

**SAMD21 ARM Cortex-M0+
Microcontroller & Harmony v3
SAM2002_{ADV} v1.00**



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



SMART | CONNECTED | SECURE

**CAE Taiwan Team
Microchip Technology Inc.**

February 2023

SAM series course history and brief

- **SAM1001** (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
- **SAM2001** (2019~2020)
 - Peripheral Library (PLIB) on APP045v2.0/v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- **SAM2001ADV** (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
- **SAM2002** (2019~2021)
 - System Service and Driver on APP045v4.x (SAM D21 Arm Cortex-M0+)
 - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- **SAM2002ADV** (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

Advance Course

- [Introduction to QTouch® Peripheral Touch Controller \(PTC\)](#)
- [Harmony Touch Configurator](#)
- [APP045 EVB QTouch® Design and Pattern](#)
 -  [Lab1 Qtouch Buttons](#)
 -  [Lab2 Qtouch 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](#)
 -  [Lab5 USB Device HID Class - Basic TX and RX](#)

CAE空中教室 SAM2002ADV系列-01
SAMD21 ARM Cortex-M0+
Microcontroller & Harmony v3
SAM2002ADV v1.00



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



SMART | CONNECTED | SECURE

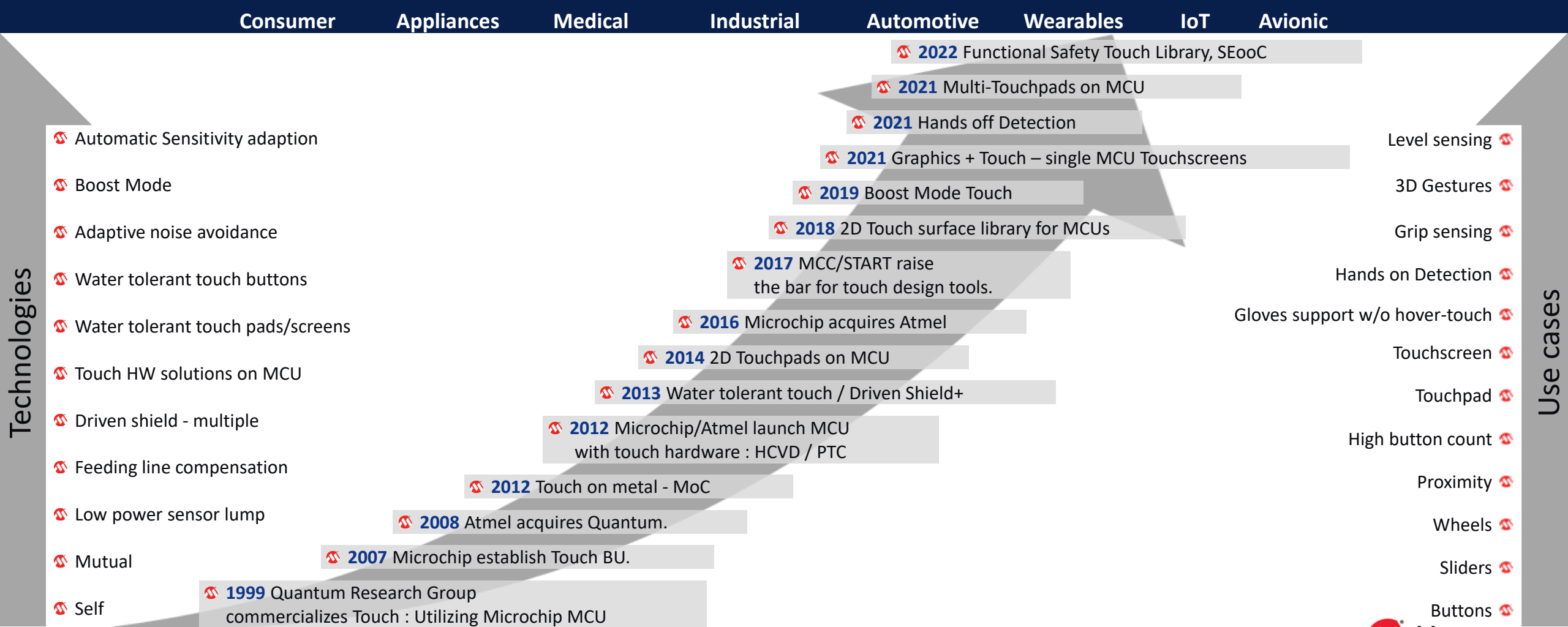
CAE Taiwan Team
Microchip Technology Inc.
February 20, 2023

Introduction to QTouch® Peripheral Touch Controller (PTC)

Introduction to QTouch® Peripheral Touch Controller (PTC)

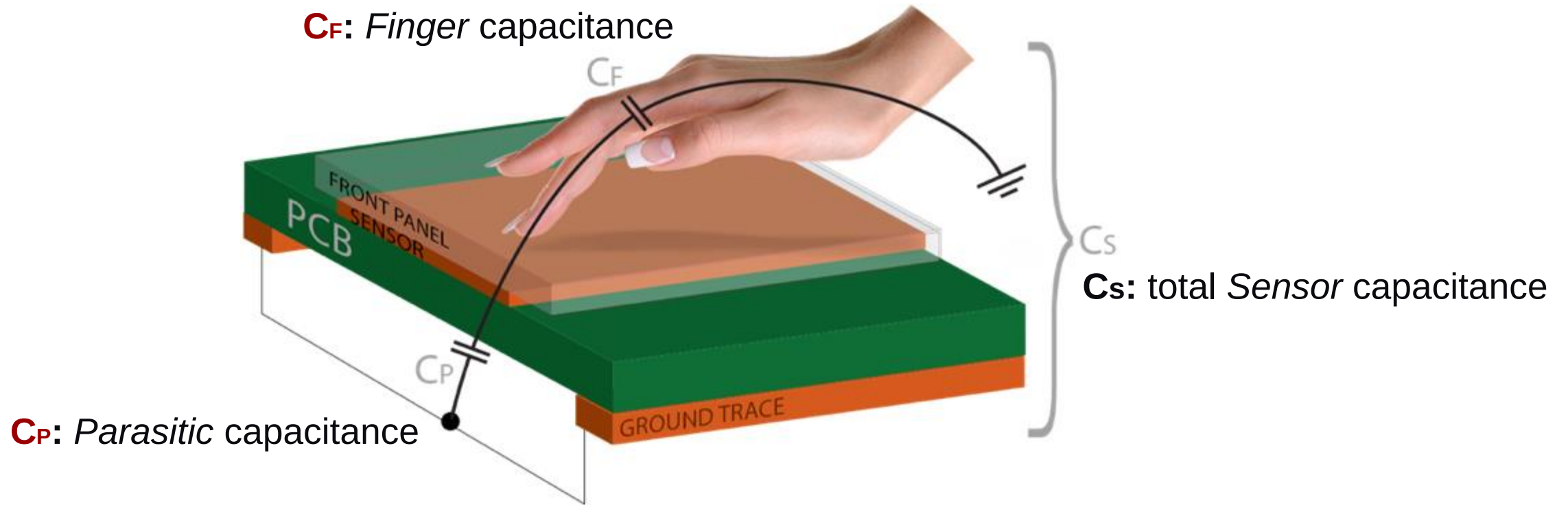
Long-Term innovative Player in Touch

Microchip serves all Markets and all use cases for touch.



Introduction to QTouch® Peripheral Touch Controller (PTC)

