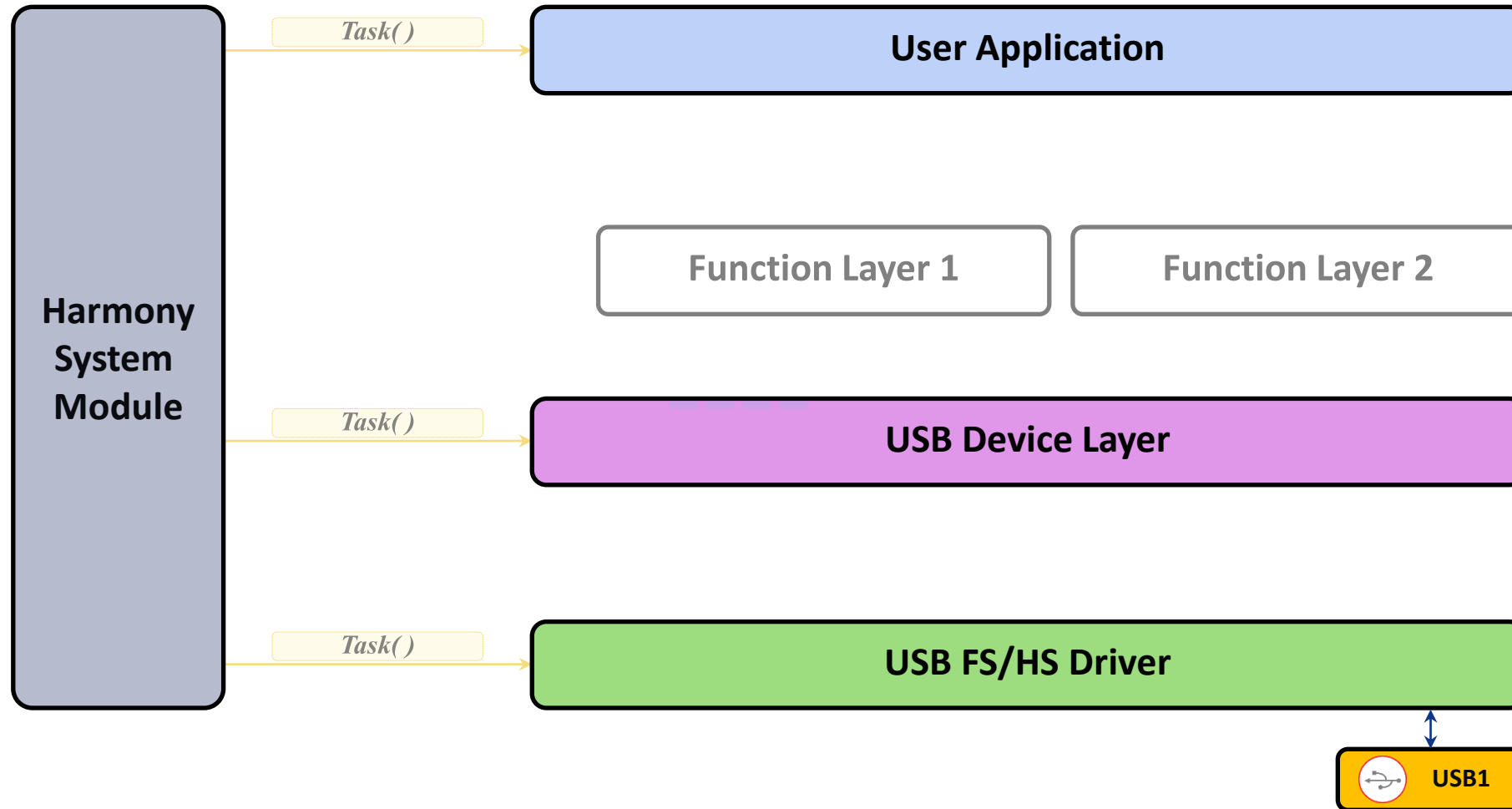
USB Stack Abstraction Model - Function Deinitialize



USB Stack Abstraction Model - Detach



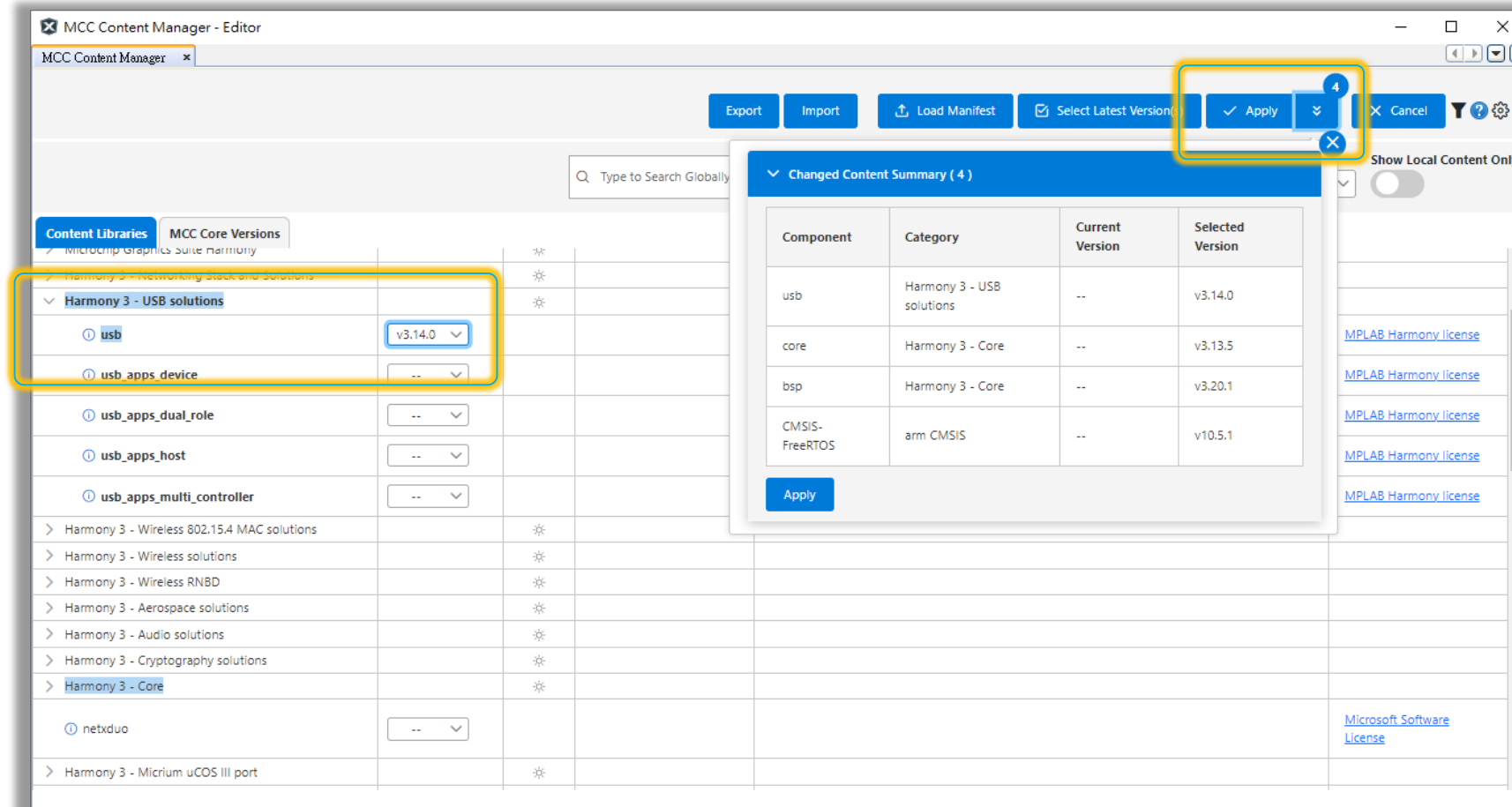
USB Stack Abstraction Model - Disconnection



USB Libraries Maintain at Microchip Harmony 3

USB Libraries at MH3

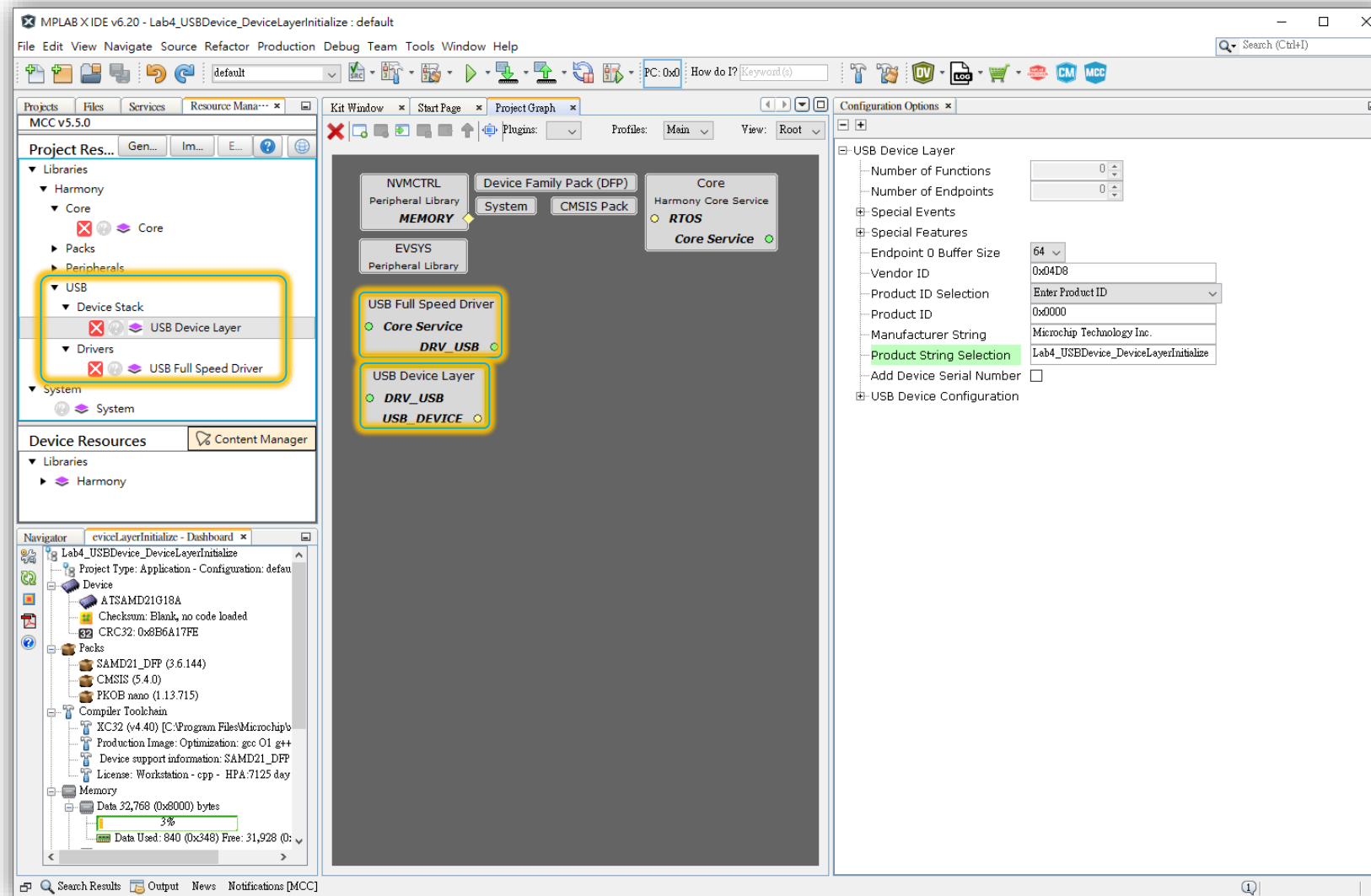
- **USB Libraries must be preparation, before develop the USB application.**
- **Select suitable version of “Harmony 3 - USB solutions” at MCC Content Manager & Apply it.**



Harmony USB Device Device Layer Configurator

Harmony USB Device - Device Layer Configurator

Add USB FS/HS Driver & USB Device Layer in MHC



Harmony USB Device - Device Layer Configurator

Configure USB_DM(D-) and USB_DP(D+) at Pin Table

Pin Table - Editor

Package: TQFP48

Module	Function	PA16	PA17	PA18	PA19	PA20	PA21	PA22	PA23	USB_DM	USB_DP	ONDI0	VDDIO	PB22	PB23	PA27	RESET_...	PA28	ONDI0	VDDCOR
TCC0	TCC0_W03																			
	TCC0_W04																			
	TCC0_W05																			
	TCC0_W06																			
	TCC0_W07																			
TCC1	TCC1_W00																			
	TCC1_W01																			
	TCC1_W02																			
	TCC1_W03																			
TCC2	TCC2_W00																			
	TCC2_W01																			
USB	USB_DM																			
	USB_DP																			
	USB_SOF_1KHZ																			

Harmony USB Device - Device Layer Configurator

Confirm & Enable GCLK for USB Module

Peripheral Clock Configuration

Peripheral	Enable	Source	Peripheral Clock Frequency
SERCOM2_CORE	☐	GCLK0	--
SERCOM3_CORE	☐	GCLK0	--
SERCOM4_CORE	☐	GCLK0	--
SERCOM5_CORE	☐	GCLK0	--
SYSCTRL_DFLL48	☐	GCLK0	--
SYSCTRL_FDPILL	☐	GCLK0	--
SYSCTRL_FDPILL32K	☐	GCLK0	--
TC3, TCC2	☐	GCLK0	--
TC4, TC5	☐	GCLK0	--
TCC0, TCC1	☐	GCLK0	--
USB	☒	GCLK0	48,000,000 Hz
WDT	☐	GCLK0	--

Close

USB Device API

API functions in `usb_device.c`

<code>USB_DEVICE_Initialize()</code>	Creates and initializes an instance of the USB device layer
<code>USB_DEVICE_Deinitialize()</code>	De-initializes the specified instance of the USB device layer
<code>USB_DEVICE_Tasks()</code>	USB Device layer calls all other function driver tasks in this function
<code>USB_DEVICE_Open()</code>	Opens the specified USB device layer instance and returns a handle to it
<code>USB_DEVICE_Close()</code>	Closes an opened handle to an instance of the USB device layer
<code>USB_DEVICE_Status()</code>	Provides the current status of the USB device layer
<code>USB_DEVICE_ClientStatusGet()</code>	Gets the current client-specific status of the USB device layer
<code>USB_DEVICE_StateGet()</code>	Returns the current state of the USB device
<code>USB_DEVICE_Attach()</code>	This function will attach the device to the USB
<code>USB_DEVICE_Detach()</code>	This function will detach the device from the USB
<code>USB_DEVICE_ActiveSpeedGet()</code>	Informs the client of the current operation speed of the USB bus
<code>USB_DEVICE_PowerStateSet()</code>	Sets the power state of the device
<code>USB_DEVICE_ActiveConfigurationGet()</code>	Informs the client of the current USB device configuration set by the USB
<code>USB_DEVICE_EventHandlerSet()</code>	Sets up the callback function that will be called from the USB device layer

USB Device API cont.

API functions in `usb_device.c`

`USB_DEVICE_IsSuspended( )`

Returns true if the device is in a suspended state

`USB_DEVICE_RemoteWakeupStatusGet( )`

This function returns the status of remote wakeup capability of the device

`void USB_DEVICE_RemoteWakeupStart( )`

This function will start the resume signalling

`void USB_DEVICE_RemoteWakeupStop( )`

This function will stop the resume signalling

`void USB_DEVICE_RemoteWakeupStartTimed ( )`

This function will start a self timed Remote Wake-up

`USB_DEVICE_ControlSend( );`

Sends data stage of the control transfer from device to host

`USB_DEVICE_ControlReceive( );`

Receives data stage of the control transfer from host to device

`USB_DEVICE_ControlStatus( );`

Initiates status stage of the control transfer

`USB_DEVICE_EndpointStall( );`

This function stalls an endpoint in the specified direction

`USB_DEVICE_EndpointStallClear( );`

This function clears the stall on an endpoint in the specified direction

`USB_DEVICE_IRPSubmit( );`

This function submits anIRP for processing to the Hi-Speed USB Driver

`USB_DEVICE_IRPCancelAll( );`

This function cancels all IRPs that are queued and in progress at the specified endpoint

`USB_DEVICE_IRPCancel( );`

This function cancels the specific IRP that are queued and in progress at the specified endpoint

USB Device API - Device Handler Register

// Register the Device Driver Layer event handler.

```
void USB_DEVICE_EventHandlerSet  
(  
    USB_DEVICE_HANDLE usbDeviceHandle,  
    const USB_DEVICE_EVENT_HANDLER callBackFunc,  
    uintptr_t context  
);
```

e.g. :

```
USB_DEVICE_EventHandlerSet  
(  
    appData.usbDevHandle,  
    APP_USBDeviceEventHandler,  
    0  
);
```

USB CDC - Events of Device Layer Event Handler (usb_device.h)

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandle ( USB_DEVICE_EVENTevent , void *pData , uintptr_t context)
{
    ...
    switch(event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED : break;
        case USB_DEVICE_EVENT_POWER_REMOVED : break;
        case USB_DEVICE_EVENT_SUSPENDED : break;
        case USB_DEVICE_EVENT_SOF : break;
        case USB_DEVICE_EVENT_RESET : break;
        case USB_DEVICE_EVENT_DECONFIGURED : break;
        case USB_DEVICE_EVENT_ERROR : break;
        case USB_DEVICE_EVENT_CONFIGURED : break;
        case USB_DEVICE_EVENT_RESUMED : break;
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST : break;
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED : break;
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT : break;
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED : break;
        case USB_DEVICE_EVENT_ENDPOINT_READ_COMPLETE : break;
        case USB_DEVICE_EVENT_ENDPOINT_WRITE_COMPLETE : break;
        case USB_DEVICE_EVENT_SET_DESCRIPTOR : break;
        case USB_DEVICE_EVENT_SYNCH_FRAME : break;
        default : break;
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

USB CDC - Events of Device Layer Event Handler

- **USB_DEVICE_EVENT_POWER_DETECTED**
- **USB_DEVICE_EVENT_POWER_REMOVED**
 - **USB Attach & Detach**
- **USB_DEVICE_EVENT_CONFIGURED**
 - **USB Configured**
- **USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST**
- **USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED**
 - **Control Transfer Data Require & Received**

Lab4 USB Device Device Layer Initialize

Lab4 USB Device Device Layer Initialize

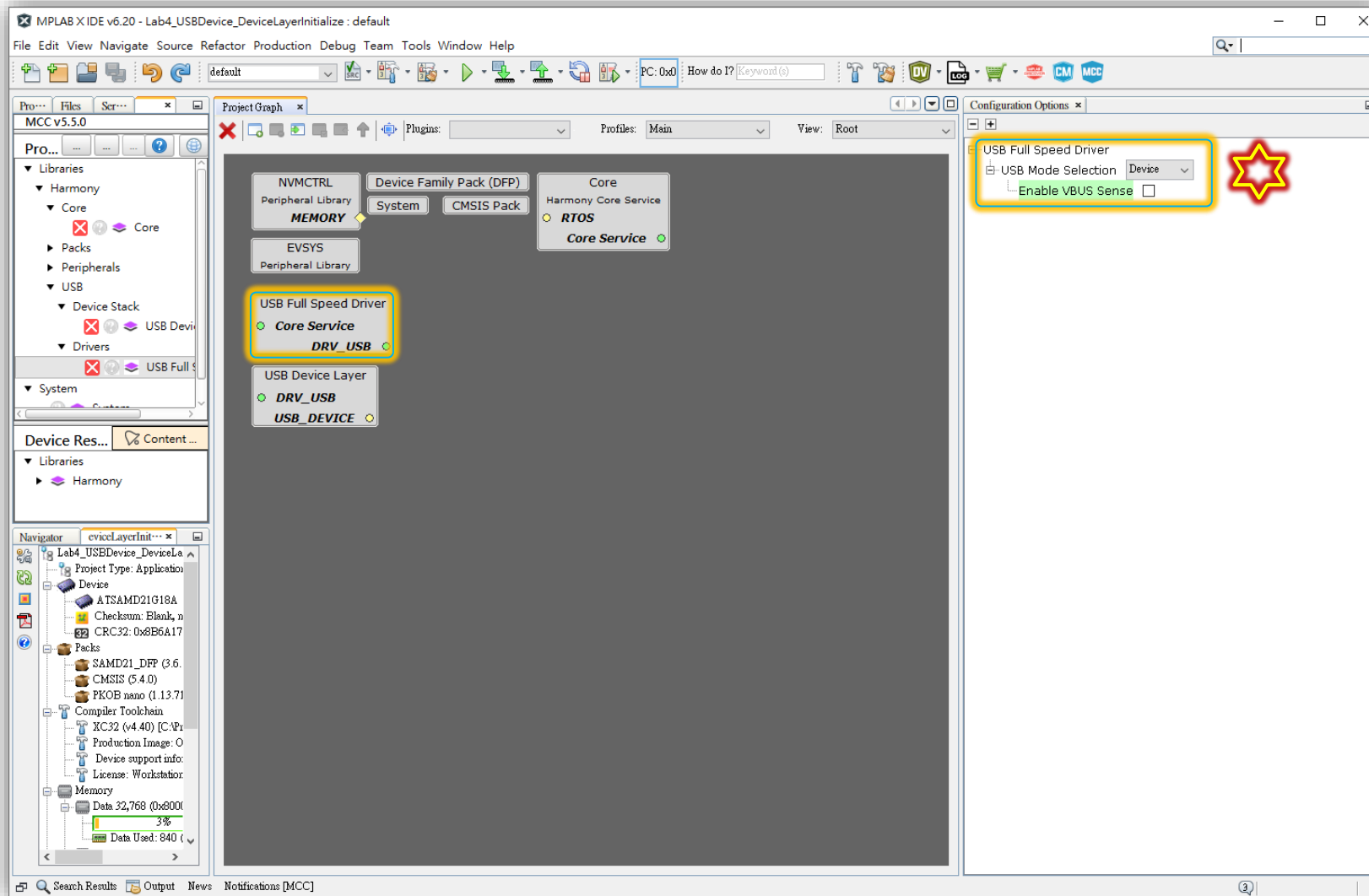
- Create a new project and add USB FS Driver & USB Device Layer in MHC.

The screenshot displays the MPLAB X IDE v6.20 interface for the project 'Lab4_USBDevice_DeviceLayerInitialize'. The Project Resources tree on the left shows the 'USB' folder expanded, with 'USB Device Layer' and 'USB Full Speed Driver' highlighted by red star icons. The Project Graph in the center shows the 'USB Device Layer' and 'USB Full Speed Driver' components. The Configuration Options dialog on the right shows the 'USB Device Layer' and 'USB Full Speed Driver' components. The Pin Table Editor on the right shows the pin configuration for the 'TCC0' and 'TCC1' modules, with the 'USB' module highlighted by a red star icon. The Peripheral Clock Configuration dialog at the bottom shows the clock configuration for various peripherals, with the 'USB' peripheral highlighted by a red star icon.

Peripheral	Enable	Source	Peripheral Clock Frequency
SERCOM2_CORE	☐	GCLK0	--
SERCOM3_CORE	☐	GCLK0	--
SERCOM4_CORE	☐	GCLK0	--
SERCOM5_CORE	☐	GCLK0	--
SYSCTRL_DFLL48	☐	GCLK0	--
SYSCTRL_FDPLL	☐	GCLK0	--
SYSCTRL_FDPLL32K	☐	GCLK0	--
TC3, TCC2	☐	GCLK0	--
TC4, TC5	☐	GCLK0	--
TCC0, TCC1	☐	GCLK0	--
USB	☒	GCLK0	48,000,000 Hz
WDT	☐	GCLK0	--

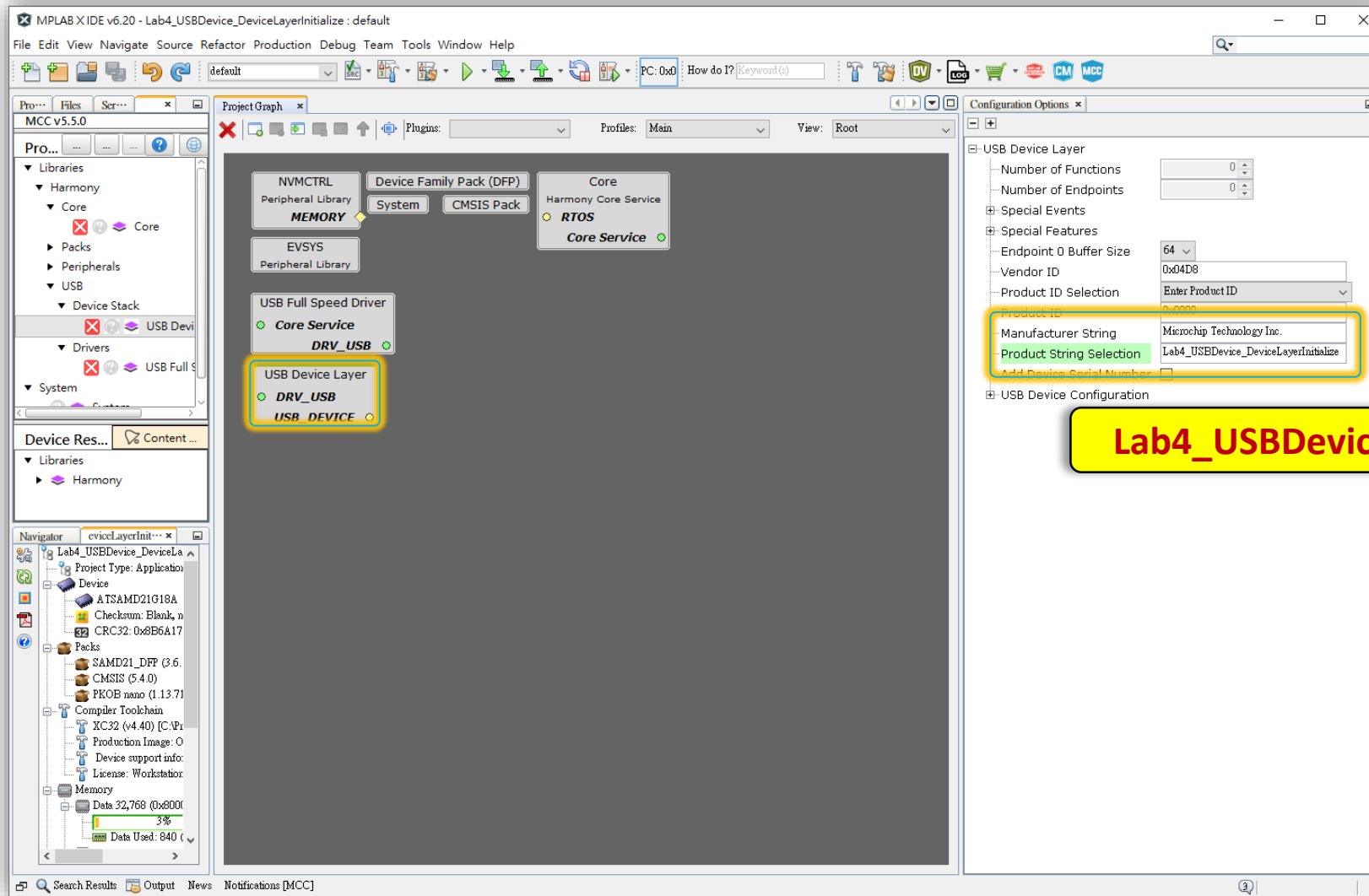
Lab4 USB Device Device Layer Initialize

- Uncheck “Enable VBUS Sense”



⚡ Lab4 USB Device Device Layer Initialize

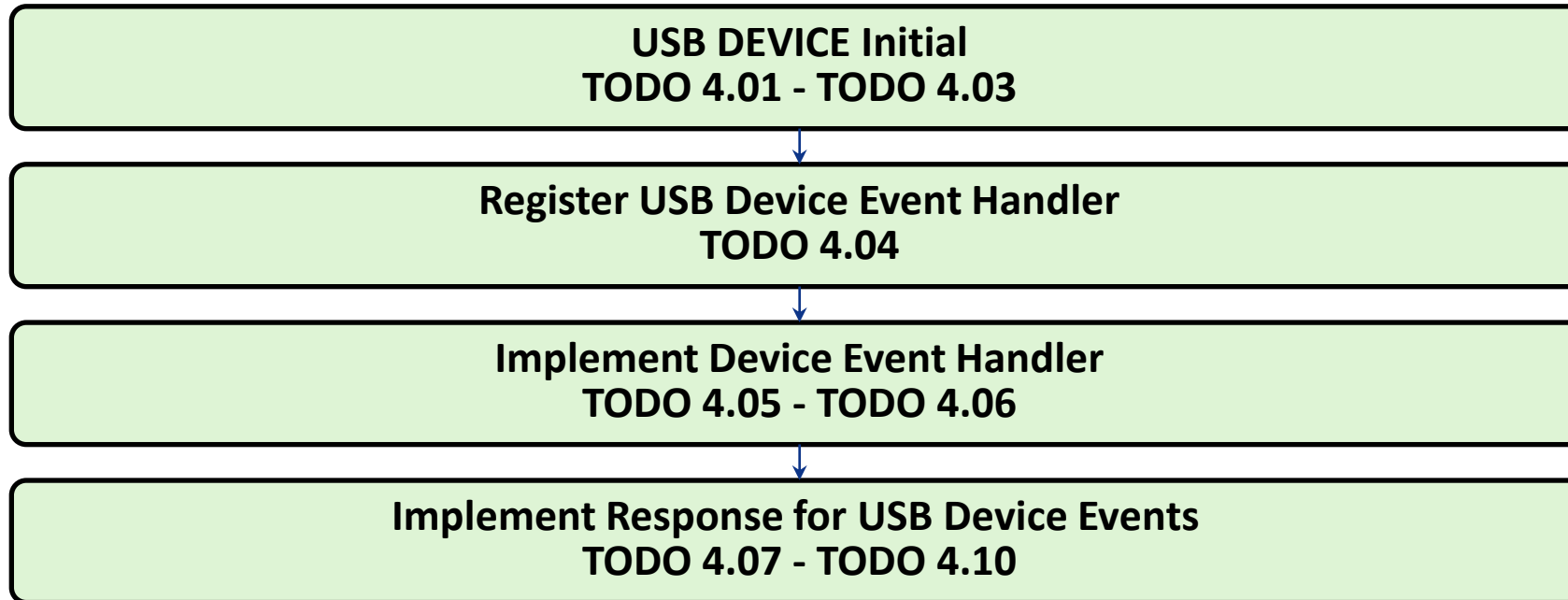
- Change “Product String Selection” to “Lab4_USBDevice_DeviceLayerInitialize”



Lab4_USBDevice_DeviceLayerInitialize

Lab4 USB Device Device Layer Initialize

- Code Implementation state



⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.h.

```
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include "configuration.h"
#include "config/default/usb/usb_device.h"
```

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;

    /* TODO: Define any additional data used by the application. */
    // TODO 4.01
    USB_DEVICE_HANDLE usbDevHandle;
    // TODO 4.02
    bool deviceIsConfigured;
} APP_DATA;
```

⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.

```
#include "app.h"
#include "usb/usb_device.h"
```

```
void APP_Tasks(void)
{
    /* Check the application's current state. */
    switch (appData.state)
    {
        /* Application's initial state. */
        case APP_STATE_INIT:

            bool appInitialized = true;

            // TODO 4.03
            appData.usbDevHandle = USB_DEVICE_Open(USB_DEVICE_INDEX_0, 0);
            appInitialized &= (appData.usbDevHandle != USB_DEVICE_HANDLE_INVALID);

            if (appInitialized)
            {
                // TODO 4.04
                USB_DEVICE_EventHandlerSet(appData.usbDevHandle, APP_USBDeviceEventHandler, 0);
                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;

        case APP_STATE_SERVICE_TASKS:
            break;

        default:
            break;
    }
}
```

⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.

(To copy “APP_USBDeviceEventHandler” callback function form usb_device.h to app.c)

```
/* TODO: Add any necessary callback functions.
 */
// TODO 4.05
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;
    int16_t frameNumber;
    USB_SPEED attachSpeed;
    USB_SETUP_PACKET * setupEventData;

    // Handling of each event
    switch (event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // This means the device detected a valid VBUS voltage and is
            // attached to the USB. The application can now call
            // USB_DEVICE_Attach() function to enable D+/D- pull up
            // resistors.
            break;
        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This means the device is not attached to the USB.
            // The application should now call the USB_DEVICE_Detach()
            // function.
            break;
        case USB_DEVICE_EVENT_SUSPENDED:
            break;
        case USB_DEVICE_EVENT_SOF:
            break;
        case USB_DEVICE_EVENT_RESET:
            break;
        ...

    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

✖ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.
(remove all code segment at “APP_USBDeviceEventHandler”)

```
/* TODO: Add any necessary callback functions.
 */
// TODO 4.05
// TODO 4.06
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;
    int16_t frameNumber;
    USB_SPEED attachSpeed;
    USB_SETUP_PACKET * setupEventData;

    // Handling of each event
    switch (event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // This means the device detected a valid VBUS voltage and is
            // attached to the USB. The application can now call
            // USB_DEVICE_Attach() function to enable D+/D- pull up
            // resistors.
            break;
        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This means the device is not attached to the USB.
            // The application should now call the USB_DEVICE_Detach()
            // function.
            break;
        case USB_DEVICE_EVENT_SUSPENDED:
            break;
        case USB_DEVICE_EVENT_SOF:
            break;
        case USB_DEVICE_EVENT_RESET:
            break;
        ...

    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;

    // Handling of each event
    switch (event)
    {
        case USB_DEVICE_EVENT_POWER_DETECTED:
            // TODO 4.07
            USB_DEVICE_Attach(appData.usbDevHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
            // This means the device is not attached to the USB.
            // TODO 4.08
            USB_DEVICE_Detach(appData.usbDevHandle);
            appData.deviceIsConfigured = false;
            break;

        ...

        case USB_DEVICE_EVENT_CONFIGURED:
            // TODO 4.09
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
                appData.deviceIsConfigured = true;
            break;

        ...

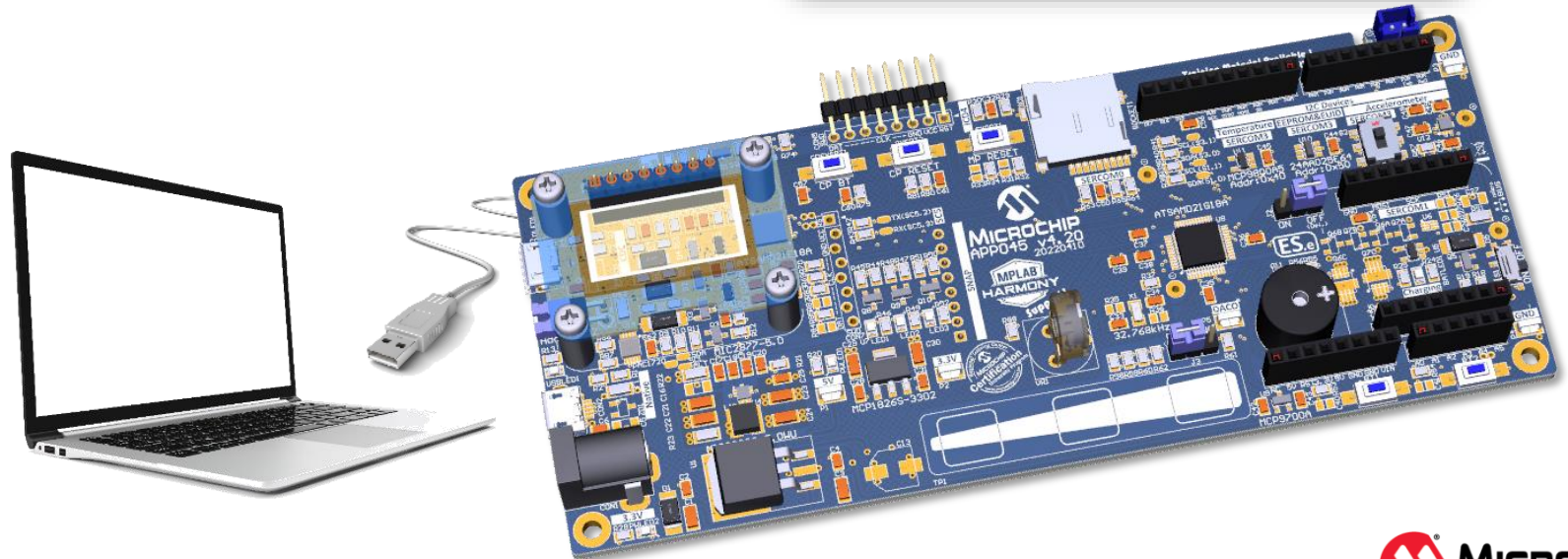
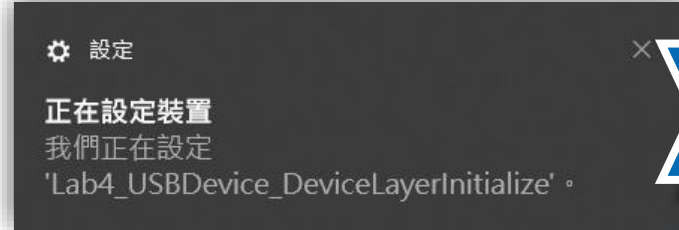
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
            // TODO 4.10
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        ...

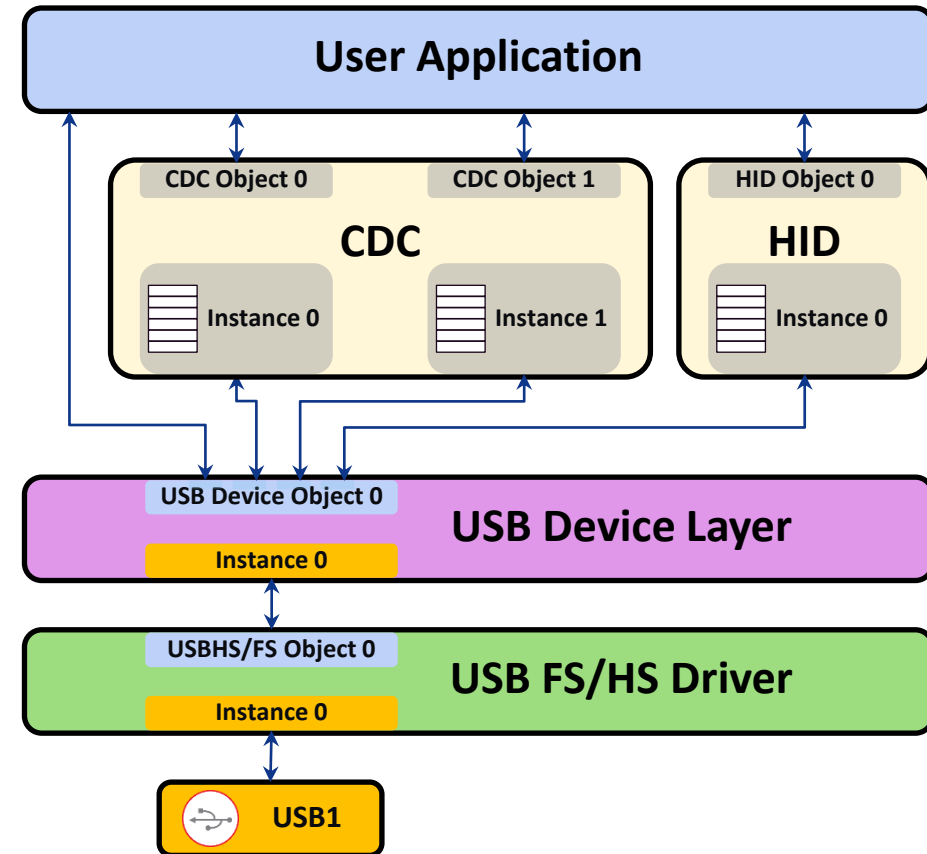
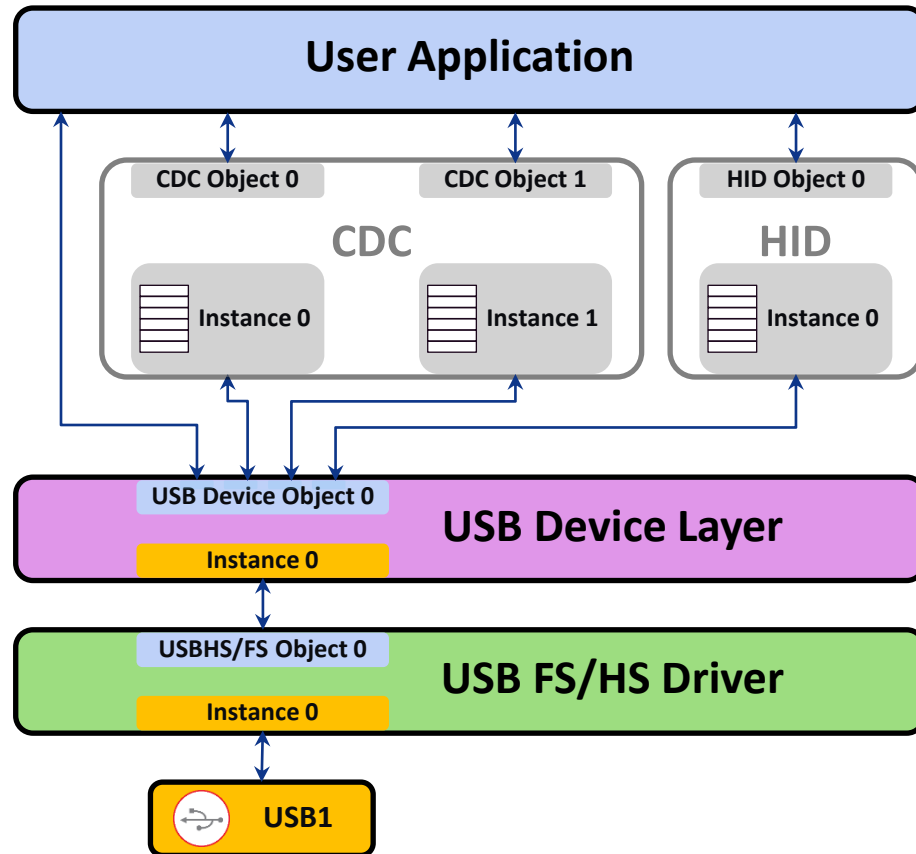
        return USB_DEVICE_EVENT_RESPONSE_NONE;
    }
}
```

✖ Lab4 USB Device Device Layer Initialize

- The device list at Device Manager, Named “Lab4_USBDevice_DeviceLayerInitialize”. After configured.