Capacitive Touch Sensor

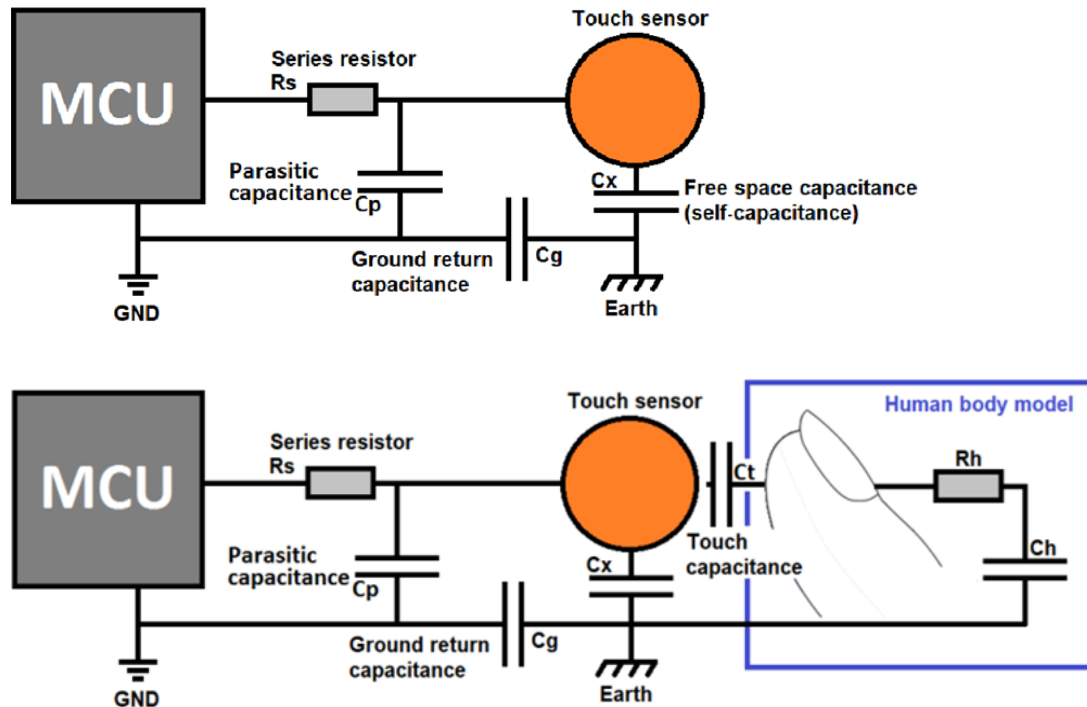


$$\text{Sensor Capacitance } (C_S) = C_P + C_F$$

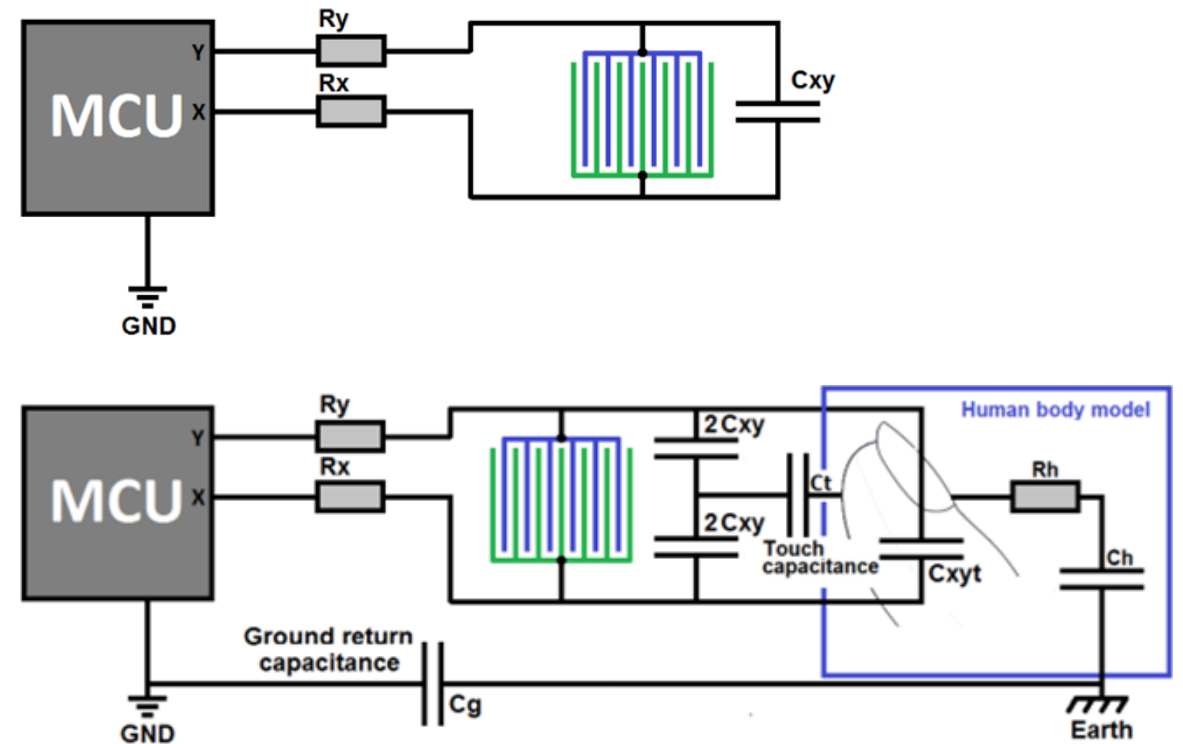
Introduction to QTouch® Peripheral Touch Controller (PTC)

Capacitance Touch Measurement

Self Capacitance Touch



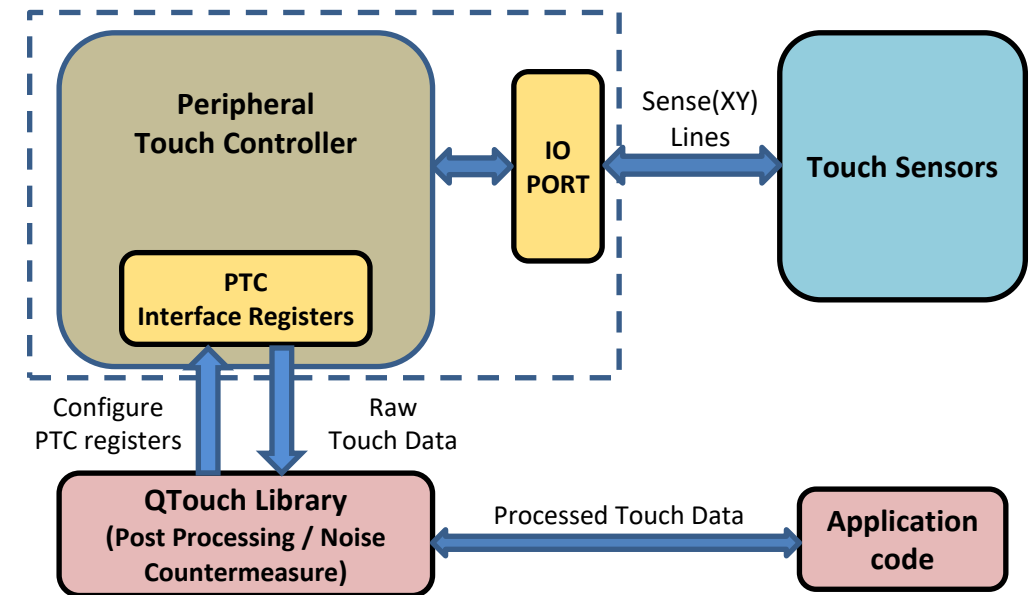
Mutual Capacitance Touch



Introduction to QTouch® Peripheral Touch Controller (PTC)

Peripheral Touch Controller (PTC)

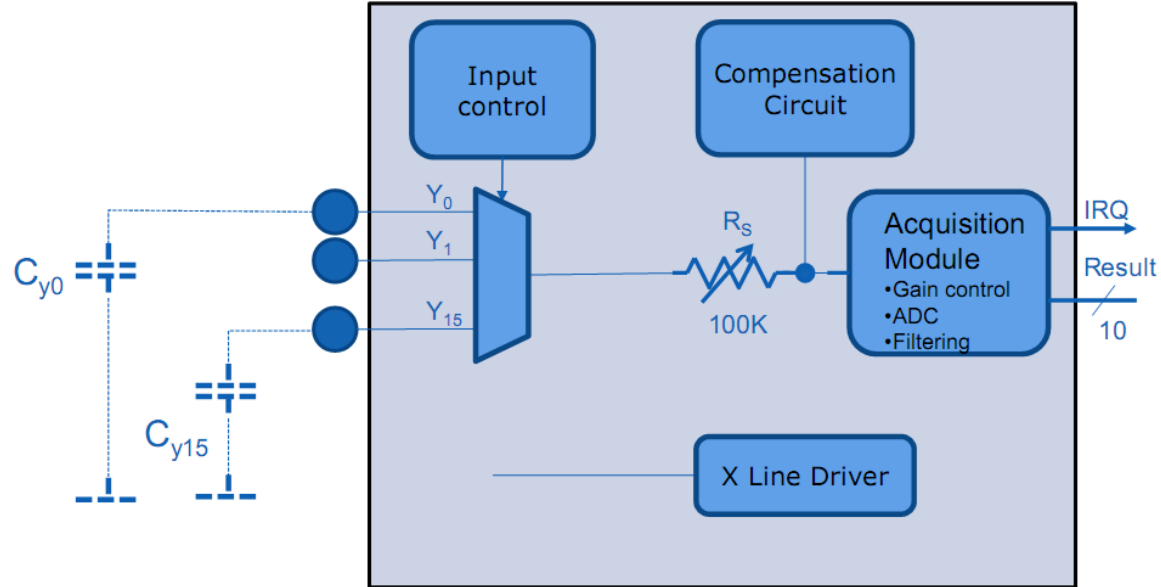
- The Peripheral Touch Controller (PTC) acquires signals in order to detect touch on capacitive sensors.
- The external capacitive touch sensor is typically formed on a PCB, and the sensor electrodes are connected to the analog front end of the PTC through the I/O pins in the device.
- The PTC supports 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.