Apply Function Service Next Step



USB Device CDC Class

USB Device CDC Class

- CDC (Communications Device Class)
- CDC include lot of subclass,
 - PTSN (Public Switched Telephone Network)
 - DLM : Direct Line Control Model
 - ACM : Abstract Control Model
 - TCM : Telephone Control Model
 - ISDN (Integrated Services Digital Network)
 - MLCM : Multi Line Control Model
 - CCM : CAPI(Common Application Programming Interface) Control Model
 - ECM : Ethernet Networking Control Model
 - ATM : Asynchronous Transfer Mode
 - etc..
- Virtual COM (USB Serial) was defined at PTSN > ACM(02h) (AT v.250)(01h)

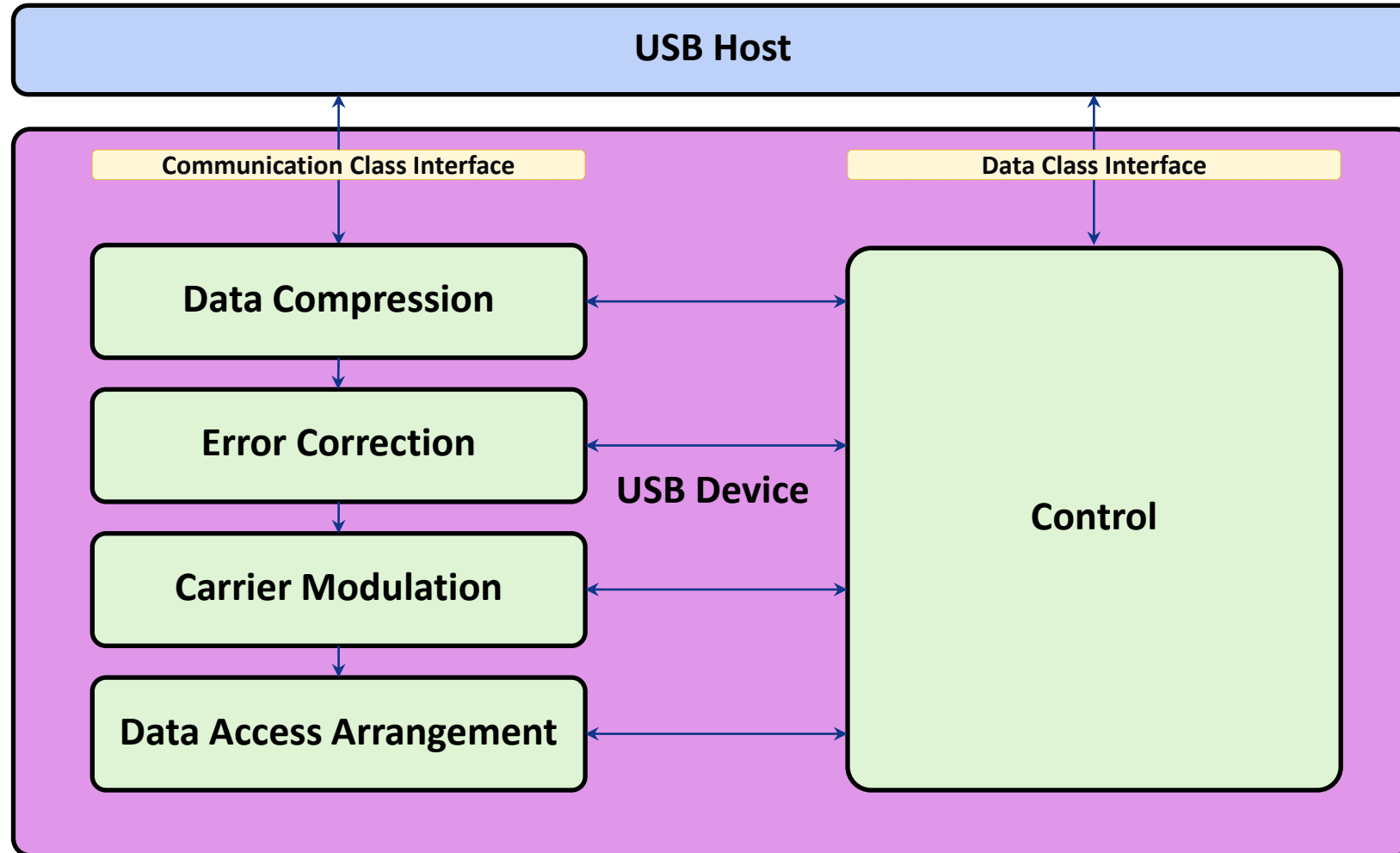
Communications Class Subclass Codes

Code	Subclass	Reference
00h	RESERVED	
01h	Direct Line Control Model	[USBPSTN1.2]
02h	Abstract Control Model	[USBPSTN1.2]
03h	Telephone Control Model	[USBPSTN1.2]
04h	Multi-Channel Control Model	[USBISDN1.2]
05h	CAPI Control Model	[USBISDN1.2]
06h	Ethernet Networking Control Model	[USBECM1.2]
07h	ATM Networking Control Model	[USBATM1.2]
08h	Wireless Handset Control Model	[USBWMC1.1]
09h	Device Management	[USBWMC1.1]
0Ah	Mobile Direct Line Model	[USBWMC1.1]
0Bh	OBEX	[USBWMC1.1]
0Ch	Ethernet Emulation Model	[USB EEM1.0]
0Dh	Network Control Model	[USBNCM1.0]
0Dh-7Fh	RESERVED (future use)	
80-FEh	RESERVED (vendor specific)	

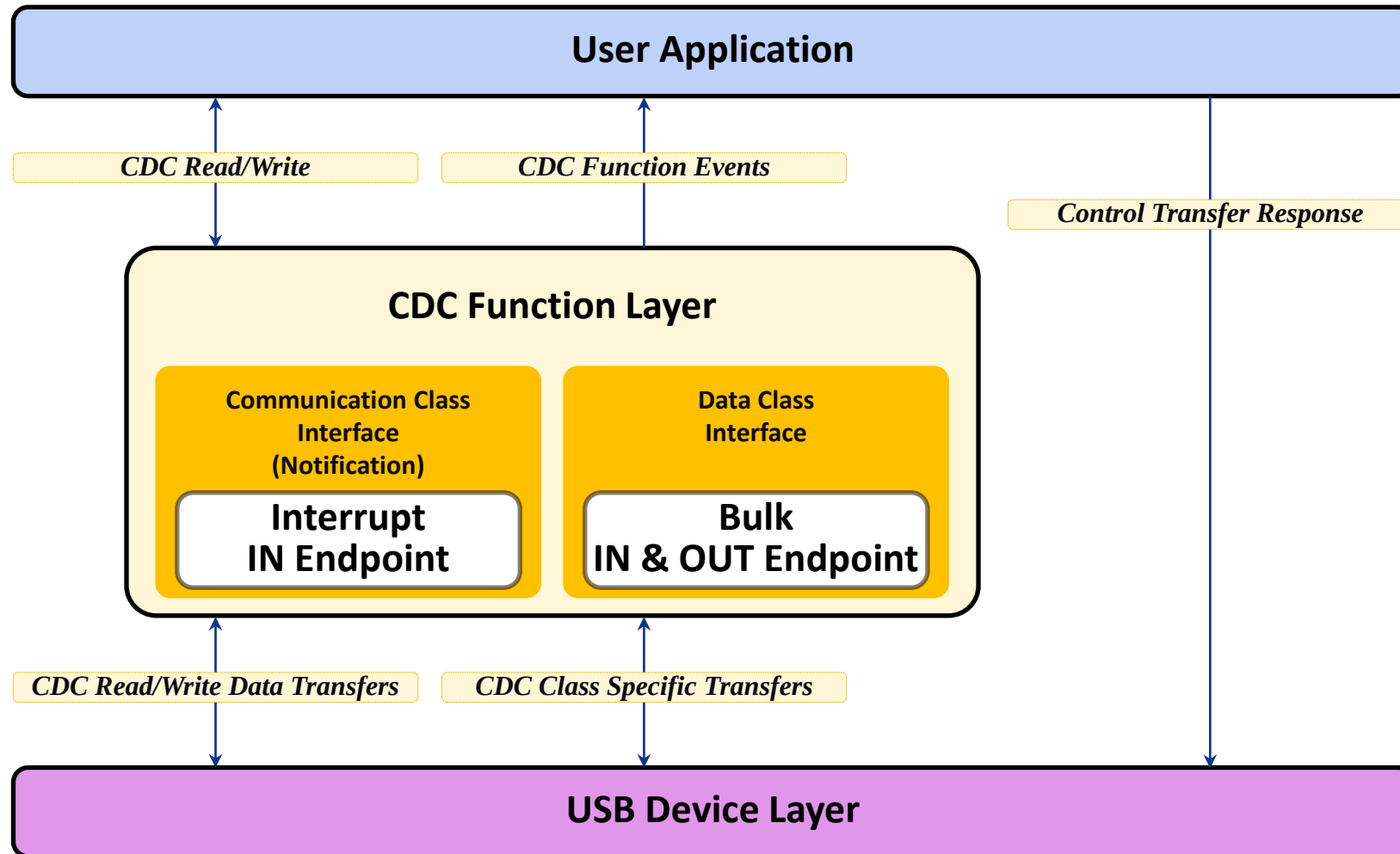
Communications Class Protocol Codes

Protocol Code	Reference document	Description
00h	USB specification	No class specific protocol required
01h	ITU-T V.250	AT Commands: V.250 etc
02h	PCCA-101	AT Commands defined by PCCA-101
03h	PCCA-101	AT Commands defined by PCCA-101 & Annex O
04h	GSM 7.07	AT Commands defined by GSM 07.07
05h	3GPP 27.07	AT Commands defined by 3GPP 27.007
06h	C-S0017-0	AT Commands defined by TIA for CDMA
07h	USB EEM	Ethernet Emulation Model
08h-FDh		RESERVED (future use)
FEh		External Protocol: Commands defined by Command Set Functional Descriptor
FFh	USB Specification	Vendor-specific

USB CDC Subclass - Abstract Control Model



USB CDC Abstraction Model



USB CDC ACM Configuration Descriptor Example

```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    /* Configuration Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_CONFIGURATION, // Descriptor Type
    USB_DEVICE_16bitTo8bitArrange(67), //(67 Bytes)Size of the Configuration descriptor
    2, // Number of interfaces in this configuration
    0x01, // Index value of this configuration
    0x00, // Configuration string index
    USB_ATTRIBUTE_DEFAULT | USB_ATTRIBUTE_SELF_POWERED, // Attributes
    50, // Maximum power consumption (mA) /2
    ...
}
```

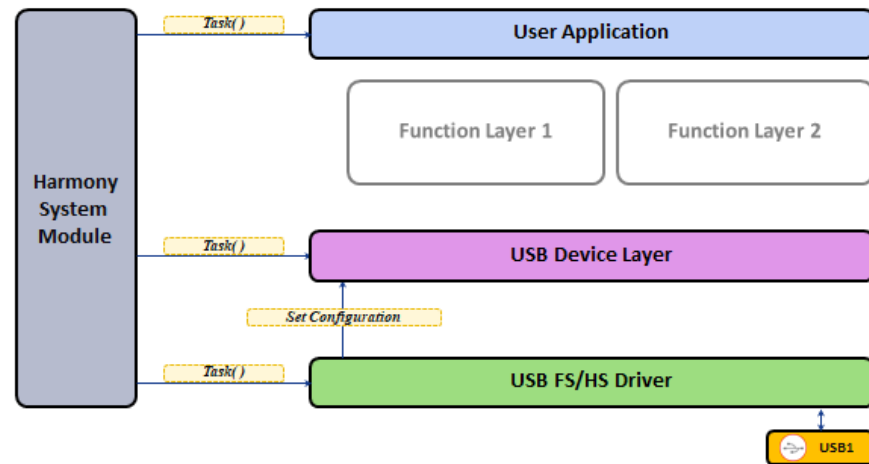
```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // Descriptor Type is Int
    0, // Interface Number
    0x00, // Alternate Setting Number
    0x01, // Number of endpoints in this interface
    USB_CDC_COMMUNICATIONS_INTERFACE_CLASS_CODE, // Cla
    USB_CDC_SUBCLASS_ABSTRACT_CONTROL_MODEL, // Subclas
    USB_CDC_PROTOCOL_AT_V250, // Protocol code
    0x00, // Interface string index
    ...
}
```

```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    1, // Interface Number
    0x00, // Alternate Setting Number
    0x02, // Number of endpoints in this interface
    USB_CDC_DATA_INTERFACE_CLASS_CODE, // Class code
    0x00, // Subclass code
    USB_CDC_PROTOCOL_NO_CLASS_SPECIFIC, // Protocol code
    0x00, // Interface string index
    ...
}
```

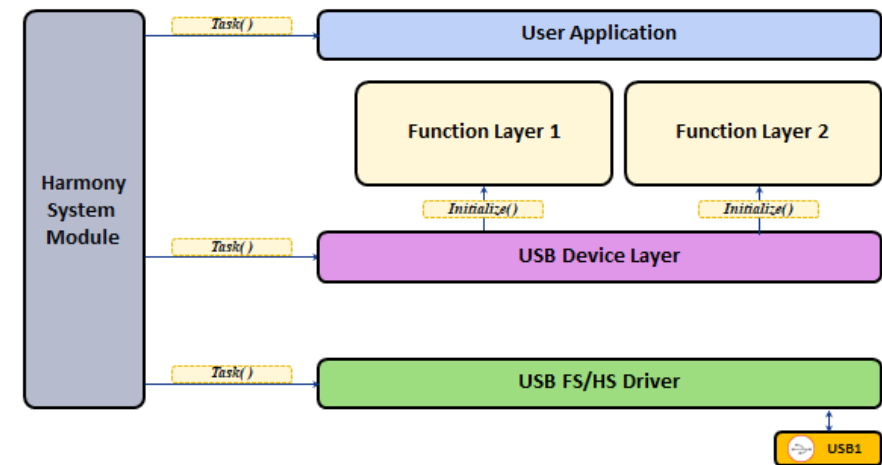
Review USB Stack Abstraction Model

- Ideally, Function Layer be initialized after “Set Configuration” state.
- So, function layer needs initialize and set function event handler at there.

USB Stack Abstraction Model - Set Configuration

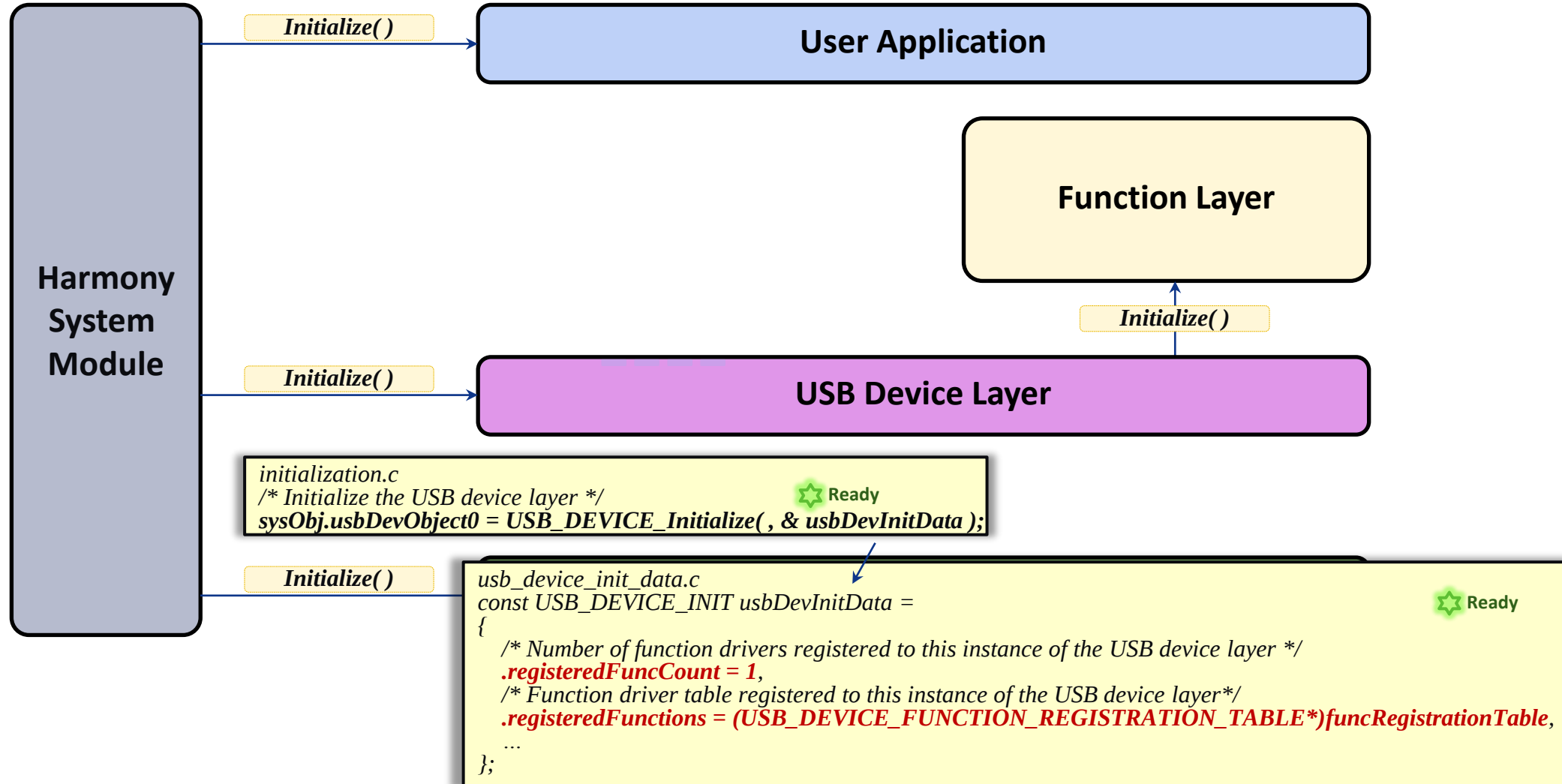


USB Stack Abstraction Model - Function Initialize & Open



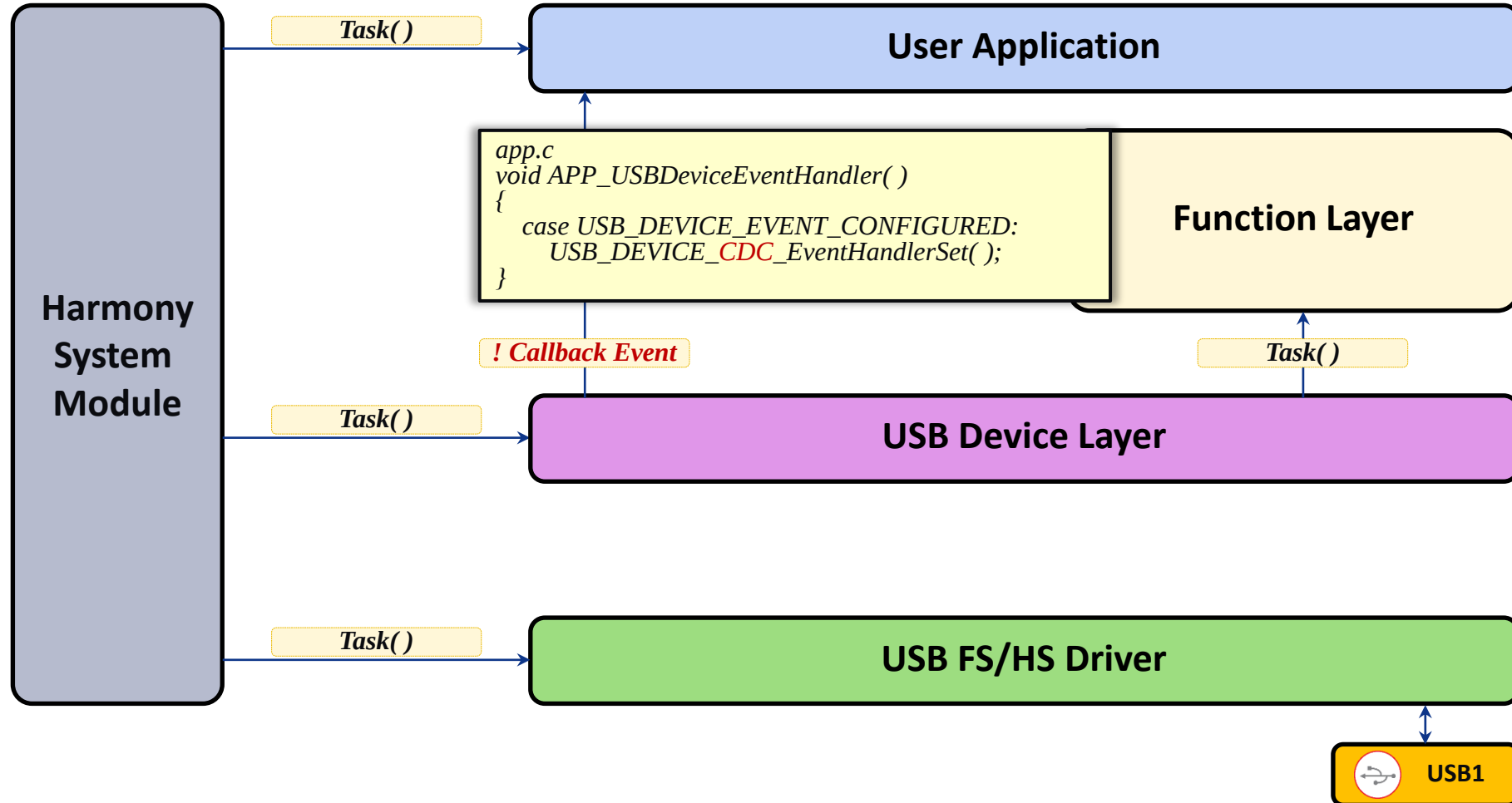
Review USB Stack Abstraction Model

- In fact, Function Layer be initialized at “Device Layer Open” state Harmony v3.