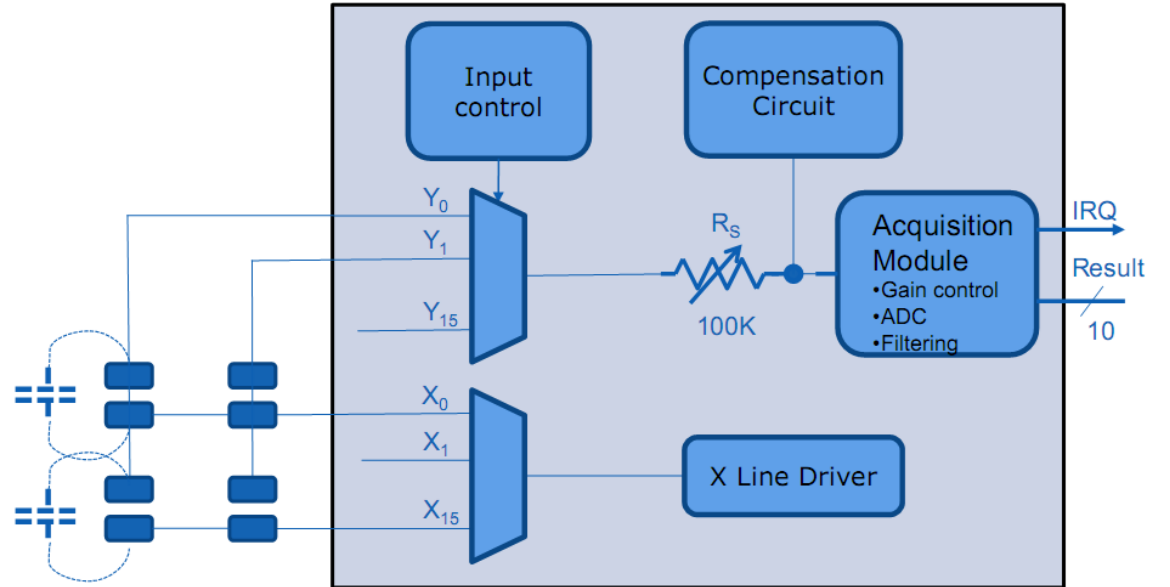
Introduction to QTouch® Peripheral Touch Controller (PTC)

PTC Block Diagram

Self Capacitance Touch



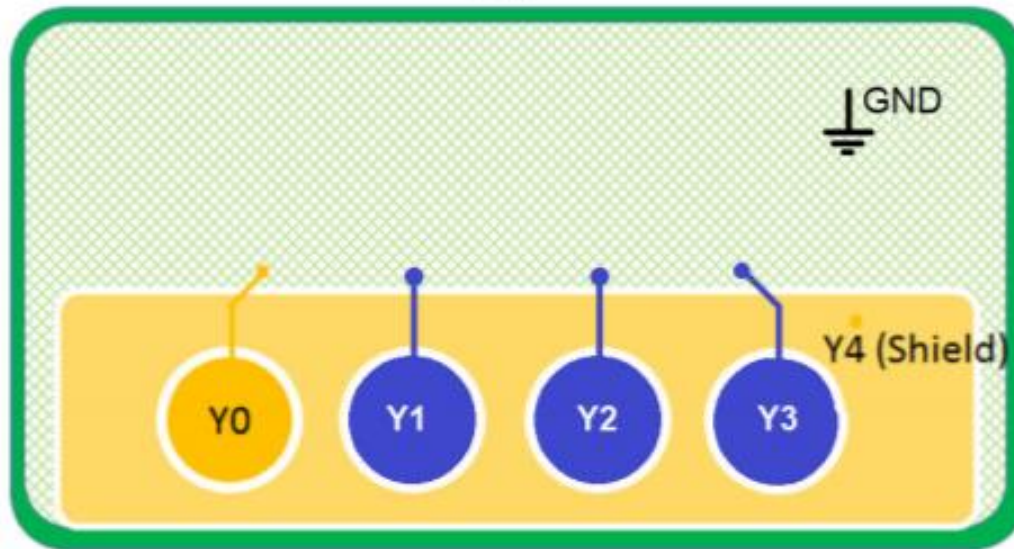
Mutual Capacitance Touch



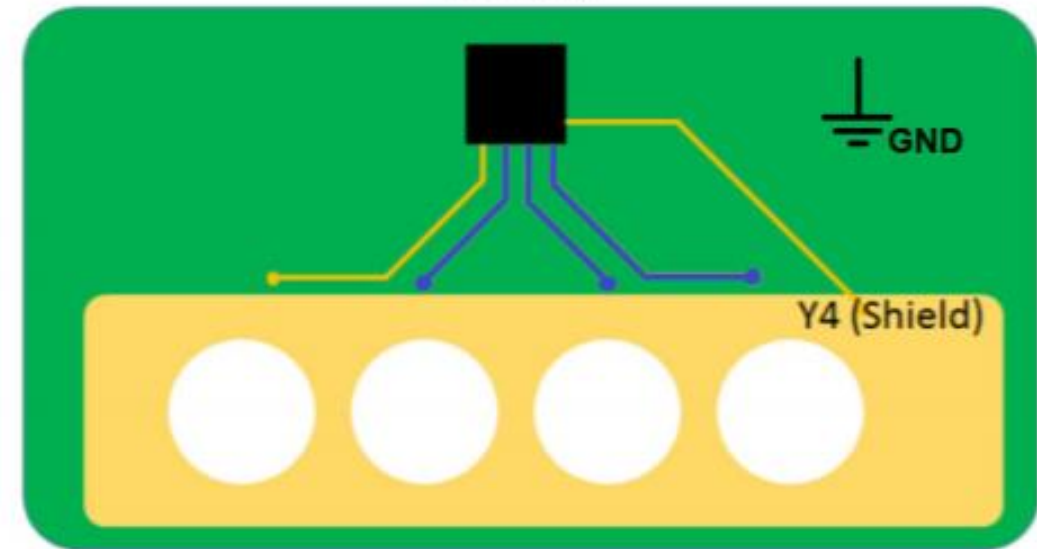
Introduction to QTouch® Peripheral Touch Controller (PTC)