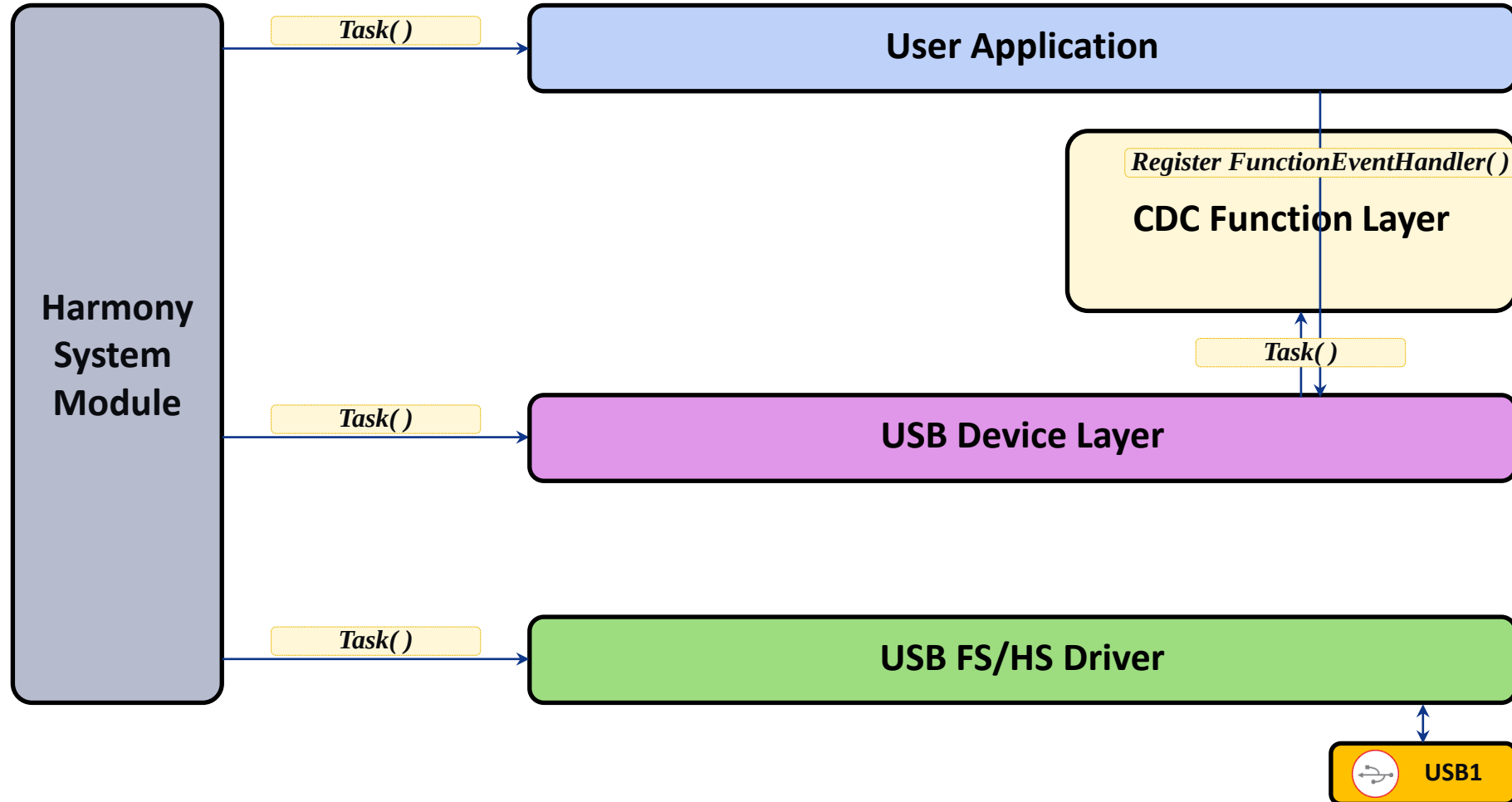


USB Stack Abstraction Model - Register Function Event Handler

- So, Register Function Event Handler only, after “Set Configuration” state.



USB Stack Abstraction Model - Register Function Event Handler



USB CDC API

API functions in `usb_device_cdc.c`

`_USB_DEVICE_CDC_GlobalInitialize( )`
`_USB_DEVICE_CDC_Initialization( )`
`_USB_DEVICE_CDC_Deinitialization( )`
`_USB_DEVICE_CDC_EndpointDisable( )`

Called by the device layer during Set Configuration Processing.
Called by the device layer during Set Configuration Processing.
Deinitializes the function driver instance.
Disabled USB Device CDC endpoints.

`_USB_DEVICE_CDC_ControlTransferHandler( )`
`_USB_DEVICE_CDC_ReadIRPCallback( )`
`_USB_DEVICE_CDC_WriteIRPCallback( )`
`_USB_DEVICE_CDC_SerialStateSendIRPCallback( )`

Control Transfer Handler for class specific control transfer.
for IRPs submitted through the `_USB_DEVICE_CDC_Read()`.
for IRPs submitted through the `_USB_DEVICE_CDC_Write()`.
for IRPs submitted through the `_USB_DEVICE_CDC_SerialStateSend()`.

`USB_DEVICE_CDC_EventHandlerSet( )`

Register the CDC Function Driver Layer event handler

`USB_DEVICE_CDC_Read( )`
`USB_DEVICE_CDC_Write( )`

Requests a data read from the CDC Function Driver Layer.
Requests a data write to the USB Device CDC Function Driver Layer.

`USB_DEVICE_CDC_ReadPacketSizeGet( )`
`USB_DEVICE_CDC_WritePacketSizeGet( )`
`USB_DEVICE_CDC_SerialStateNotificationSend( )`

Return read packet size.
Return write packet size.
Request to send serial state notification to the host.

USB CDC API - Event Handler Register

// Register the CDC Function Driver Layer event handler.

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_EventHandlerSet  
(  
    USB_DEVICE_CDC_INDEX iCDC,  
    USB_DEVICE_CDC_EVENT_HANDLER eventHandler,  
    uintptr_t userData  
);
```

e.g. :

```
USB_DEVICE_CDC_EventHandlerSet  
(  
    USB_DEVICE_CDC_INDEX_0,  
    USBDeviceCDCEventHandler,  
    0  
);
```

USB CDC API - Event Handler Register Example

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    switch (event)
    {
        ...
        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0, USBDeviceCDCEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

        ...
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```



```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(USB_DEVICE_CDC_INDEX instanceIndex, USB_DEVICE_CDC_EVENT event,
void * pData, uintptr_t userData)
{
    switch (event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING: break;
        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING: break;
        case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE: break;
        case USB_DEVICE_CDC_EVENT_SEND_BREAK: break;
        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_SENT: break;
        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED: break;
        case USB_DEVICE_CDC_EVENT_WRITE_COMPLETE: break;
        case USB_DEVICE_CDC_EVENT_READ_COMPLETE: break;
        case USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE: break;

        ...
    }
    return USB_DEVICE_CDC_EVENT_RESPONSE_NONE;
}
```

USB CDC - Events of CDC Event Handler

- **USB_DEVICE_CDC_EVENT_SET_LINE_CODING**
- **USB_DEVICE_CDC_EVENT_GET_LINE_CODING**
 - Line coding set & get.

```
Usb_cdc.h
typedef struct __attribute__((packed))
{
    /* data terminal rate in bits per second */
    uint32_t dwDTERate;
    /* stop bits */
    uint8_t bCharFormat;
    /* Parity */
    uint8_t bParityType;
    /* Data bits */
    uint8_t bDataBits;
} USB_CDC_LINE_CODING;
```

- **USB_DEVICE_CDC_EVENT_SERIAL_STATE_NOTIFICATION_COMPLETE**

```
Usb_cdc.h
typedef struct __attribute__((packed))
{
    uint8_t bRxCarrier :1;
    uint8_t bTxCarrier :1;
    uint8_t bBreak :1;
    uint8_t bRingSignal :1;
    uint8_t bFraming :1;
    uint8_t bParity :1;
    uint8_t bOverRun :1;
    uint8_t :1;
    uint8_t :8;
} USB_CDC_SERIAL_STATE;
```

- **USB_DEVICE_CDC_EVENT_READ_COMPLETE**
- **USB_DEVICE_CDC_EVENT_WRITE_COMPLETE**
 - Completed read/write transfer and the amount of data get/send

USB CDC API - USB_DEVICE_CDC_Read

// Requests a data read from the CDC Function Driver Layer.

```
USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Read  
(  
    USB_DEVICE_CDC_INDEX iCDC,  
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle,  
    void * data, size_t size  
);
```

e.g. :

```
USB_DEVICE_CDC_Read  
(  
    USB_DEVICE_CDC_INDEX_0,  
    &appData.readTransferHandle,  
    appData.CDCrxDataBuffer, 64  
);
```

USB CDC API - USB_DEVICE_CDC_Write

// Requests a data write to the USB Device CDC Function Driver Layer.

USB_DEVICE_CDC_RESULT USB_DEVICE_CDC_Write

```
(  
    USB_DEVICE_CDC_INDEX iCDC ,  
    USB_DEVICE_CDC_TRANSFER_HANDLE * transferHandle ,  
    const void * data , size_t size ,  
    USB_DEVICE_CDC_TRANSFER_FLAGS flags  
);
```

e.g. :

USB_DEVICE_CDC_Write

```
(  
    USB_DEVICE_CDC_INDEX_0,  
    &appData.writeTransferHandle,  
    appData.CDCtxDataBuffer, sizeof( appData.CDCtxDataBuffer ),  
    USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE  
);
```

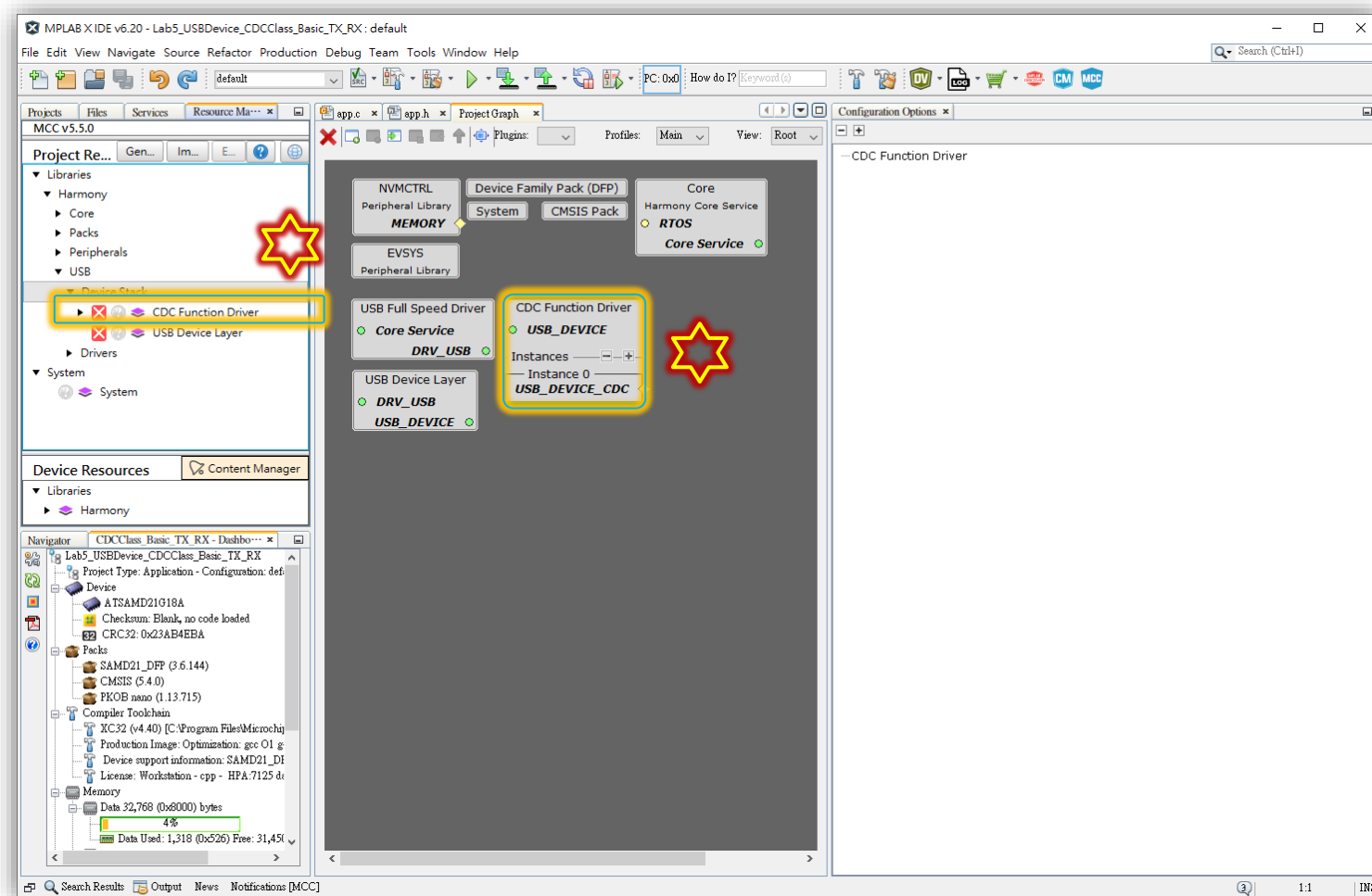
Lab5 USB Device CDC Class - Basic TX and RX

Lab5 USB Device CDC Class - Basic TX and RX

- At this lab, we try to add CDC function to our USB Device.
- The CDC function follow ACM subclass and protocol based on AT.250. (Virtual COM).
- Application require,
 - LED1 to LED3 be toggled by '1', '2' and '3' key, by terminal software (Tera Team).
 - Shows "BT1 Pressed!" in terminal screen, when BT1 be pressed at APP045.
- **Let's go !**

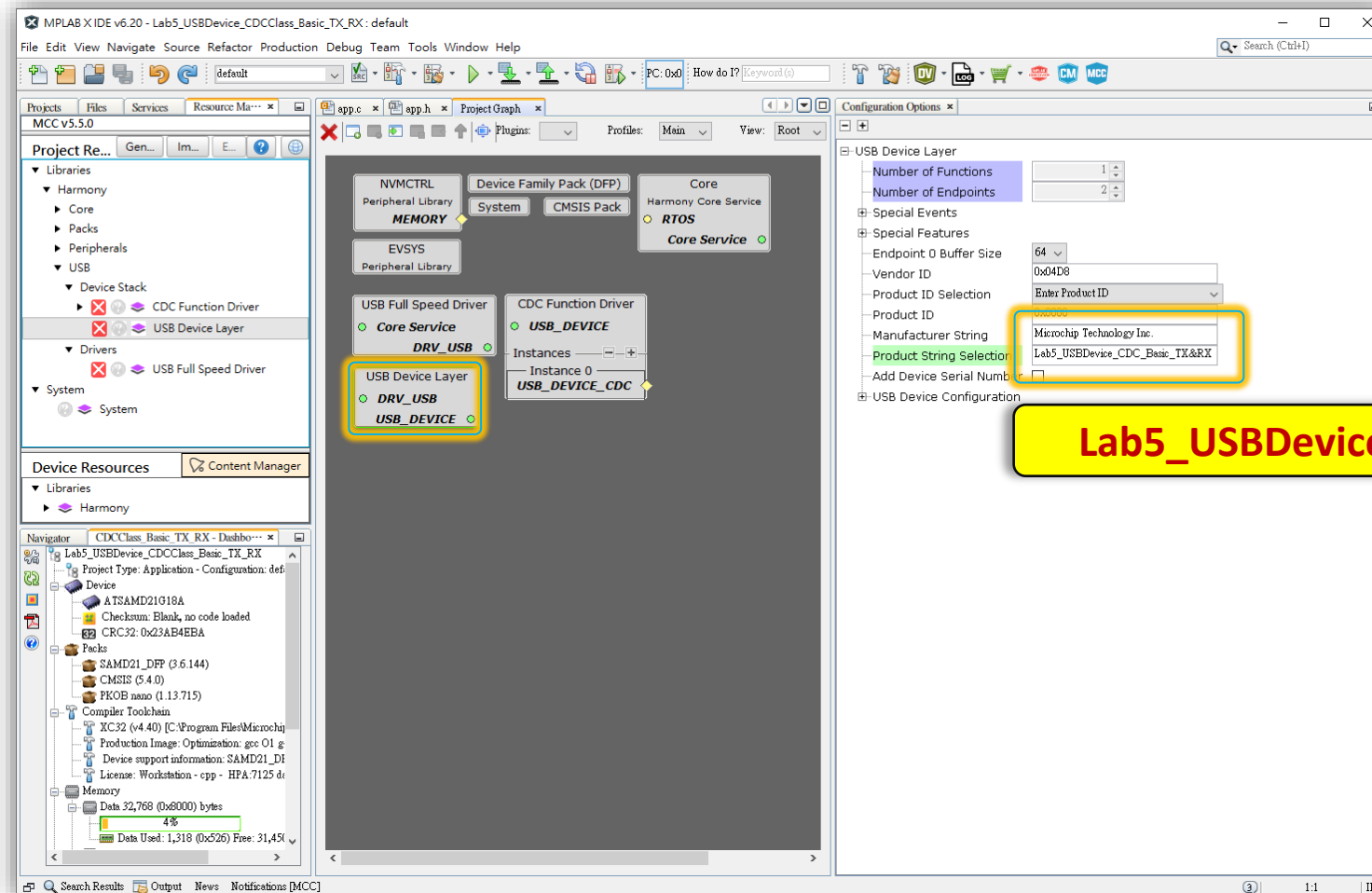
⚡ Lab5 USB Device CDC Class - Basic TX and RX

- Basic on Lab4 and add CDC Function Layer in MHC.



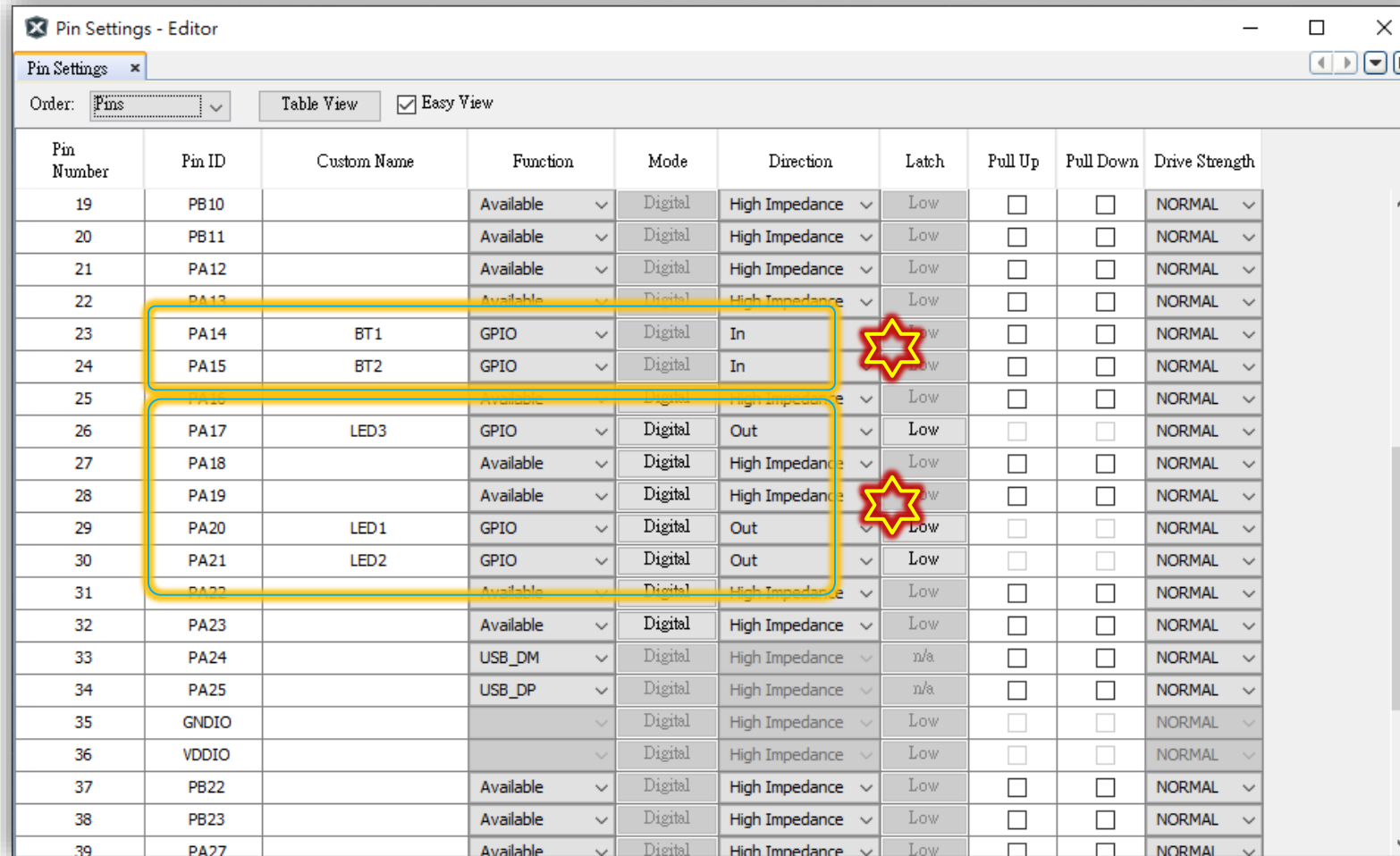
⚡ Lab5 USB Device CDC Class - Basic TX RX

- Change “Product String Selection” to “Lab5_USBDevice_CDC_Basic_TX&RX”



⚙️ Lab5 USB Device CDC Class - Basic TX RX

- Set LEDs and BTs in Pin Settings



Pin Settings - Editor

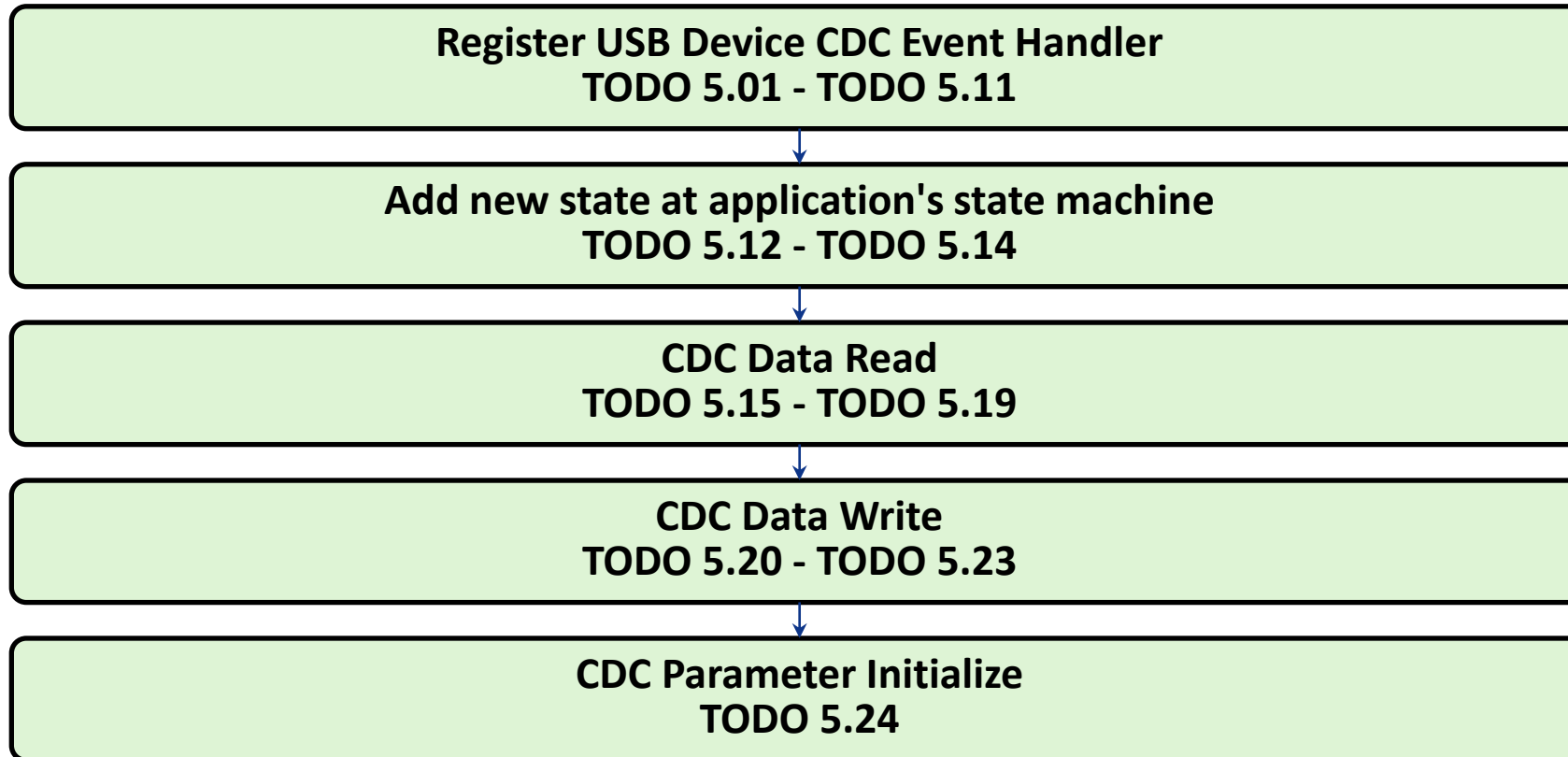
Pin Settings x

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
19	PB10		Available	Digital	High Impedance	Low	☐	☐	NORMAL
20	PB11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
21	PA12		Available	Digital	High Impedance	Low	☐	☐	NORMAL
22	PA13		Available	Digital	High Impedance	Low	☐	☐	NORMAL
23	PA14	BT1	GPIO	Digital	In	Low	☐	☐	NORMAL
24	PA15	BT2	GPIO	Digital	In	Low	☐	☐	NORMAL
25	PA16		Available	Digital	High Impedance	Low	☐	☐	NORMAL
26	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
27	PA18		Available	Digital	High Impedance	Low	☐	☐	NORMAL
28	PA19		Available	Digital	High Impedance	Low	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
30	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
31	PA22		Available	Digital	High Impedance	Low	☐	☐	NORMAL
32	PA23		Available	Digital	High Impedance	Low	☐	☐	NORMAL
33	PA24		USB_DM	Digital	High Impedance	n/a	☐	☐	NORMAL
34	PA25		USB_DP	Digital	High Impedance	n/a	☐	☐	NORMAL
35	GNDIO			Digital	High Impedance	Low	☐	☐	NORMAL
36	VDDIO			Digital	High Impedance	Low	☐	☐	NORMAL
37	PB22		Available	Digital	High Impedance	Low	☐	☐	NORMAL
38	PB23		Available	Digital	High Impedance	Low	☐	☐	NORMAL
39	PA27		Available	Digital	High Impedance	Low	☐	☐	NORMAL

Lab5 USB Device CDC Class - Basic TX RX

- Code Implementation state



⚡ Lab5 USB Device CDC Class - Basic TX RX

- Register USB Device CDC Event Handler

(To copy "APP_USBDeviceEventHandler" callback function form usb_device_cdc.h to app.c)

```
// TODO 5.01
// TODO 5.02
//uint16_t breakDuration;
//USB_DEVICE_HANDLE usbDeviceHandle;
//USB_CDC_LINE_CODING lineCoding;
//USB_CDC_CONTROL_LINE_STATE * controllLineStateData;

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * pData,
    uintptr_t userData
){}
```

```
// TODO 5.10
#include "usb/usb_device_cdc.h"

typedef struct
{
    /* The application's current state */
    APP_STATES state;

    // TODO 5.03
    uint16_t breakDuration;
    USB_CDC_LINE_CODING lineCoding;
    USB_CDC_CONTROL_LINE_STATE * controllLineStateData;
} APP_DATA;
```

```
// TODO 5.09
#include "usb/usb_device_cdc.h"

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * pData,
    uintptr_t userData
)
{
    switch (event)
    {
        case USB_DEVICE_CDC_EVENT_SET_LINE_CODING:
            // TODO 5.04
            USB_DEVICE_ControlReceive(appData.usbDevHandle, &appData.lineCoding,
                                      sizeof (USB_CDC_LINE_CODING));

            break;

        case USB_DEVICE_CDC_EVENT_GET_LINE_CODING:
            // TODO 5.05
            USB_DEVICE_ControlSend(appData.usbDevHandle, &appData.lineCoding,
                                   sizeof (USB_CDC_LINE_CODING));

            break;

        case USB_DEVICE_CDC_EVENT_SET_CONTROL_LINE_STATE:
            // TODO 5.06
            appData.controllLineStateData = (USB_CDC_CONTROL_LINE_STATE *) pData;
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_CDC_EVENT_SEND_BREAK:
            // TODO 5.07
            appData.breakDuration = ((USB_DEVICE_CDC_EVENT_DATA_SEND_BREAK *) pData)->breakDuration;
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        case USB_DEVICE_CDC_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
            // TODO 5.08
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);

            break;

        default:
            break;
    }

    return USB_DEVICE_CDC_EVENT_RESPONSE_NONE;
}
```

Lab5 USB Device CDC Class - Basic TX RX

- Register USB Device CDC Event Handler

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;

    // Handling of each event
    switch (event)
    {
        ...
        case USB_DEVICE_EVENT_DECONFIGURED:
            break;

        case USB_DEVICE_EVENT_ERROR:
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                //TODO 5.11
                USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0, USBDeviceCDCEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            break;

        ...
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

Lab5 USB Device CDC Class - Basic TX RX

- Add new state at application's state machine

```
typedef enum
{
    /* Application's state machine's initial state. */
    // TODO 5.12
    //APP_STATE_INIT = 0,
    //APP_STATE_SERVICE_TASKS,
    APP_STATE_INIT = 0,
    APP_STATE_WAIT_CONFIGURED,
    APP_STATE_SERVICE_TASKS,
    /* TODO: Define states used by the application state machine. */
} APP_STATES;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_INIT:
        {
            if (appInitialized)
            {
                USB_DEVICE_EventHandlerSet(appData.usbDevHandle, APP_USBDeviceEventHandler, 0);

                //TODO 5.13
                appData.state = APP_STATE_WAIT_CONFIGURED;
            }
            break;
        }

        //TODO 5.14
        case APP_STATE_WAIT_CONFIGURED:
        {
            break;
        }

        case APP_STATE_SERVICE_TASKS:
        {
            break;
        }

        default:
        {
            break;
        }
    }
}
```

⚡ Lab5 USB Device CDC Class - Basic TX RX

- CDC Data Read

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;

    //TODO 5.15
    volatile bool cdcReadCompleted;
    USB_DEVICE_CDC_TRANSFER_HANDLE readTransferHandle;
    uint8_t CDCrxDataBuffer[64] __attribute__((aligned(32)));
} APP_DATA;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_WAIT_CONFIGURED:
        {
            //TODO 5.17
            if (appData.deviceIsConfigured)
            {
                appData.cdcReadCompleted = false;
                USB_DEVICE_CDC_Read(USB_DEVICE_CDC_INDEX_0, &appData.readTransferHandle, appData.CDCrxDataBuffer, 64);
                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;
        }
    }
}
```

```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * pData,
    uintptr_t userData
)
{
    switch (event)
    {
        case USB_DEVICE_CDC_EVENT_READ_COMPLETE:
            // TODO 5.16
            appData.cdcReadCompleted = true;
            break;
    }

    return USB_DEVICE_CDC_EVENT_RESPONSE_NONE;
}
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_SERVICE_TASKS:
        {
            //TODO 5.18
            if (appData.cdcReadCompleted)
            {
                switch (appData.CDCrxDataBuffer[0])
                {
                    case '1': LED1_Toggle(); break;
                    case '2': LED2_Toggle(); break;
                    case '3': LED3_Toggle(); break;
                }
                appData.cdcReadCompleted = false;
                USB_DEVICE_CDC_Read(USB_DEVICE_CDC_INDEX_0, &appData.readTransferHandle, appData.CDCrxDataBuffer, 256);
            }
            ...
        }
    }
}
```

⚡ Lab5 USB Device CDC Class - Basic TX RX

- CDC Data Write

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;

    //TODO 5.20
    volatile bool cdcWriteCompleted;
    USB_DEVICE_CDC_TRANSFER_HANDLE writeTransferHandle;
    uint8_t CDCtxDataBuffer[64] __attribute__((aligned(32)));
} APP_DATA;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_WAIT_CONFIGURED:
        {
            //TODO 5.17
            if (appData.deviceIsConfigured)
            {
                appData.cdcReadCompleted = false;
                USB_DEVICE_CDC_Read(USB_DEVICE_CDC_INDEX_0, &appData.readTransferHandle, appData.CDCrxDataBuffer, 64);
                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;
        }
    }
}
```

```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceCDCEventHandler
(
    USB_DEVICE_CDC_INDEX instanceIndex,
    USB_DEVICE_CDC_EVENT event,
    void * pData,
    uintptr_t userData
)
{
    switch (event)
    {
        case _DEVICE_CDC_EVENT_WRITE_COMPLETE:
            // TODO 5.21
            appData.cdcWriteCompleted = true;
            break;
    }

    return USB_DEVICE_CDC_EVENT_RESPONSE_NONE;
}
```

```
//TODO 5.23
#include <stdio.h>

void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_SERVICE_TASKS:
        {
            //TODO 5.22
            if (appData.cdcWriteCompleted)
            {
                if (!BT1_Get())
                {
                    static uint16_t i = 0;
                    appData.cdcWriteCompleted = false;
                    sprintf((char *) appData.CDCtxDataBuffer, "BT1 Pressed! (%05d)\r\n", ++i);
                    USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0, &appData.writeTransferHandle, appData.CDCtxDataBuffer,
                                         sizeof (appData.CDCtxDataBuffer), USB_DEVICE_TRANSFER_FLAGS_DATA_COMPLETE);
                }
            }
            ...
        }
    }
}
```

Lab5 USB Device CDC Class - Basic TX RX

- CDC Parameter Initialize

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;

    // Handling of each event
    switch (event)
    {
        ...
        case USB_DEVICE_EVENT_DECONFIGURED:
            break;

        case USB_DEVICE_EVENT_ERROR:
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                //TODO 5.24
                appData.lineCoding.dwDTERate = 9600;
                appData.lineCoding.bParityType = 0;
                appData.lineCoding.bDataBits = 8;
                appData.lineCoding.bCharFormat = 0;
                appData.cdcReadCompleted = false;
                appData.cdcWriteCompleted = true;

                USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0, USBDeviceCDCEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

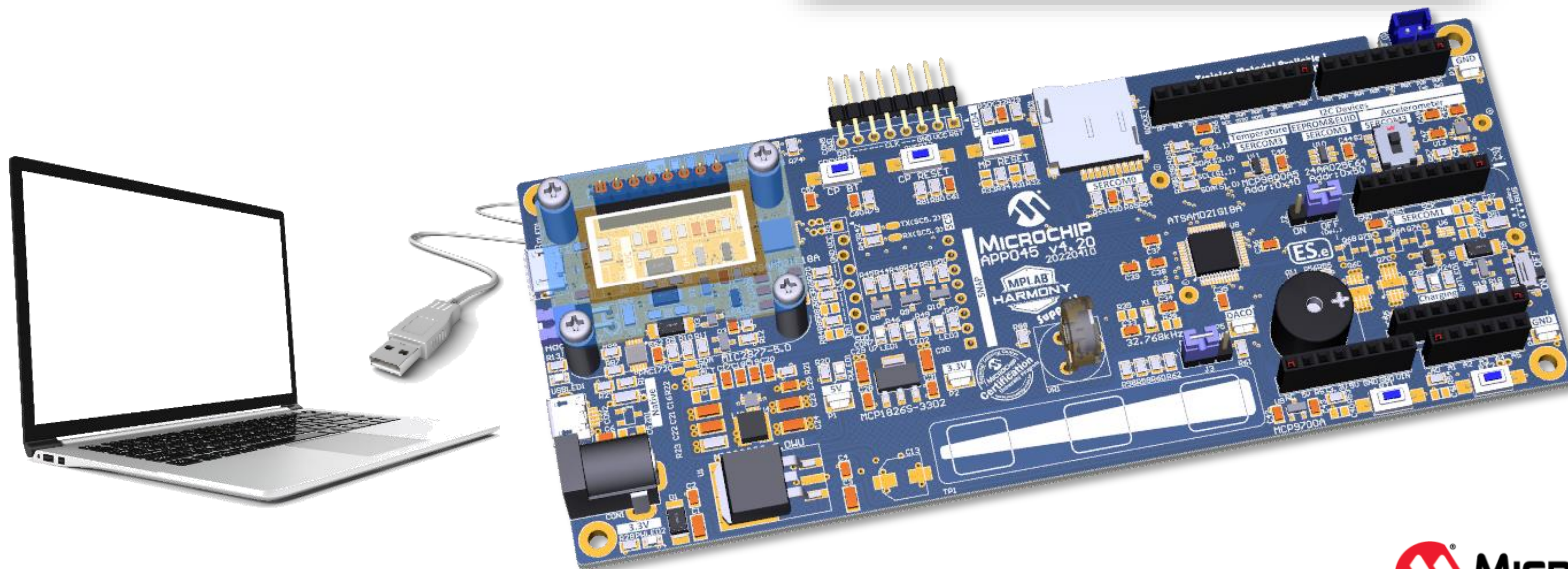
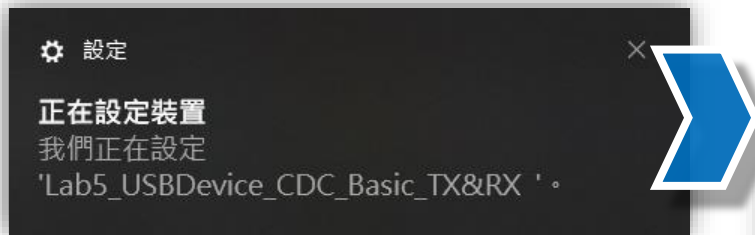
        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            break;

        ...
        return USB_DEVICE_EVENT_RESPONSE_NONE;
    }
}
```

Lab5 USB Device CDC Class - Basic TX RX

- The device list at Device Manager, Named “USB 序列裝置(COMn)” After configured.



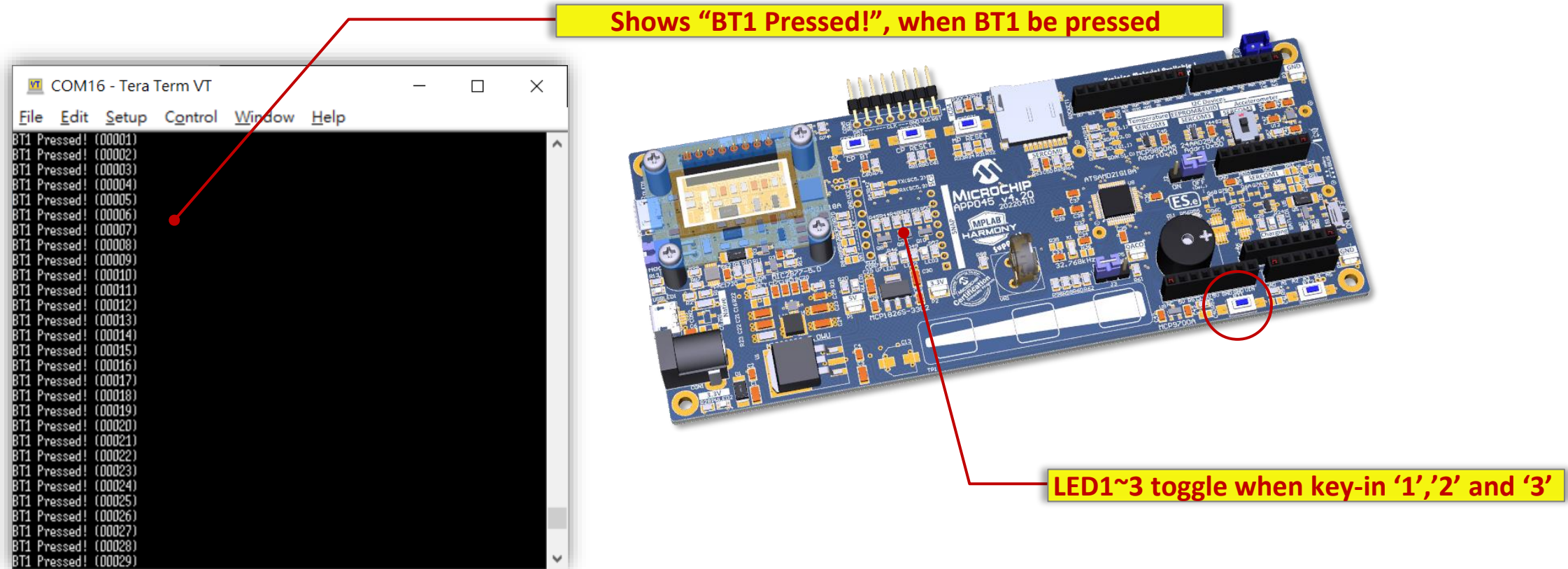
✂ Lab5 USB Device CDC Class - Basic TX RX

- VID/PID: 04D8/0000
- Product String Selection : “Lab5_USBDevice_CDC_Basic_TX&RX”



Lab5 USB Device CDC Class - Basic TX RX

- LED1 to LED3 be toggled by '1', '2' and '3' key, by terminal software (Tera Team).
- Shows "BT1 Pressed!" in terminal screen, when BT1 be pressed at APP045.