Driven Shield

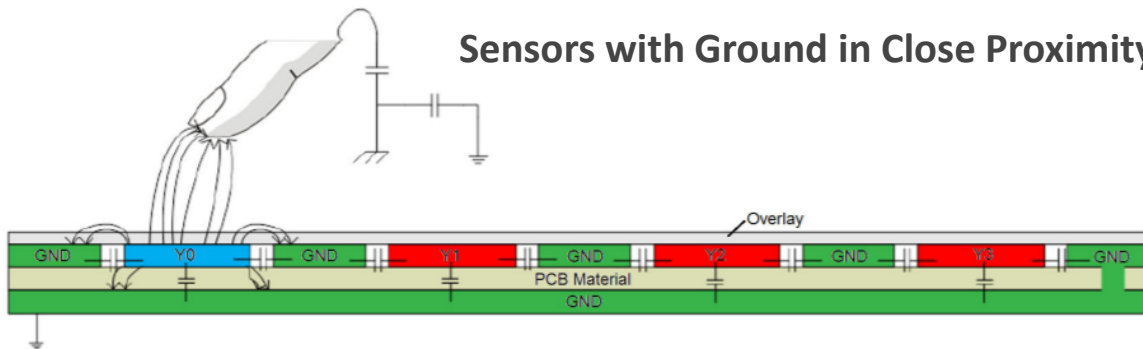
Top



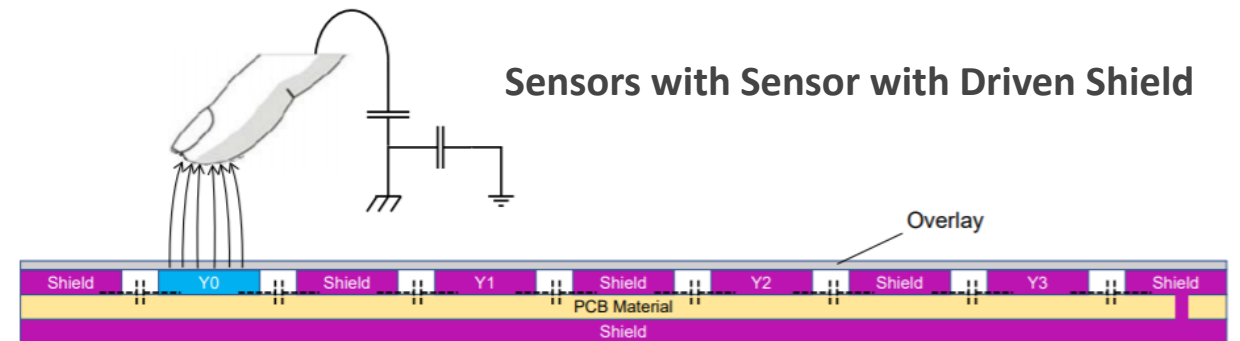
Bottom



Sensors with Ground in Close Proximity



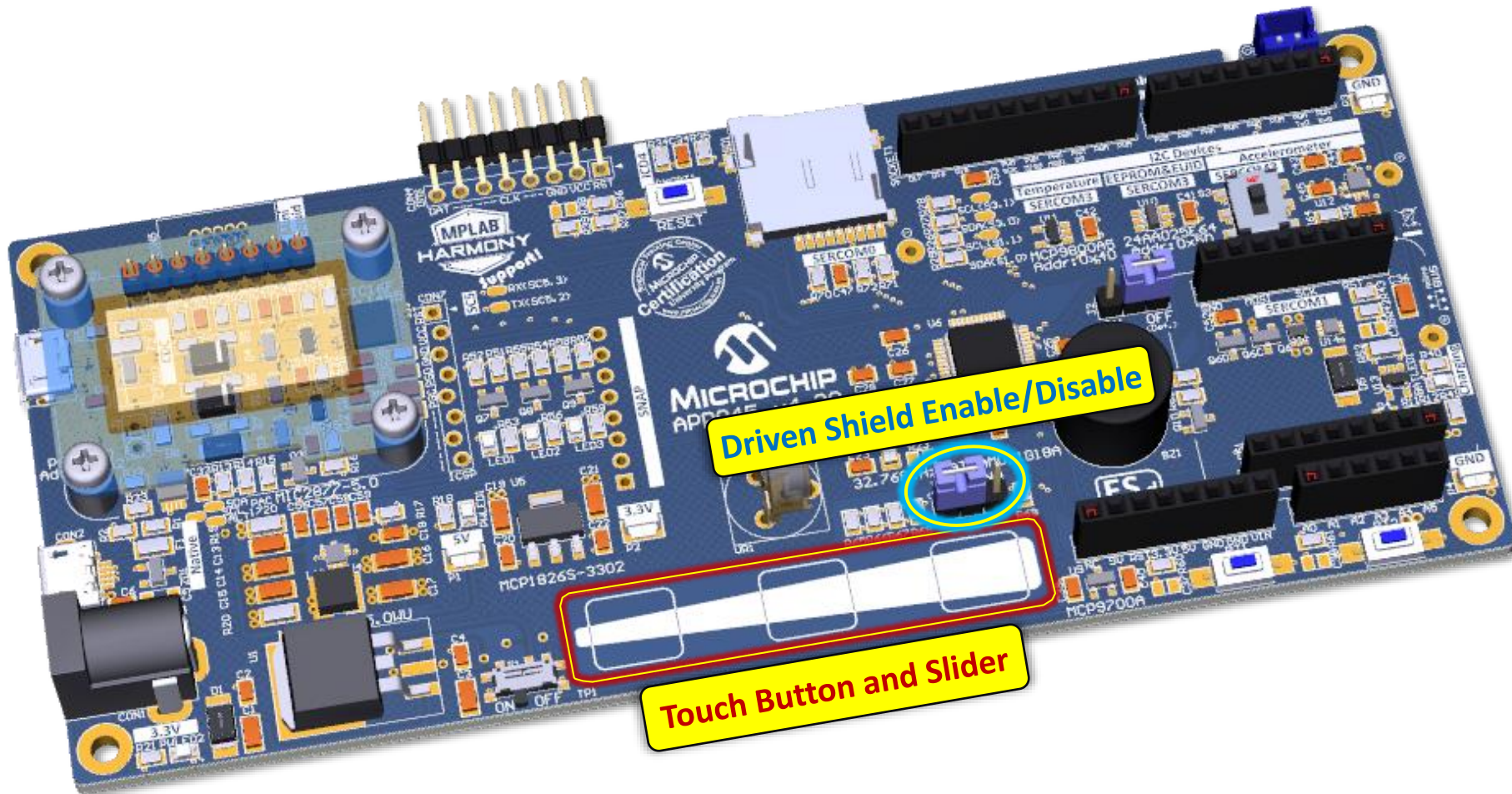
Sensors with Sensor with Driven Shield



APP045 EVB QTouch® Design and Pattern

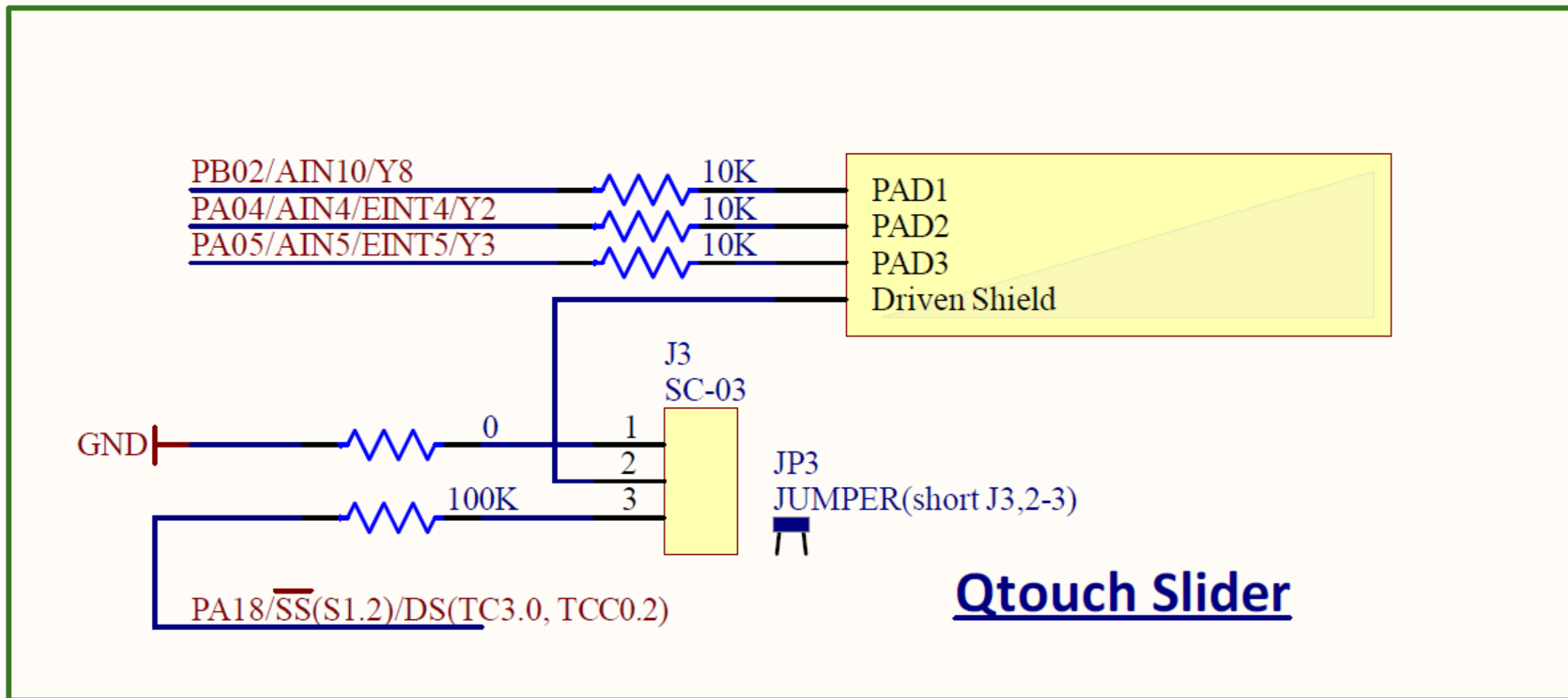
APP045 EVB QTouch® Design and Pattern

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



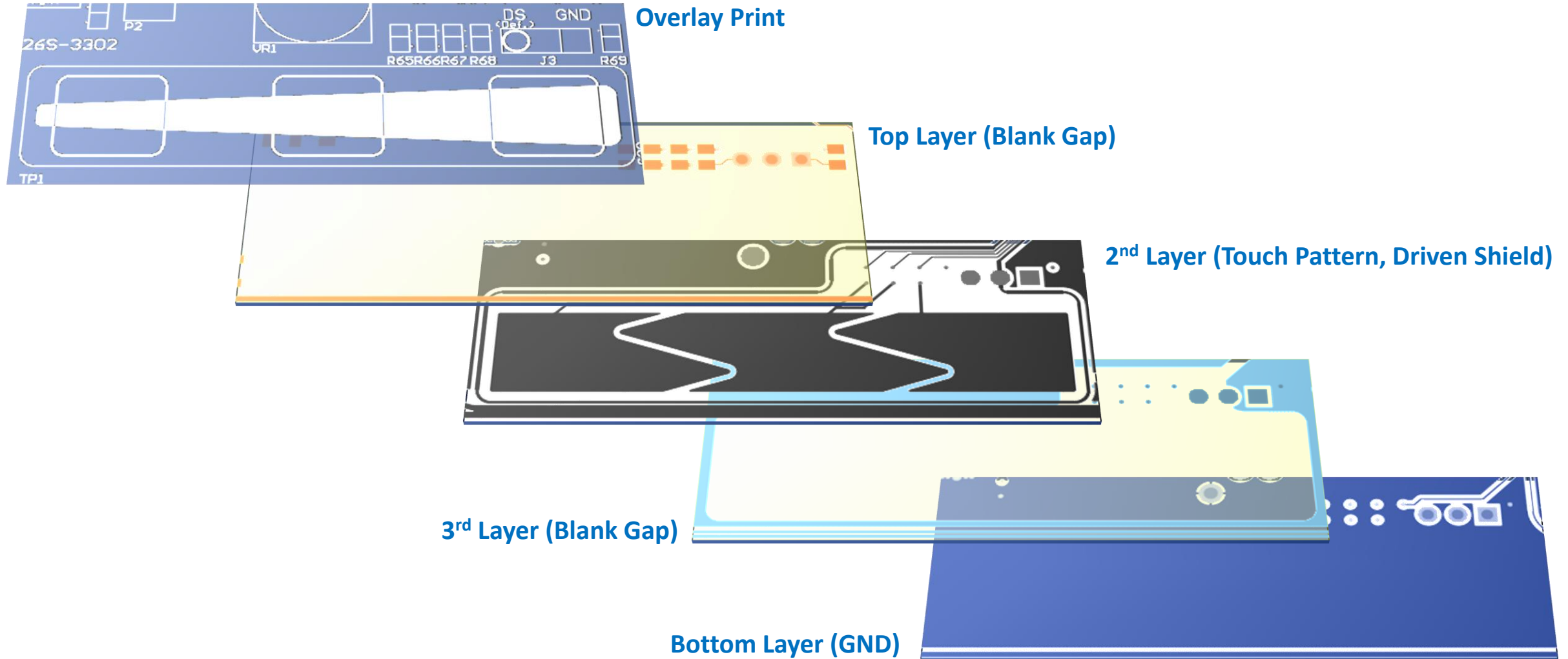
APP045 EVB QTouch® Design and Pattern

Schematic



APP045 EVB QTouch® Design and Pattern

PCB Stack



Harmony Touch Configurator

Harmony Touch Configurator

Add Touch Library in MHC

MPLAB X IDE v6.00 - Lab1_QTouch_Buttons : default

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

default

Projects Files Act... Servi...

Project Graph*

View: Root

Lab1_Q... main() - ... Availa...

Peripherals

- RTC
- SERCOM
- TC
- TCC
- System Services
- Third Party Libraries
- Tools
- Touch
 - PTC
 - Touch Library

Double click

Project Graph*

View: Root

NVMCTRL Peripheral Library MEMORY

Device Family Pack (DFP) System CMSIS Pack

EVSYS Peripheral Library

Core Harmony Core Service RTOS Core Service

TIME System Service Core Service SYS_TIME TMR

TC4 Peripheral Library TMR

SERCOM5 Peripheral Library UART

RTC Peripheral Library TMR

PTC Acq_Engine

Touch Library Peripheral Touch Controller(PTC) TMR Acq_Engine TouchData

CONSOLE System Service Core Service

Instances Instance 0 UART USB_DEVICE_CDC SYS_CONSOLE

Harmony Touch Configurator

Select MHC\Tools\Touch Configuration

The screenshot displays the MPLAB X IDE v6.00 interface. The 'MHC' menu is open, and the 'Tools' option is selected, which has opened a sub-menu. In this sub-menu, 'Touch Configuration' is highlighted. On the left side of the IDE, the 'Peripherals' tree is visible, with 'RTC', 'PTC', and 'Touch Library' highlighted. The 'Project Graph' window on the right shows a block diagram of the system components, including 'NVMCTRL', 'Device Family Pack (DFP)', 'System', 'CMSIS Pack', 'Core Service', 'TIME', 'SYS_TIME', 'TMR', 'CONSOLE', 'SYS_CONSOLE', 'UART', 'PTC', 'Acq_Engine', 'Touch Library', 'Peripheral Touch Controller(PTC)', 'TMR', 'Acq_Engine', and 'TouchData'. A red circle with a white 'X' and the word 'Click' is placed over the 'Touch Configuration' option in the sub-menu.

Harmony Touch Configurator

Sensors : Select Sensing Type and add Touch Sensors

Project Graph* Touch Configurator x

Sensors

Combine Sensors/
Create Lump Sensors

Delete All Sensors

Enable Low Power

Delete Sensor

Self Capacitance Sensing v

Self Capacitance Sensing

Mutual Capacitance Sensing

Technology: ? Self Capacitance Sensing

Number of Buttons: ? 1

Add Cancel

Technology: ? Self Capacitance Sensing

Number of Sliders: ? 1

Number of Channels: ? 3

Add Cancel

Technology: ? Self Capacitance Sensing

Number of Wheels: ? 1

Number of Channels: ? 3

Add Cancel

Technology: ? Self Capacitance Sensing