USB Device HID Class

USB HID Class

- HID (Human Interface Devices)
- HID include difference of Subclass and Protocol,
 - Sub Class Code,
 - 0x00 : No Subclass
 - 0x01 : Boot Interface Subclass
 - Protocol Code,
 - 0x00 : None
 - 0x01 : Keyboard
 - 0x03 : Mouse
 - 0x03 - 0xFF : Reserved
- HID Custom Application,
 - Sub Class Code = 0x00
 - Protocol Code = 0x00

```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // Descriptor Type is Interface descriptor
    0, // Interface Number
    0x00, // Alternate Setting Number
    0x02, // Number of endpoints in this interface
    USB_HID_CLASS_CODE, // Class code
    USB_HID_SUBCLASS_CODE_NO_SUBCLASS, // Subclass code
    USB_HID_PROTOCOL_CODE_NONE, // No Protocol
    ...
}
```

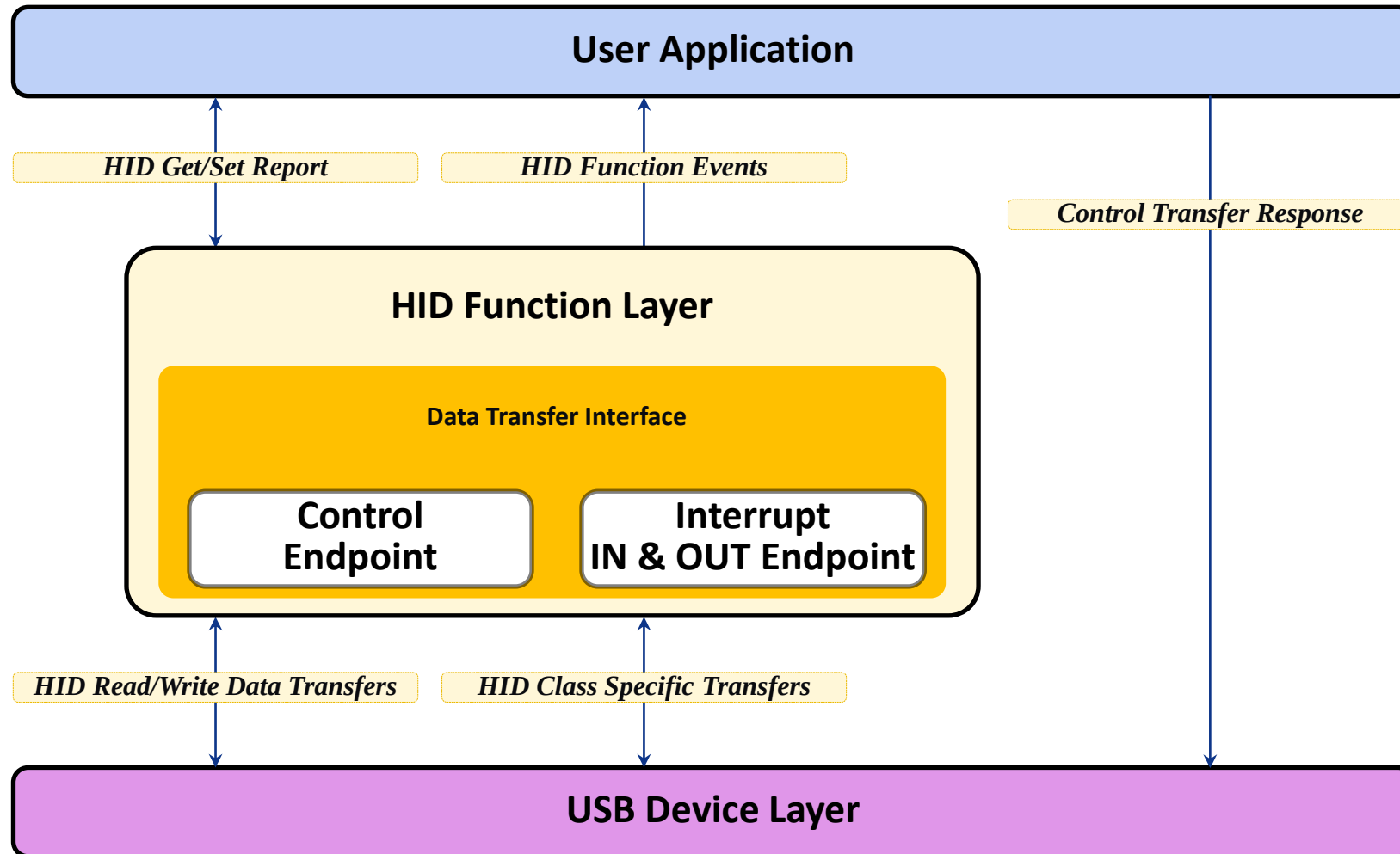
HID Usage Tables

- Usages are part of the HID Report descriptor and supply an application developer with information about what a control is measuring or reporting.
- In addition, a Usage tag can be used to indicate the vendor's suggested use for a specific control or group of controls.

HID Custom Application Report Descriptor Example

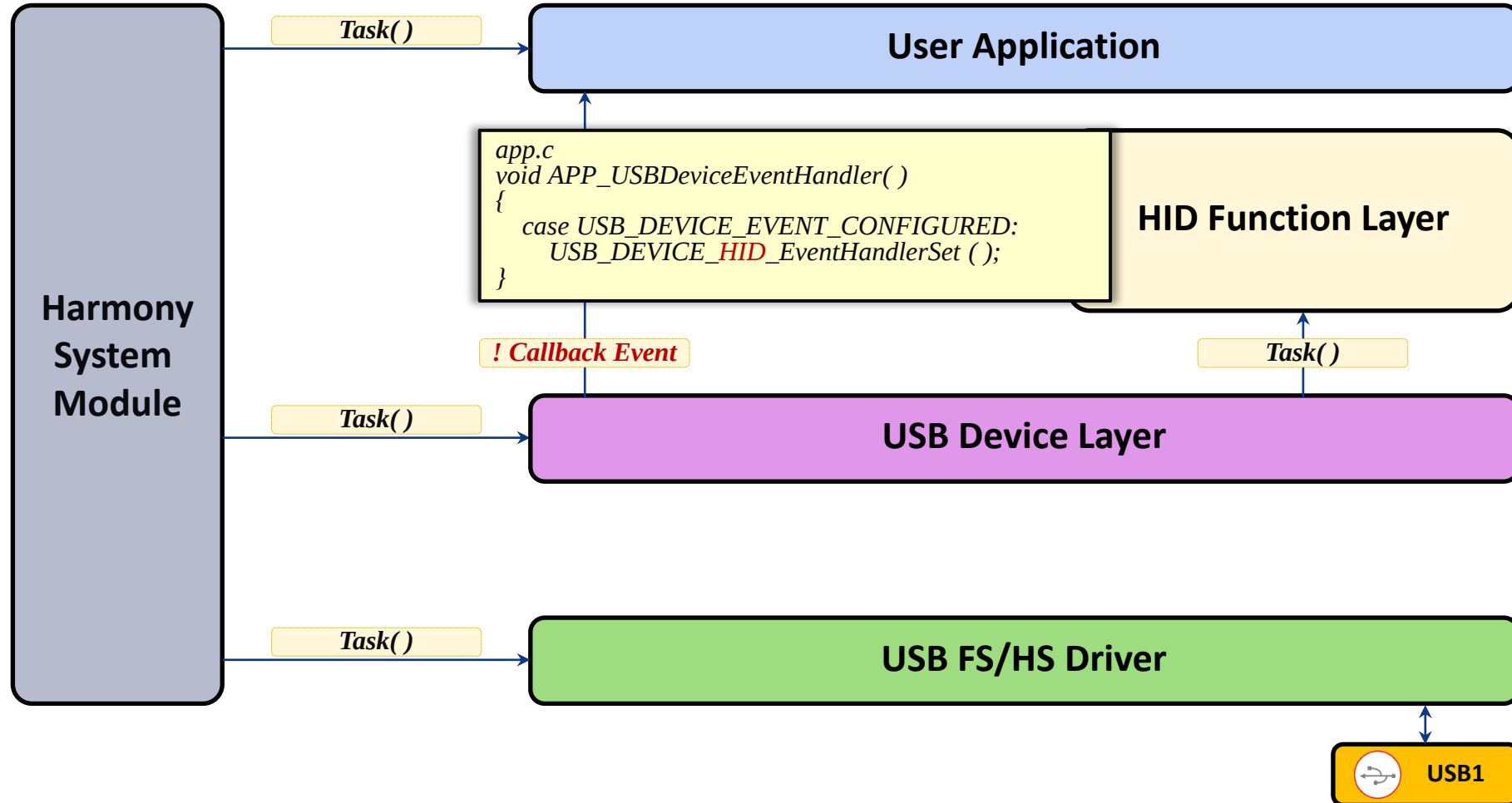
```
usb_device_init_data.c
const uint8_t hid_rpt0[] =
{
    0x06, 0x00, 0xFF, // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,      // Usage (Vendor Usage 1)
    0xA1, 0x01, // Colsslection (Application)
    0x19, 0x01, // Usage Minimum
    0x29, 0x40, // Usage Maximum //64 input usages total (0x01 to 0x40)
    0x15, 0x01, // Logical Minimum (data bytes in the report may have minimum value = 0x00)
    0x25, 0x40, // Logical Maximum (data bytes in the report may have maximum value = 0x00FF = unsigned 255)
    0x75, 0x08, // Report Size: 8-bit field size
    0x95, 0x40, // Report Count: Make sixty-four 8-bit fields (the next time the parser hits an "Input", "Output", or "Feature" item)
    0x81, 0x00, // Input (Data, Array, Abs):
                // Instantiates input packet fields based on the above report size, count, logical min/max, and usage.
    0x19, 0x01, // Usage Minimum
    0x29, 0x40, // Usage Maximum //64 output usages total (0x01 to 0x40)
    0x91, 0x00, // Output (Data, Array, Abs): Instantiates output packet fields.
                // Uses same report size and count as "Input" fields, since nothing new/different
                // was specified to the parser since the "Input" item.
    0xC0
};
```

USB HID Abstraction Model

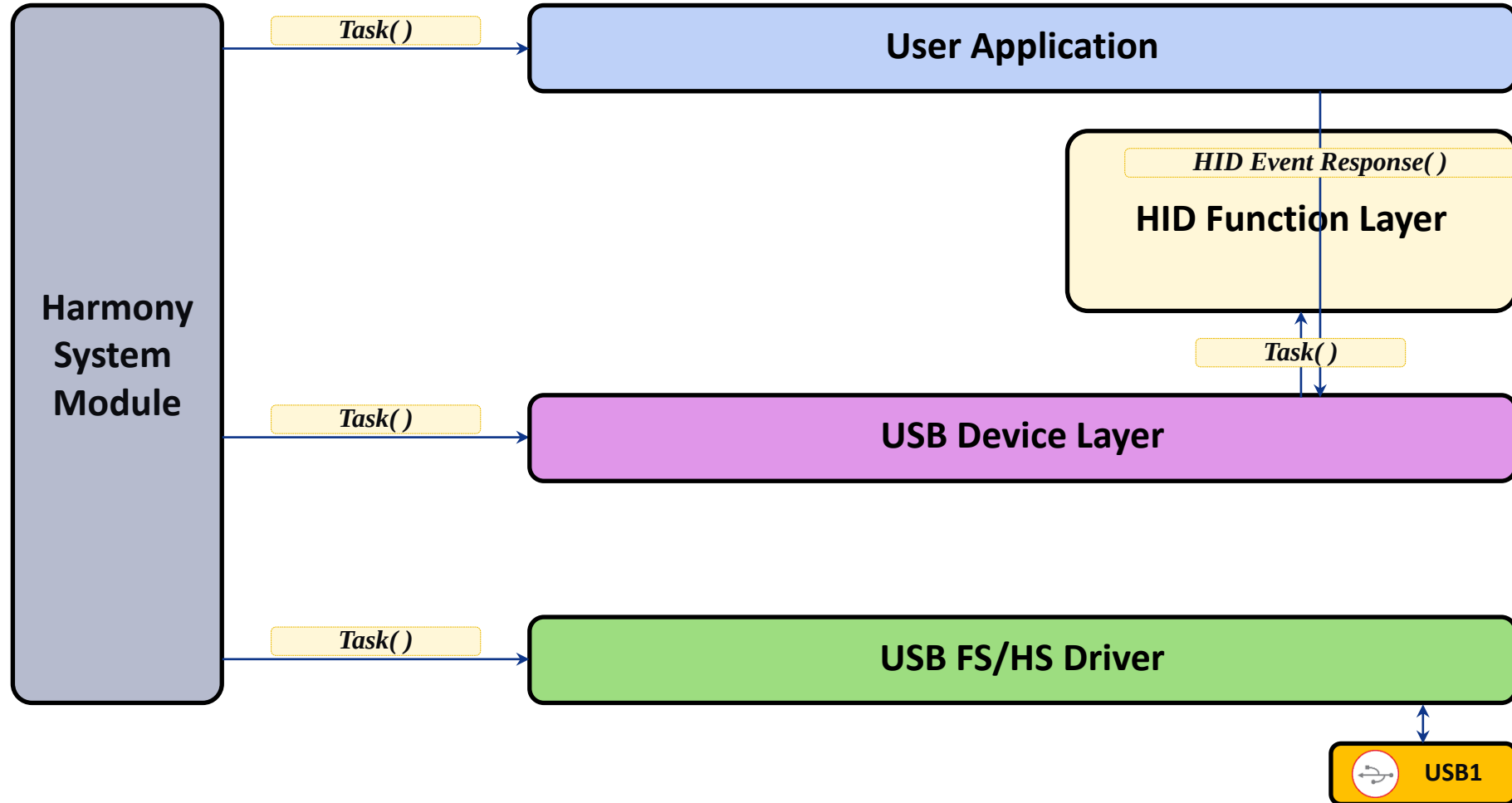


USB Stack Abstraction Model - Register Function Event Handler

- Register HID Function Event Handler only, after “Set Configuration” state.



USB Stack Abstraction Model - Register Function Event Handler



USB HID API

API functions in `usb_device_hid.c`

<code>_USB_DEVICE_HID_GlobalInitialize()</code>	Called by the device layer during Set Configuration Processing.
<code>_USB_DEVICE_HID_InitializeByDescriptorType()</code>	Initialize the HID function driver by interface.
<code>_USB_DEVICE_HID_ControlTransferHandler</code>	local function and should not be called directly by the application.
<code>_USB_DEVICE_HID_DeInitialize()</code>	
<code>_USB_DEVICE_HID_ReportReceiveCallBack()</code>	This callback forwards the event to application callback function.
<code>_USB_DEVICE_HID_ReportSendCallBack()</code>	This callback forwards the event to application callback function.
<code>USB_DEVICE_HID_TransferCancel()</code>	Cancels the HID transfer which has been submitted before.
<code>USB_DEVICE_HID_EventHandlerSet()</code>	Allows the application to register an event handler.
<code>USB_DEVICE_HID_ReportReceive()</code>	Receive a report from host.
<code>USB_DEVICE_HID_ReportSend()</code>	Send a report to host.

USB HID API - Event Handler Register

```
// Register the HID Function Driver Layer event handler.  
USB_DEVICE_HID_RESULT USB_DEVICE_HID_EventHandlerSet  
(  
    USB_DEVICE_HID_INDEX iHID,  
    USB_DEVICE_HID_EVENT_HANDLER eventHandler,  
    uintptr_t userData  
);
```

e.g. :

```
USB_DEVICE_HID_EventHandlerSet  
(  
    USB_DEVICE_HID_INDEX_0,  
    USBDeviceHIDEventHandler,  
    0  
);
```

USB HID API - Event Handler Register Example

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler
(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    switch (event)
    {
        ...
        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                RESULT USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, USBDeviceHIDEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

        ...
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

```
USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(USB_DEVICE_HID_INDEX instanceIndex, USB_DEVICE_HID_EVENT event,
void * pData, uintptr_t userData)
{
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_GET_REPORT: break;
        case USB_DEVICE_HID_EVENT_SET_REPORT: break;
        case USB_DEVICE_HID_EVENT_GET_PROTOCOL: break;
        case USB_DEVICE_HID_EVENT_SET_PROTOCOL: break;
        case USB_DEVICE_HID_EVENT_GET_IDLE: break;
        case USB_DEVICE_HID_EVENT_SET_IDLE: break;
        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_RECEIVED: break;
        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_DATA_SENT: break;
        case USB_DEVICE_HID_EVENT_CONTROL_TRANSFER_ABORTED: break;
        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED: break;
        case USB_DEVICE_HID_EVENT_REPORT_SENT: break;

        ...
    }
    return USB_DEVICE_HID_EVENT_RESPONSE_NONE;
}
```

USB HID - Events of HID Event Handler

- **USB_DEVICE_HID_EVENT_REPORT_RECEIVED**
 - HID report receive request has completed.
- **USB_DEVICE_HID_EVENT_REPORT_SENT**
 - HID report send request has completed.

USB HID API - USB_DEVICE_HID_ReportReceive

// Receive a report from host.

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportReceive  
(  
    USB_DEVICE_HID_INDEX iHID,  
    USB_DEVICE_HID_TRANSFER_HANDLE * transferHandle,  
    void * buffer, size_t size  
);
```

e.g. :

```
USB_DEVICE_HID_ReportReceive  
(  
    USB_DEVICE_HID_INDEX_0,  
    &appData.readTransferHandle,  
    appData.HIDrxDataBuffer, 64  
);
```

USB HID API - USB_DEVICE_HID_ReportSend

// Receive a report from host.

```
USB_DEVICE_HID_RESULT USB_DEVICE_HID_ReportSend  
(  
    USB_DEVICE_HID_INDEX iHID,  
    USB_DEVICE_HID_TRANSFER_HANDLE * transferHandle,  
    void * buffer, size_t size  
);
```

e.g. :

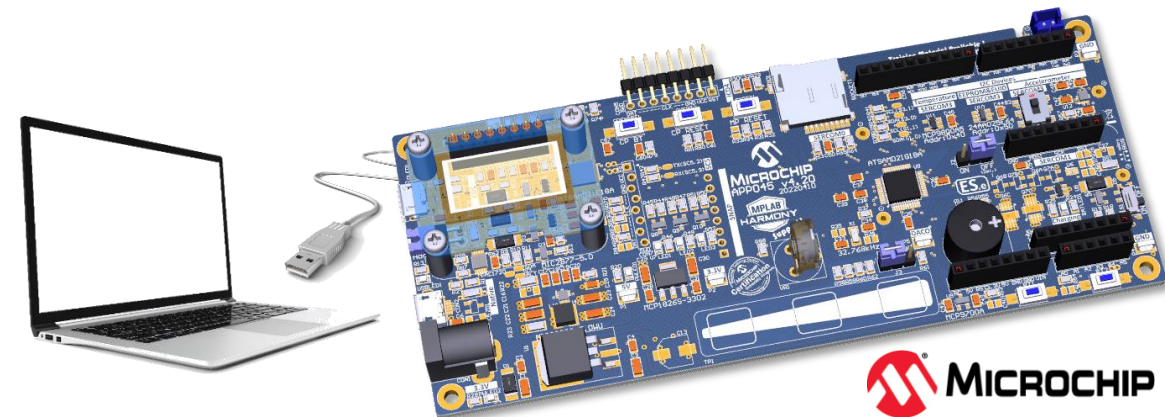
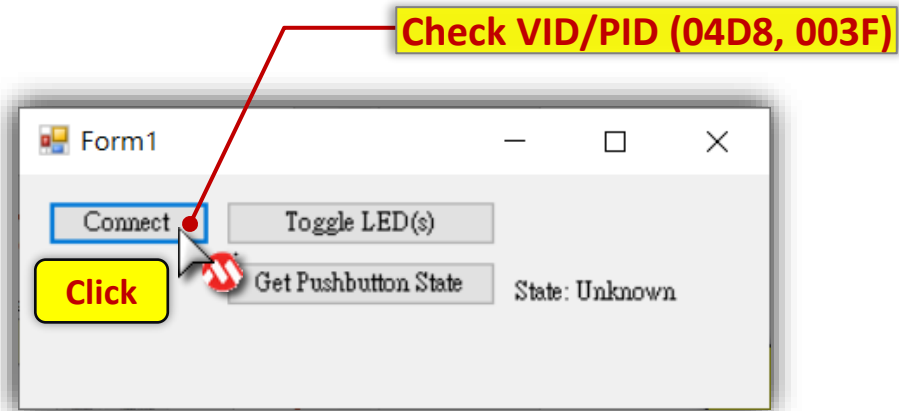
```
USB_DEVICE_HID_ReportSend  
(  
    USB_DEVICE_HID_INDEX_0,  
    &appData.writeTransferHandle,  
    appData.HIDtxDataBuffer, 64  
);
```

Lab6 USB Device HID Class - Basic TX and RX

Lab6 USB Device HID Class - Basic TX and RX

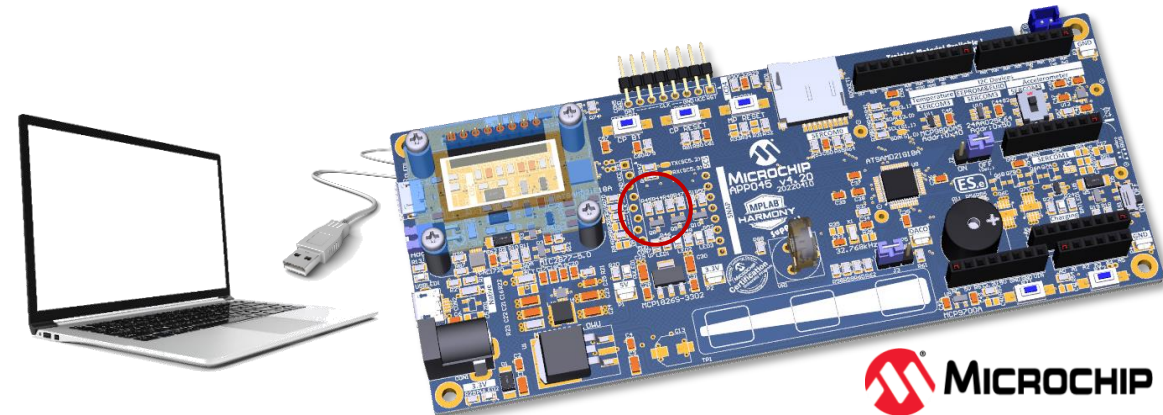
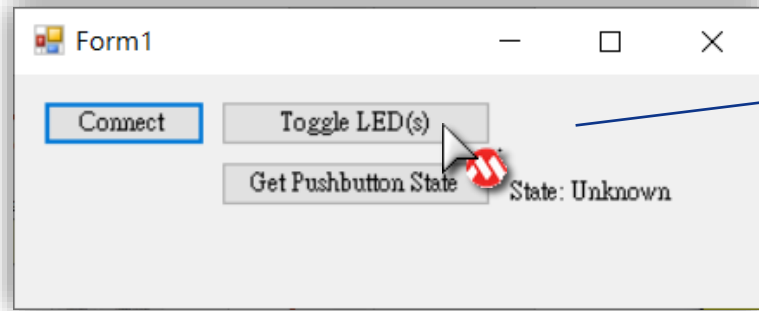
- At this lab, we try to add CDC function to our USB Device.
- The CDC function subclass is “No Subclass” (0x00) , protocol is “None” (0x00).
- Application require,
 - LED1 be toggle by ‘Toggle LED(s)’ GUI button at Generic HID Simple Demo.exe.
 - Show the BT1 status at GUI when click “Get pushbutton State” button.
- **Let's go !**

PC Application - Generic HID Simple Demo.exe

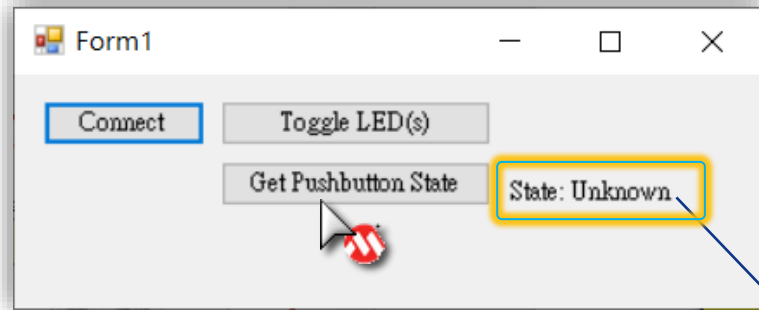


PC Application - Generic HID Simple Demo.exe

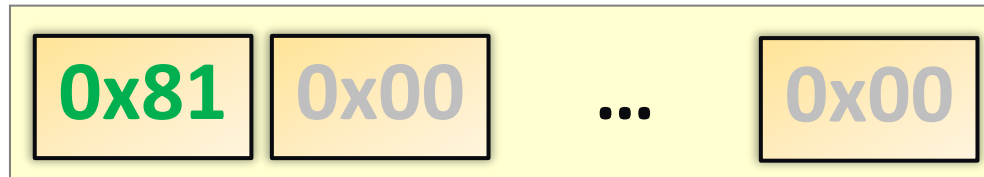
Toggle LED, (OUT Report)



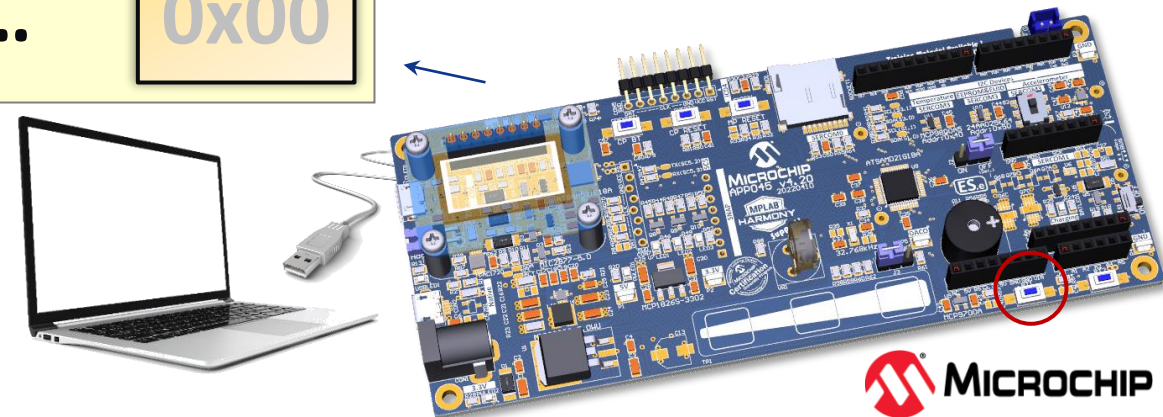
PC Application - Generic HID Simple Demo.exe



Get Push Button State (OUT Report)

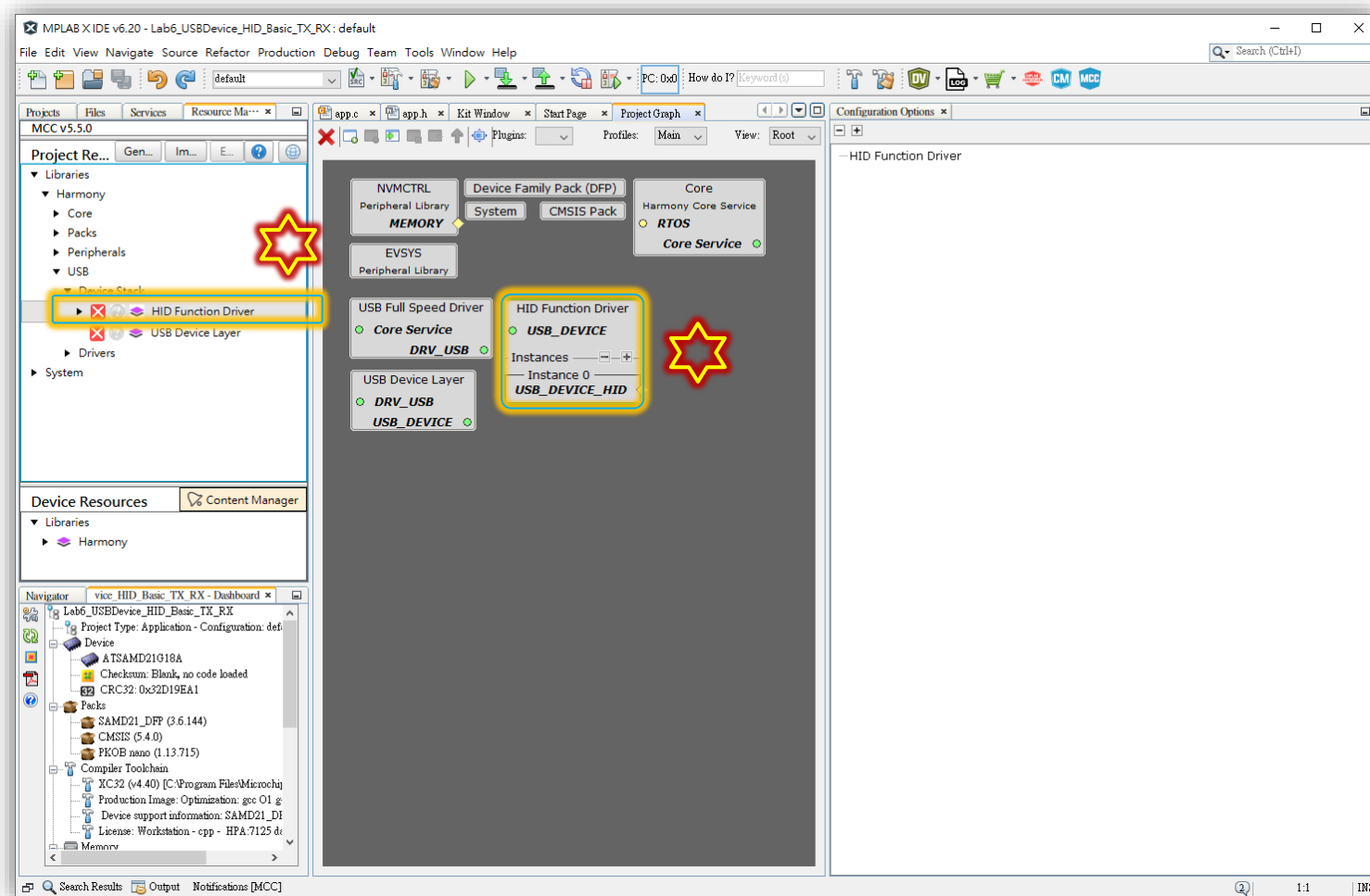


(IN Report)



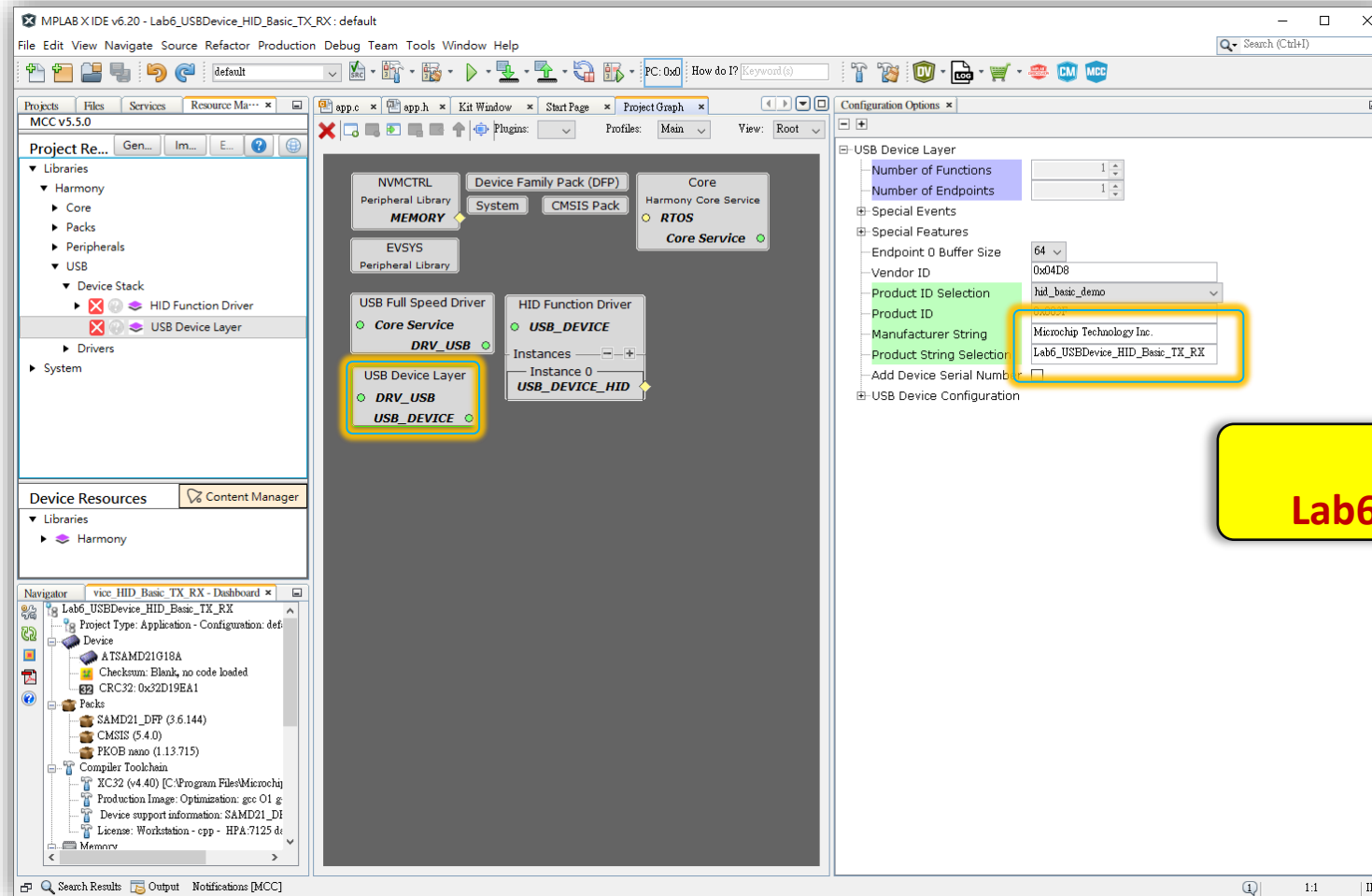
⚡ Lab6 USB Device HID Class - Basic TX and RX

- Basic on Lab4 and add HID Function Layer in MHC.



⚙️ Lab6 USB Device HID Class - Basic TX and RX

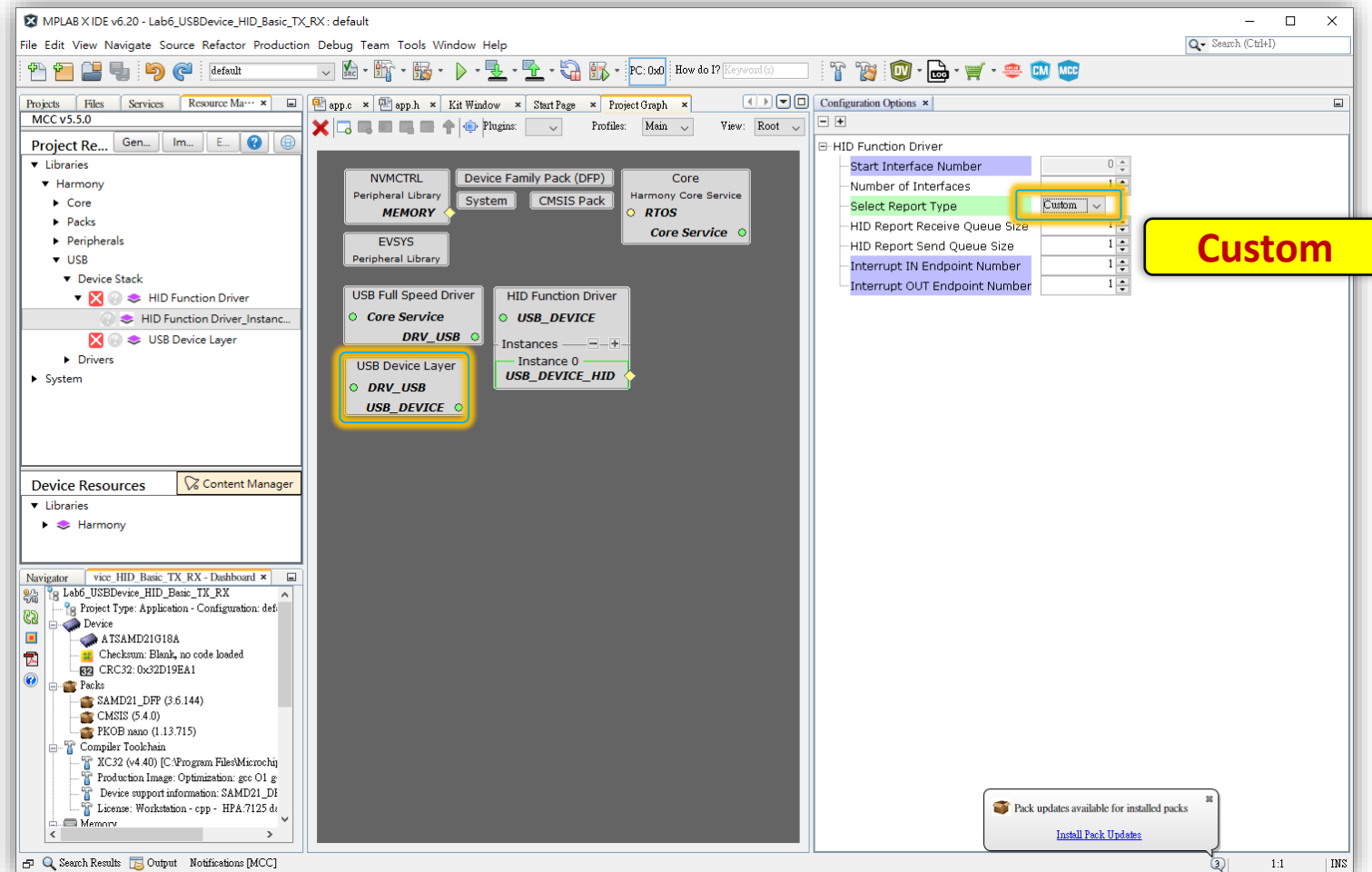
- Change VID/PID to : 0x04D8, 0x003F
“Product String Selection” to “Lab6_USBDevice_HID_Basic_TX_RX”



0x04D8, 0x003F
Lab6_USBDevice_HID_Basic_TX_RX

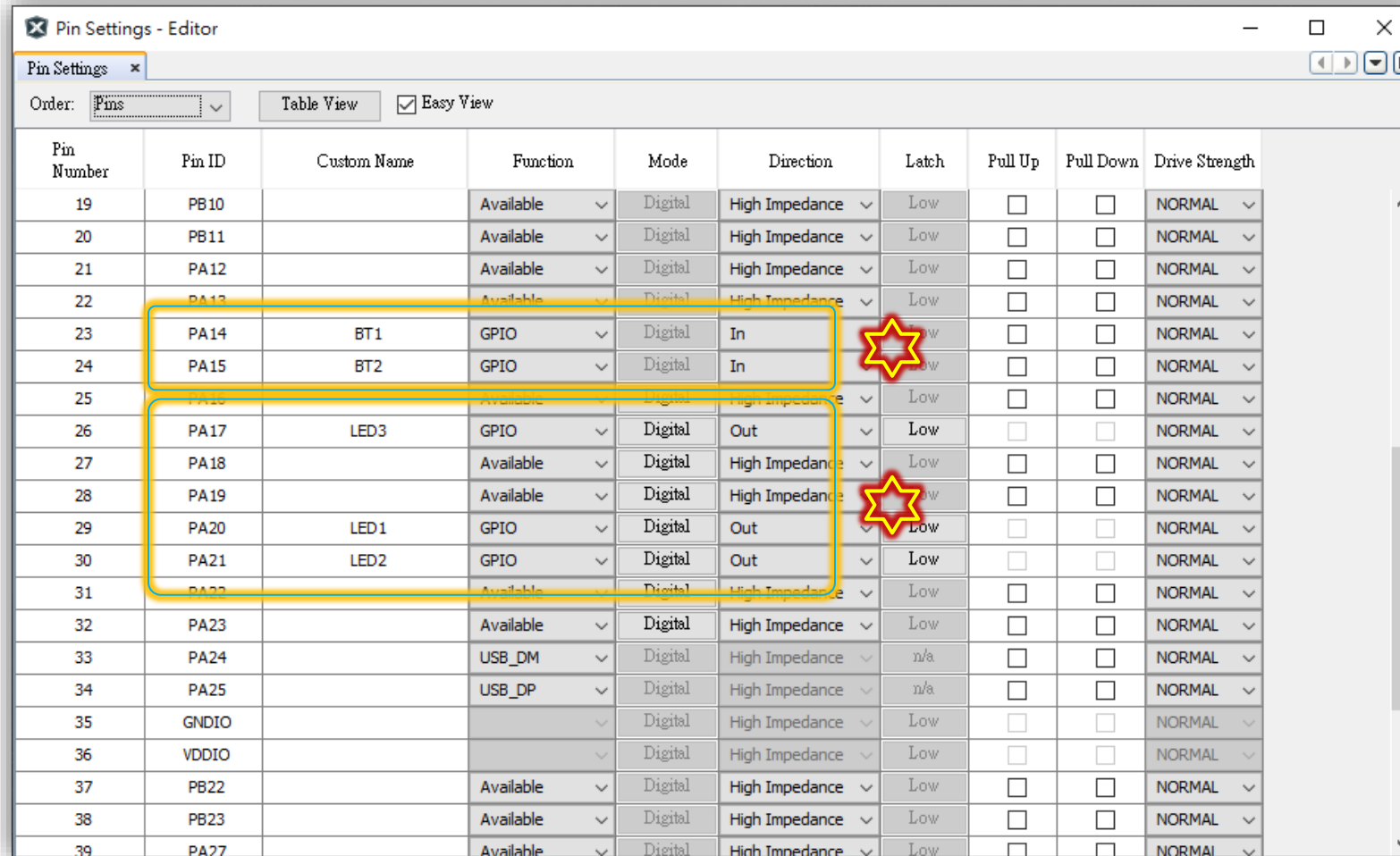
⚙️ Lab6 USB Device HID Class - Basic TX and RX

- Change Report Type to “Custom”



⚙️ Lab6 USB Device HID Class - Basic TX and RX

- Set LEDs and BTs in Pin Settings



Pin Settings - Editor

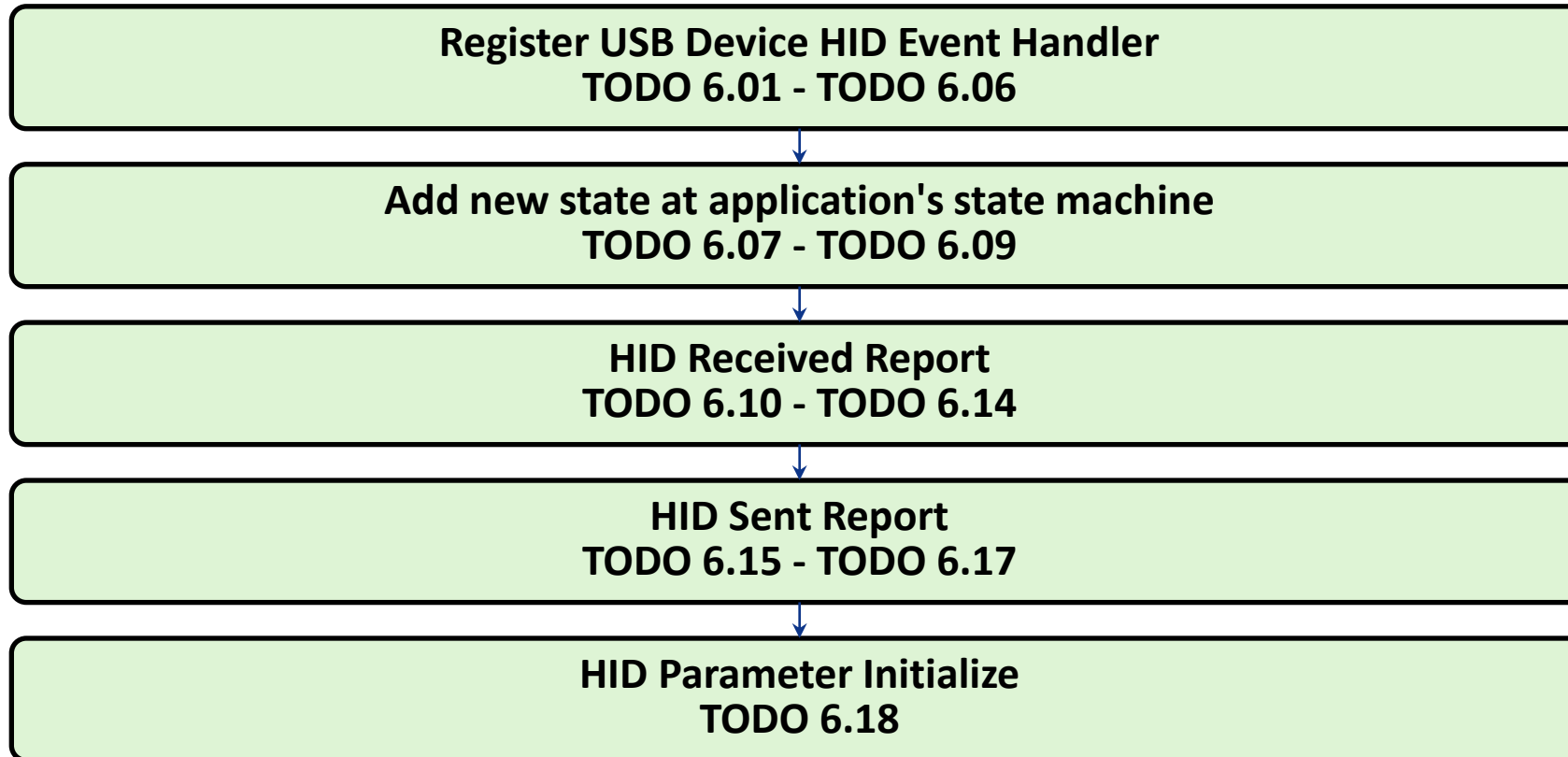
Pin Settings x

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
19	PB10		Available	Digital	High Impedance	Low	☐	☐	NORMAL
20	PB11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
21	PA12		Available	Digital	High Impedance	Low	☐	☐	NORMAL
22	PA13		Available	Digital	High Impedance	Low	☐	☐	NORMAL
23	PA14	BT1	GPIO	Digital	In	Low	☐	☐	NORMAL
24	PA15	BT2	GPIO	Digital	In	Low	☐	☐	NORMAL
25	PA16		Available	Digital	High Impedance	Low	☐	☐	NORMAL
26	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
27	PA18		Available	Digital	High Impedance	Low	☐	☐	NORMAL
28	PA19		Available	Digital	High Impedance	Low	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
30	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
31	PA22		Available	Digital	High Impedance	Low	☐	☐	NORMAL
32	PA23		Available	Digital	High Impedance	Low	☐	☐	NORMAL
33	PA24		USB_DM	Digital	High Impedance	n/a	☐	☐	NORMAL
34	PA25		USB_DP	Digital	High Impedance	n/a	☐	☐	NORMAL
35	GNDIO			Digital	High Impedance	Low	☐	☐	NORMAL
36	VDDIO			Digital	High Impedance	Low	☐	☐	NORMAL
37	PB22		Available	Digital	High Impedance	Low	☐	☐	NORMAL
38	PB23		Available	Digital	High Impedance	Low	☐	☐	NORMAL
39	PA27		Available	Digital	High Impedance	Low	☐	☐	NORMAL

Lab6 USB Device HID Class - Basic TX and RX

- Code Implementation state



🔧 Lab6 USB Device HID Class - Basic TX and RX

- Register USB Device HID Event Handler