Number of Horizontal Channels: ? 3

Number of Vertical Channels: ? 3

Gesture Type: ? No Gestures v

Add Cancel

Technology: ? Mutual Capacitance Sensing

Number of Buttons: ? 1

Add Cancel

Technology: ? Mutual Capacitance Sensing

Number of Sliders: ? 1

Number of Channels: ? 3

Add Cancel

Technology: ? Mutual Capacitance Sensing

Number of Wheels: ? 1

Number of Channels: ? 3

Add Cancel

Technology: ? Mutual Capacitance Sensing

Number of Horizontal Channels: ? 3

Number of Vertical Channels: ? 3

Gesture Type: ? No Gestures v

Add Cancel

Harmony Touch Configurator

Sensors : Auto configure the clock for CPU and peripherals

The screenshot shows the Harmony Touch Configurator interface. A 'WARNING' dialog box is open, displaying the following text:

The clock for CPU and peripherals will be set as follows if **YES** option is selected.

If **NO** is selected, then user needs to configure the clock for PTC and RTC peripherals.

Refer to notification tab for details.

Oscillator	Prescaler Divisor	Frequency
OSC8M	2	4MHz
DFLL48M	1	48MHz(GCLK3, open loop, multi factor 1500)
Generic clock(Peripherals)	Source	Frequency
GCLK0(CPU)	DFLL48M	48MHz
GCLK1(Timer)	OSCULP32K	2.048KHz
GCLK2(PTC,UART)	OSC8M	4MHz
Wait states		
NVM states	2	

At the bottom of the dialog, there are 'Yes' and 'No' buttons. A red arrow points to the 'Yes' button, which is highlighted with a yellow box and labeled 'Click Yes'.

The background shows the 'Sensors' tab selected in the 'Touch Configurator' window. The 'Technology' dropdown is set to 'Self Capacitance Sensing'. The 'Number of Buttons' is set to 3. The 'Number of Channels' is set to 3. The 'Gesture Type' is set to 'No Gestures'.

Harmony Touch Configurator

Sensors : Type of Self or Mutual Capacitance Sensing and Add Sensors

Project Graph* Touch Configurator ×

Sensors

Pins

Parameters

Notifications

Summary

Combine Sensors/
Create Lump Sensors

Delete All Sensors

Enable Low Power

Delete Sensor

Self Capacitance Sensing ▼

0 0(3) 0(3)

Project Graph* Touch Configurator ×

Sensors

Pins

Parameters

Notifications

Summary

Combine Sensors/
Create Lump Sensors

Delete All Sensors

Disable Low Power

Delete Sensor

Mutual Capacitance Sensing ▼

0 0(3) 0(3) 0(6)

Harmony Touch Configurator

Pins : Self Capacitance Sensing Pin Selection and Driven Shield configuration

Project Graph* Touch Configurator x

Sensors

Pins

Parameters

Notifications

Summary

Auto Assign Reset

Pin Selection Driven Shield

☒ Table View

Sensor Type	Channel ID	Y-Line:	Pin Selection
button 0	0		Y0 (PA02) ▾
slider 0	1		Y1 (PA03) ▾
	2		Y2 (PA04) ▾
	3		Y3 (PA05) ▾
wheel 0	4		Y4 (PA06) ▾
	5		Y5 (PA07) ▾
	6		Y8 (PB02) ▾

Configurator x

Auto Assign Reset

Pin Selection **Driven Shield**

Driven Shield

To improve noise performance and water tolerance, driven shield can be enabled for shield sensor. More information can be found Microchip Developer Help page.

☒ Enable Dedicated Driven Shield

Any Timer WaveOutput (WO) pin can be used as Driven shield pin. Select the

Timer instance TC3 ▾

Shield pin PA18E_TC3_W... ▾

“Driven shield Plus” feature configures the PTC pins which are currently not supported in this application. Selecting the timer will add the timer to the project and will be

☐ Enable Driven Shield Plus

Harmony Touch Configurator

Pins : Mutual Capacitance Sensing kind of Pin Selection GUIs

Project Graph*

Touch Configurator ×

Sensors

Pins

Parameters

Notifications

Summary

Auto Assign

Pin Selection

Driven Shield

☐ Table View

☒ **Matrix View**

☐ Surface Matrix View (2D Surface Pin Configuration)

(Note: In matrix view, "b : x" means "button x" & "c : x" means "channel x". Same will applicab

	X0 (PA08)	X1 (PA09)	X2 (PA10)	X3 (PA11)	X4 (PA16)	X5 (PA17)	(P
Y0 (PA02)	b : 0 c : 0	s : 0 c : 1	s : 0 c : 2	s : 0 c : 3	w : 0 c : 4	w : 0 c : 5	w c
Y1 (PA03)	NA	NA	NA				
Y2 (PA04)	NA	NA	NA				
Y3 (PA05)	NA	NA	NA				
Y4 (PA06)							

☐ Table View

☐ Matrix View

☒ **Surface Matrix View (2D Surface Pin Configuration)**

(Note: In matrix view, "b : x" means "button x" & "c : x" means "channel x". Same will applicab

Ch10

Ch11

Ch12

Ch9

Ch8

Ch7

Y3
(PA05)

Y2
(PA04)

Y1
(PA03)

X0
(PA08)

X1
(PA09)

X2
(PA10)

Harmony Touch Configurator

Parameters (Channel): Channel Samples, Gain, Clock, Threshold and AKS configuration

The screenshot displays the Harmony Touch Configurator software interface. The 'Channel' tab is selected and highlighted with a yellow box. In the left sidebar, the 'Parameters' option is highlighted with a red box. The main area shows the 'PTC CLOCK FREQUENCY' section, which includes a diagram illustrating the calculation: GCLK/CLKPER (4.0000 MHz) divided by a Prescaler (4, 8, 16, or 32) results in a PTC Clock (1.0000 MHz, 0.5000 MHz, 0.2500 MHz, or 0.1250 MHz). Below this, the 'CHANNEL PARAMETERS' section features a 'Change all' checkbox and a table for configuring individual channels.

PTC CLOCK FREQUENCY

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

Diagram: GCLK/CLKPER 4.0000 MHz → Prescaler (4, 8, 16, 32) → PTC Clock (1.0000 MHz, 0.5000 MHz, 0.2500 MHz, 0.1250 MHz)

CHANNEL PARAMETERS

☐ Change all (Note: Enabling Change all, and changing a parameter will modify it for all sensors in the design. This does not apply to AKS)

Channel ID	Over Samples	Gain	Analog Gain	Series Resistor	PTC clock	Threshold	Hysterisis	Sensor AKS
Sensor List								
Sensor Id: butto								
0	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	No AKS
Sensor Id: slide								
1	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 1
2	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 1
3	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 1
Sensor Id: wheel								
4	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 2
5	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 2
6	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 2

Harmony Touch Configurator

Parameters (Sensor): Sensor Parameters, Acquisition, Noise Handling

The screenshot displays the Harmony Touch Configurator software interface. The top window title bar shows 'Project Graph*' and 'Touch Configurator *'. Below the title bar, there are four tabs: 'Channel', 'Sensor' (highlighted with a yellow box), 'Sliders and w...', and 'Tune'. On the left side, there is a vertical sidebar with five buttons: 'Sensors', 'Pins', 'Parameters' (highlighted with a red box), 'Notifications', and 'Summary'. The main content area is titled 'SENSOR PARAMETERS' and contains several configuration options, each with a help icon (question mark) and a value field:

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

Below the 'SENSOR PARAMETERS' section is the 'ACQUISITION CONFIGURATION' section, which includes:

- Scan Rate (ms): 20
- PTC Interrupt Priority: 3
- Acquisition Frequency: FREQ_SEL_0

At the bottom of the main content area, there is a section titled 'NOISE HANDLING USING FREQUENCY HOP'.

Harmony Touch Configurator

Parameters (Slider And Wheels): Scroller Resolution, Position Hysteresis and Threshold

The screenshot displays the Harmony Touch Configurator software interface. On the left, a vertical sidebar contains buttons for 'Sensors', 'Pins', 'Parameters' (highlighted with a red and yellow border), 'Notifications', and 'Summary'. The main window has a top bar with tabs: 'Channel', 'Sensor', 'Sliders and w...' (highlighted with a yellow border), and 'Tune'. Below the tabs, a table lists configuration parameters for two sensors: 'Sensor Id: slider 0' (scroller 0) and 'Sensor Id: wheel 0' (scroller 1). The table columns are 'Scroller Id', 'Scroller Resolution', 'Scroller Deadband', 'Scroller Position Hysteresis', and 'Scroller Detect Threshold'. The 'Sliders and w...' tab is active, showing the 'Scroller Resolution' and 'Scroller Deadband' settings for both sensors, both set to '8 Bit' and '10 Percent' respectively. The 'Scroller Position Hysteresis' and 'Scroller Detect Threshold' values are also visible for each sensor.

Scroller Id	? Scroller Resolution	? Scroller Deadband	? Scroller Position Hysteresis	? Scroller Detect Threshold
Sensor List				
Sensor Id: slider 0				
● scroller 0	8 Bit	10 Percent	8	20
Sensor Id: wheel 0				
● scroller 1	8 Bit	10 Percent	8	20

Harmony Touch Configurator

Parameters (Surface/Gesture): QTouch Surface and 2D Gesture configuration

Project Graph* Touch Configurator *

Sensors Pins **Parameters** Notifications Summary

Channel Sensor Sliders and w... **Surface** Gesture Tune

QTOUCH SURFACE CONFIGURATION

SURFACE CONFIGURATION PARAMETERS

Position Resolution	?	RESOL_8_BIT
Deadband Percentage	?	DB_1_PERCENT
Surface Position Hysteresis	?	3
Surface Detect threshold	?	60

Project Graph* Touch Configurator *

Sensors Pins **Parameters** Notifications Summary

Channel Sensor Sliders and w... Surface **Gesture** Tune

☒ **ENABLE GESTURE**

Enable single finger and dual finger gestures below.