```
// TODO 6.01
// TODO 6.02
USB_DEVICE_HID_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData,
    uintptr_t userData
){}
```

```
// TODO 6.05
#include "usb/usb_device_hid.h"

USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData, uintptr_t userData
)
{
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_GET_IDLE:
            // TODO 6.03
            USB_DEVICE_ControlSend(appData.usbDevHandle, &currentIdleRate, 1);
            break;
        case USB_DEVICE_HID_EVENT_SET_IDLE:
            // TODO 6.04
            currentIdleRate = ((USB_DEVICE_HID_EVENT_DATA_SET_IDLE*) pData)->duration;
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;
        default:
            break;
    }

    return USB_DEVICE_HID_EVENT_RESPONSE_NONE;
}
```

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;

    // Handling of each event
    switch (event)
    {
        ...

        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                //TODO 6.06
                USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, USBDeviceHIDEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

        ...
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

⚙️ Lab6 USB Device HID Class - Basic TX and RX

- Add new state at application's state machine

```
typedef enum
{
    /* Application's state machine's initial state. */
    // TODO 6.07
    //APP_STATE_INIT = 0,
    //APP_STATE_SERVICE_TASKS,
    APP_STATE_INIT = 0,
    APP_STATE_WAIT_CONFIGURED,
    APP_STATE_SERVICE_TASKS,
    /* TODO: Define states used by the application state machine. */
} APP_STATES;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_INIT:
        {
            if (appInitialized)
            {
                USB_DEVICE_EventHandlerSet(appData.usbDevHandle, APP_USBDeviceEventHandler, 0);

                //TODO 6.08
                appData.state = APP_STATE_WAIT_CONFIGURED;
            }
            break;
        }

        //TODO 6.09
        case APP_STATE_WAIT_CONFIGURED:
        {
            break;
        }

        case APP_STATE_SERVICE_TASKS:
        {
            break;
        }

        default:
        {
            break;
        }
    }
}
```

⚡ Lab6 USB Device HID Class - Basic TX and RX

- HID Received Report

```
// TODO 6.11
#include "usb/usb_device_hid.h"

typedef struct
{
    /* The application's current state */
    APP_STATES state;

    //TODO 6.10
    volatile bool HIDReportReceived;
    USB_DEVICE_HID_TRANSFER_HANDLE ReceivedTransferHandle;
    uint8_t HIDReceivedDataBuffer[64] __attribute__((aligned(32)));
} APP_DATA;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_WAIT_CONFIGURED:
        {
            // TODO 6.13
            if (appData.deviceIsConfigured)
            {
                appData.HIDReportReceived = false;
                USB_DEVICE_HID_ReportReceive
                (USB_DEVICE_HID_INDEX_0, &appData.ReceivedTransferHandle, appData.HIDReceivedDataBuffer, 64);
                appData.state = APP_STATE_SERVICE_TASKS;
            }
            break;
        }
    }
}
```

```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData, uintptr_t userData
)
{
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_REPORT_RECEIVED:
            // TODO 6.12
            appData.HIDReportReceived = true;
            break;
    }

    return USB_DEVICE_HID_EVENT_RESPONSE_NONE;
}
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_SERVICE_TASKS:
        {
            // TODO 6.14
            if (appData.HIDReportReceived)
            {
                switch (appData.HIDReceivedDataBuffer[0])
                {
                    case 0x80: LED1_Toggle();
                    break;
                }

                appData.HIDReportReceived = false;
                USB_DEVICE_HID_ReportReceive
                (USB_DEVICE_HID_INDEX_0, &appData.ReceivedTransferHandle, appData.HIDReceivedDataBuffer, 64);
            }
            ...
        }
    }
}
```

⚡ Lab6 USB Device HID Class - Basic TX and RX

- HID Sent Report

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;

    // TODO 6.15
    volatile bool HIDReportSent;
    USB_DEVICE_HID_TRANSFER_HANDLE SentTransferHandle;
    uint8_t HIDSentDataBuffer[64] __attribute__((aligned(32)));
} APP_DATA;
```

```
void APP_Tasks(void)
{
    switch (appData.state)
    {
        case APP_STATE_SERVICE_TASKS:
        {
            // TODO 6.14
            if (appData.HIDReportReceived)
            {
                switch (appData.HIDReceivedDataBuffer[0])
                {
                    case 0x80: LED1_Toggle();
                        break;

                    // TODO 6.17
                    case 0x81:
                    {
                        LED2_Toggle();
                        appData.HIDReportSent = false;
                        appData.HIDSentDataBuffer[0] = 0x81;
                        appData.HIDSentDataBuffer[1] = BT1_Get() & 0x01;
                        USB_DEVICE_HID_ReportSend(USB_DEVICE_HID_INDEX_0, &appData.SentTransferHandle, appData.HIDSentDataBuffer, 64);
                    }
                        break;
                }
            }

            appData.HIDReportReceived = false;
            USB_DEVICE_HID_ReportReceive
            (USB_DEVICE_HID_INDEX_0, &appData.ReceivedTransferHandle, appData.HIDReceivedDataBuffer, 64);
        }
        ...
    }
}
```

```
USB_DEVICE_CDC_EVENT_RESPONSE USBDeviceHIDEventHandler
(
    USB_DEVICE_HID_INDEX instanceIndex,
    USB_DEVICE_HID_EVENT event,
    void * pData, uintptr_t userData
)
{
    switch (event)
    {
        case USB_DEVICE_HID_EVENT_REPORT_SENT:
            // TODO 6.16
            appData.HIDReportSent = true;
            break;
    }

    return USB_DEVICE_HID_EVENT_RESPONSE_NONE;
}
```

Lab6 USB Device HID Class - Basic TX and RX

- HID Parameter Initialize

```
USB_DEVICE_EVENT_RESPONSE APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pData, uintptr_t context)
{
    uint8_t activeConfiguration;

    // Handling of each event
    switch (event)
    {
        ...
        case USB_DEVICE_EVENT_DECONFIGURED:
            break;

        case USB_DEVICE_EVENT_ERROR:
            break;

        case USB_DEVICE_EVENT_CONFIGURED:
            activeConfiguration = ((USB_DEVICE_EVENT_DATA_CONFIGURED *) (pData))->configurationValue;
            if (activeConfiguration == 1)
            {
                // TODO 6.18
                appData.HIDReportReceived = false;
                appData.HIDReportSent = false;

                // TODO 6.06
                USB_DEVICE_HID_EventHandlerSet(USB_DEVICE_HID_INDEX_0, USBDeviceHIDEventHandler, 0);
                appData.deviceIsConfigured = true;
            }
            break;

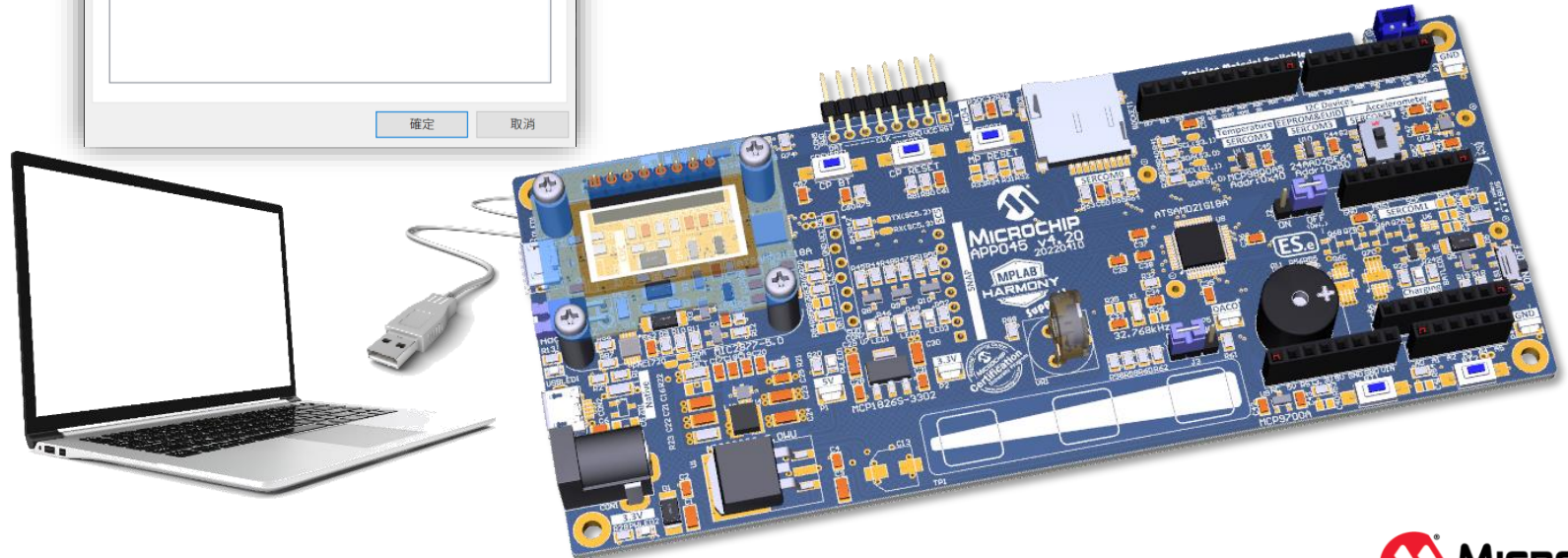
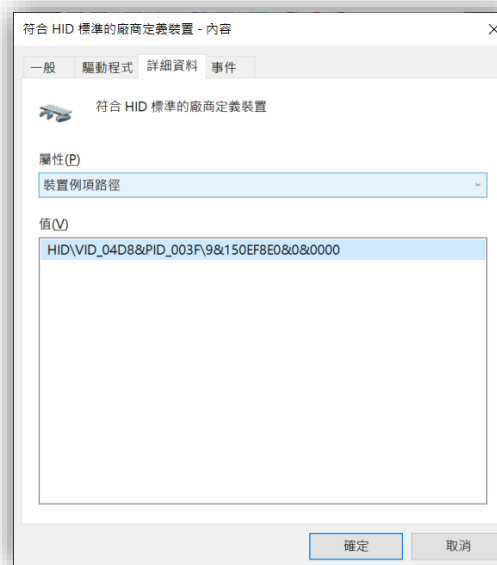
        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            break;

        ...
        return USB_DEVICE_EVENT_RESPONSE_NONE;
    }
}
```

✂ Lab6 USB Device HID Class - Basic TX and RX

- The device list at Device Manager, Named “符合 HID 標準的廠商定義裝置” After configured.

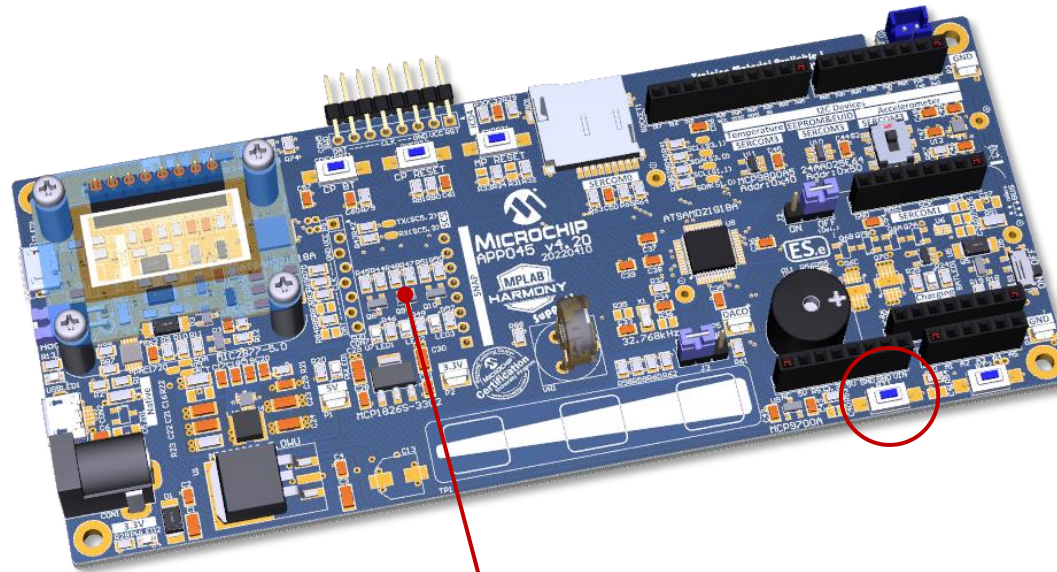


⚡ Lab6 USB Device HID Class - Basic TX and RX

- LED1 be toggle by click 'Toggle LED(s)' GUI button at Generic HID Simple Demo.exe.
- Show the BT1 status at GUI when click "Get pushbutton State" button.



Shows BT1 status when BT1 be pressed



LED1 toggle by click 'Toggle LED(s)'

Microchip Trademarks

Overview

Microchip Technology Incorporated's products are marketed and sold under one or more of the following trademarks belonging to Microchip or one of Microchip's subsidiaries. Questions regarding use of these trademarks should be addressed to the Microchip's Legal Department via email: legal.department@microchip.com.

The following are registered trademarks of Microchip in the U.S.A. and other countries:

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Klear, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA

The following are registered trademarks of Microchip in the U.S.A:

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, and ZL

The following are registered trademarks of Microchip in other countries: BeaconThings, GestIC, Silicon Storage Technology

The following are trademarks of Microchip in the U.S.A. and other countries:

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICKit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA

The following is a service mark of Microchip in the U.S.A.: SQTP

The following are logos of Microchip in the U.S.A. and other countries: The Adaptec logo, Frequency on Demand, Silicon Storage Technology, Symmcom, and Trusted Time

The following is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries: GestIC