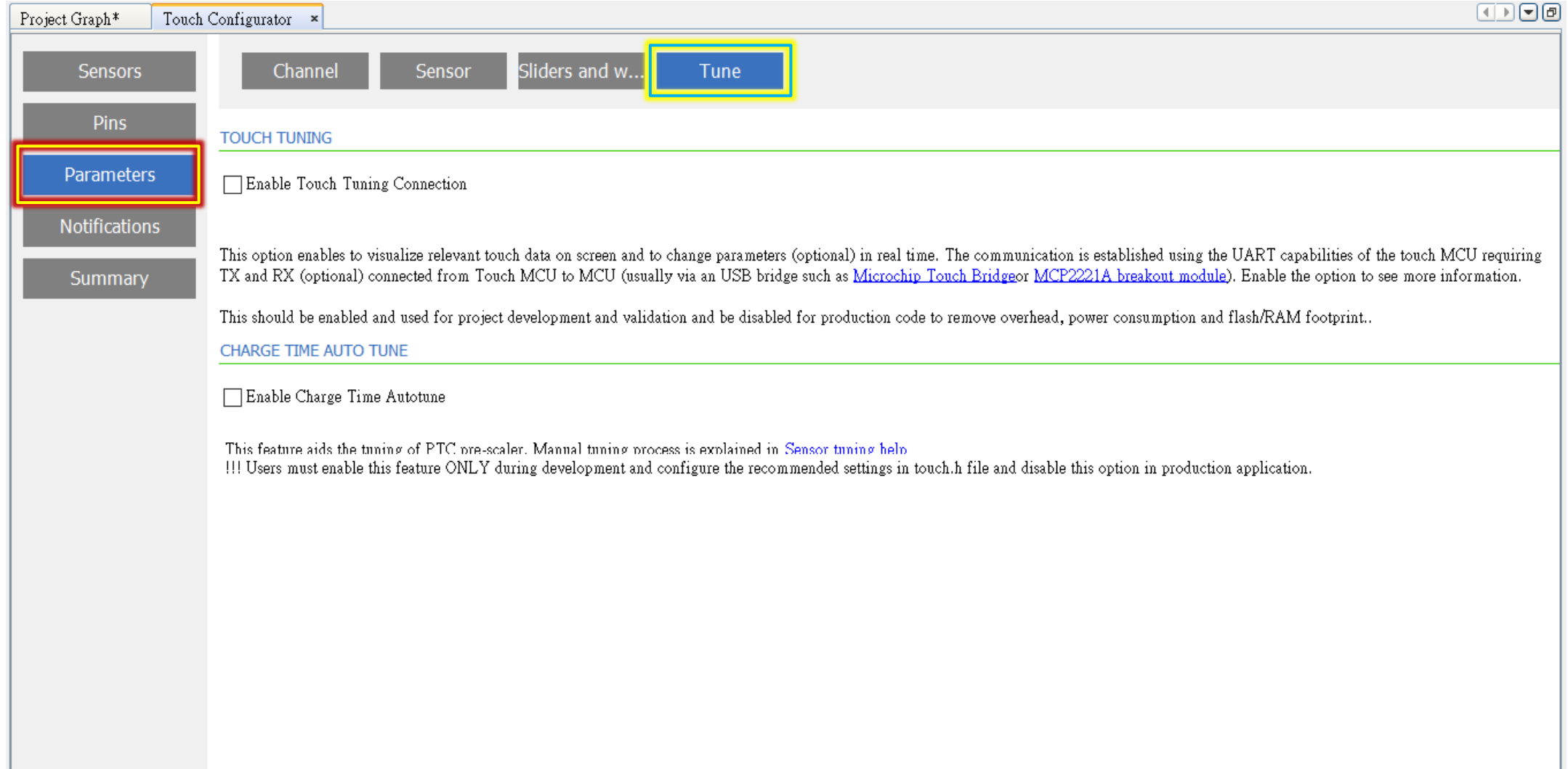
Note: Enabling Gesture here will allow only 1 Finger Gestures support. If 1&2 Finger Gestures support is needed, delete the existing Surface sensor and Surface sensor with Gesture Type option as 1&2 Finger Gestures.

GESTURES - TAP

Tap Release timeout	?	20
Tap Hold timeout	?	100
Tap Area	?	20

Harmony Touch Configurator

Parameters (Tune): Enable touch debug data to output in visualize for touch tuning



The screenshot displays the Harmony Touch Configurator software interface. On the left, a vertical sidebar contains five buttons: 'Sensors', 'Pins', 'Parameters', 'Notifications', and 'Summary'. The 'Parameters' button is highlighted with a red and yellow border. The main window has a title bar with 'Project Graph*' and 'Touch Configurator *'. Below the title bar, there is a horizontal bar with four buttons: 'Channel', 'Sensor', 'Sliders and w...', and 'Tune'. The 'Tune' button is highlighted with a yellow border. The main content area is titled 'TOUCH TUNING' and contains two sections. The first section, 'TOUCH TUNING', has a checkbox labeled 'Enable Touch Tuning Connection'. Below this checkbox, there is a paragraph explaining that this option enables visualizing touch data and changing parameters in real time, requiring TX and RX (optional) connected from Touch MCU to MCU via a USB bridge such as [Microchip Touch Bridgeor MCP2221A breakout module](#). The second section, 'CHARGE TIME AUTO TUNE', has a checkbox labeled 'Enable Charge Time Autotune'. Below this checkbox, there is a paragraph explaining that this feature aids the tuning of PTC pre-scaler, with a link to [Sensor tuning help](#). A warning message states: '!!! Users must enable this feature ONLY during development and configure the recommended settings in touch.h file and disable this option in production application.'

Project Graph* Touch Configurator *

Sensors Channel Sensor Sliders and w... Tune

Pins

Parameters

Notifications

Summary

TOUCH TUNING

☐ Enable Touch Tuning Connection

This option enables to visualize relevant touch data on screen and to change parameters (optional) in real time. The communication is established using the UART capabilities of the touch MCU requiring TX and RX (optional) connected from Touch MCU to MCU (usually via an USB bridge such as [Microchip Touch Bridgeor MCP2221A breakout module](#)). Enable the option to see more information.

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..

CHARGE TIME AUTO TUNE

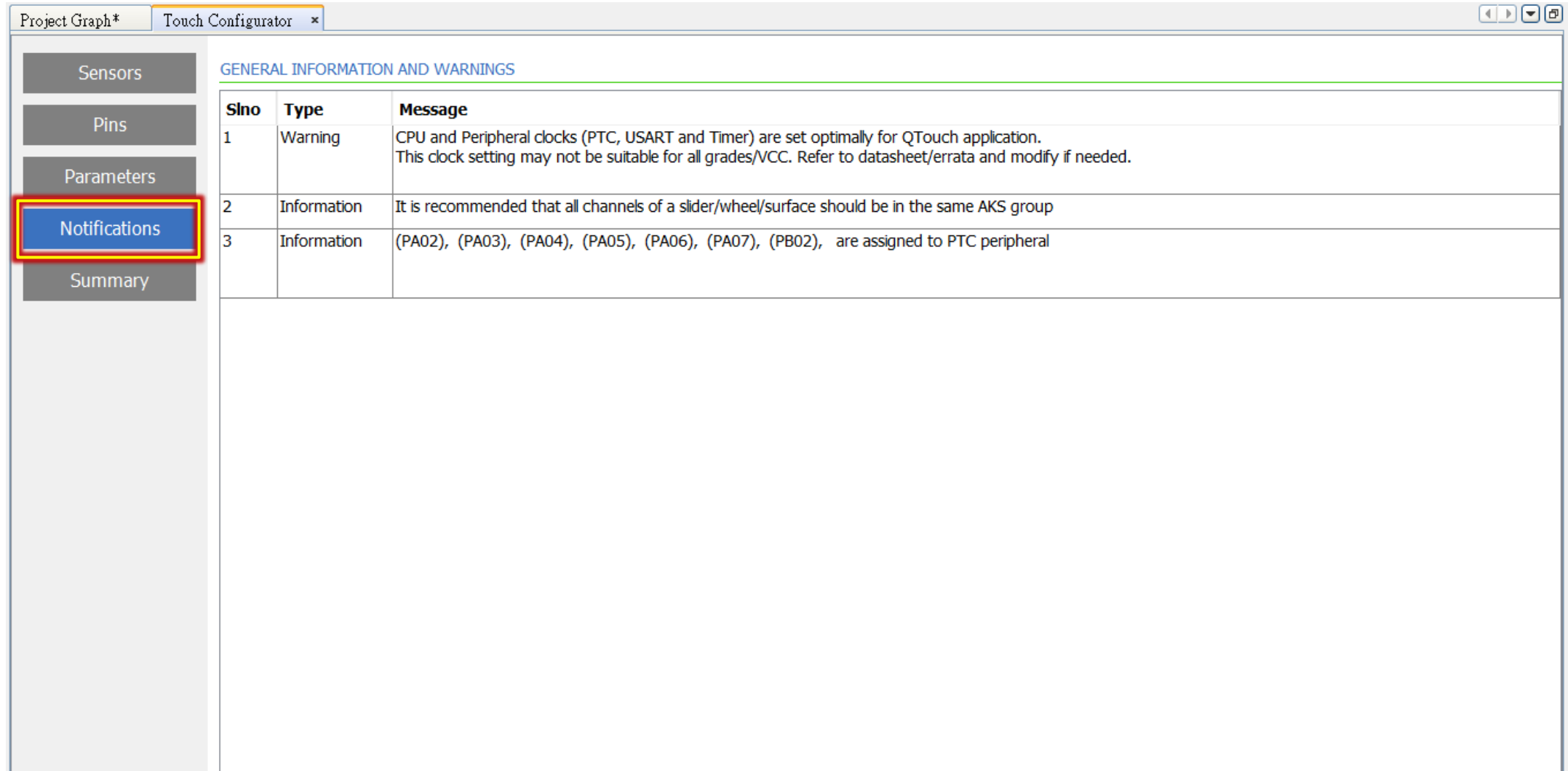
☐ Enable Charge Time Autotune

This feature aids the tuning of PTC pre-scaler. Mannal tuning process is explained in [Sensor tuning help](#)

!!! Users must enable this feature ONLY during development and configure the recommended settings in touch.h file and disable this option in production application.

Harmony Touch Configurator

Notifications: Check if there are any warnings or guidelines



The screenshot shows the Harmony Touch Configurator application window. The title bar includes tabs for 'Project Graph*' and 'Touch Configurator *'. On the left, a vertical sidebar contains buttons for 'Sensors', 'Pins', 'Parameters', 'Notifications' (highlighted with a red and yellow border), and 'Summary'. The main area is titled 'GENERAL INFORMATION AND WARNINGS' and contains a table with three rows of notifications.

Sno	Type	Message
1	Warning	CPU and Peripheral clocks (PTC, USART and Timer) are set optimally for QTouch application. This clock setting may not be suitable for all grades/VCC. Refer to datasheet/errata and modify if needed.
2	Information	It is recommended that all channels of a slider/wheel/surface should be in the same AKS group
3	Information	(PA02), (PA03), (PA04), (PA05), (PA06), (PA07), (PB02), are assigned to PTC peripheral

Harmony Touch Configurator

Summary: Review the project configuration and find versions of library modules

The screenshot shows the Harmony Touch Configurator interface. The left sidebar contains five buttons: 'Sensors', 'Pins', 'Parameters', 'Notifications', and 'Summary'. The 'Summary' button is highlighted with a red and yellow border. The main content area is titled 'SUMMARY' and contains the following sections:

- DEVICE INFORMATION**
 - Device Name: ATSAM21G18A
 - Technology: SelfCap
 - Total Sensor Count: 3
- SENSOR INFORMATION**
 - Number of Buttons: 1
 - Number of Sliders: 1
 - Number of Wheels: 1
 - Number of Surface: 0
 - Number of Lumps: 0
- CHANNEL INFORMATION**
 - Total Channels Consumed: 7
- OTHER INFORMATION**
 - Acquisition: Enabled - Manual Tuning Configuration
 - Frequency Hop: Enabled - Manual Tuning Configuration
 - Data Streamer: Disabled
 - Gesture: Disabled
 - Low Power: Disabled
- MODULES & VERSIONS**

Lab Preparation

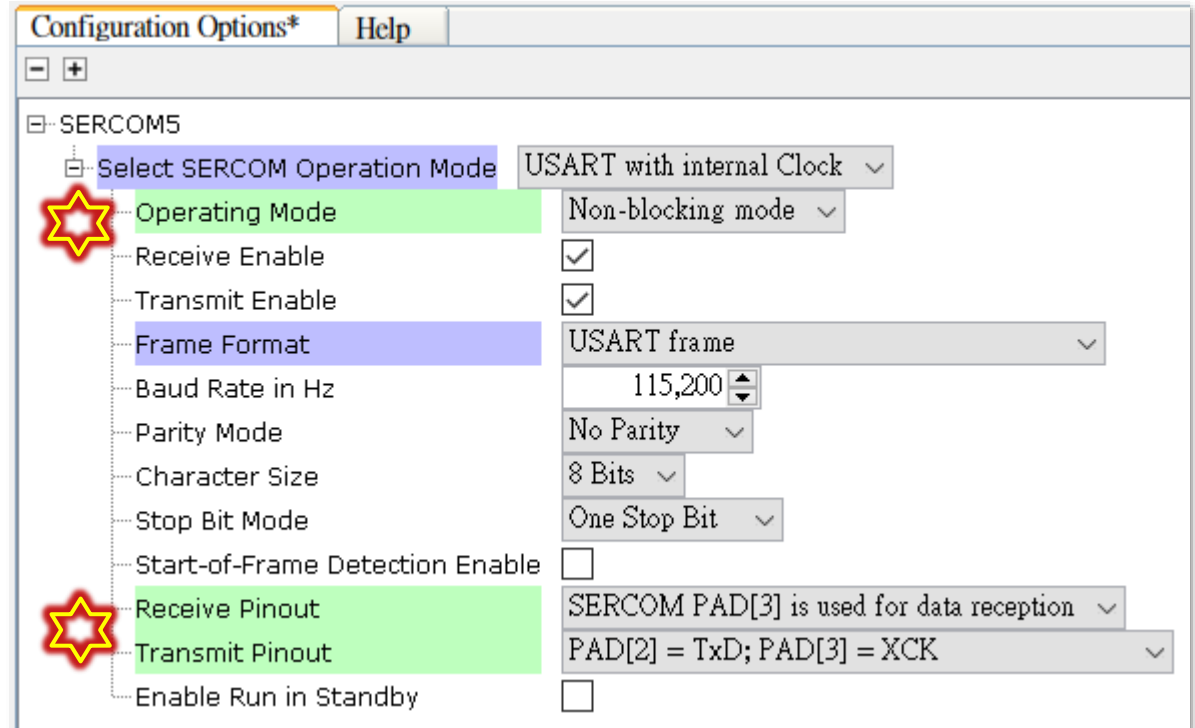
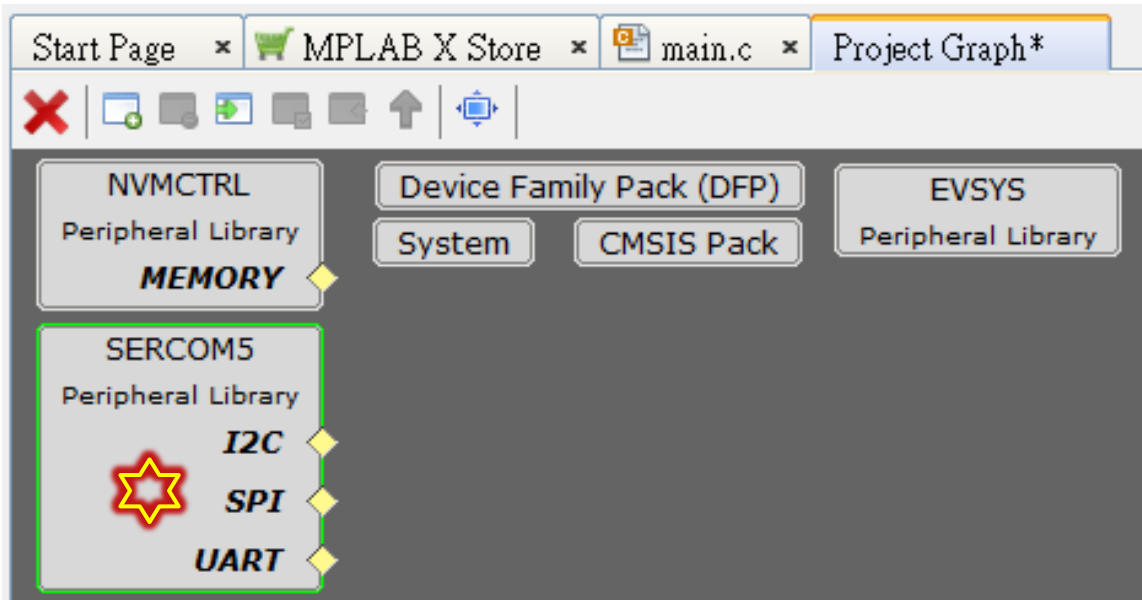
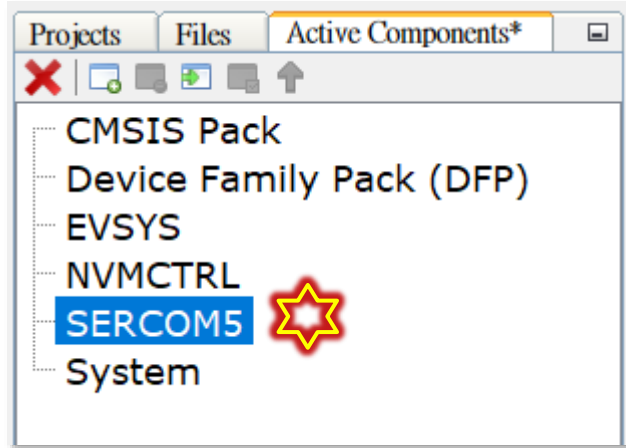
Lab Preparation

- **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 6.00 <https://www.microchip.com/mplab/mplab-x-ide>
 - MPLAB® XC32 v4.10 <https://www.microchip.com/mplab/compilers>
 - MPLAB® Harmony 3 Launcher v3.6.4 (From X IDE plugin)
 - MPLAB® Harmony Framework (From X IDE Harmony 3 Content Manager)
 - core v3.11.1
 - csp v3.14.0
 - mhc v3.8.5
 - dev_packs v3.14.0
 - touch v3.13.0
 - usb v3.10.0
 - SAMD21 DFP v3.6.144 (Device Firmware Pack) (From X IDE \Tools\Packs)
 - CMSIS v5.8.0 (Common Microcontroller Software Interface Standard)
- **Let's go !**

Lab1-1 Qtouch Buttons

⚙️ Lab1-1 Qtouch Buttons

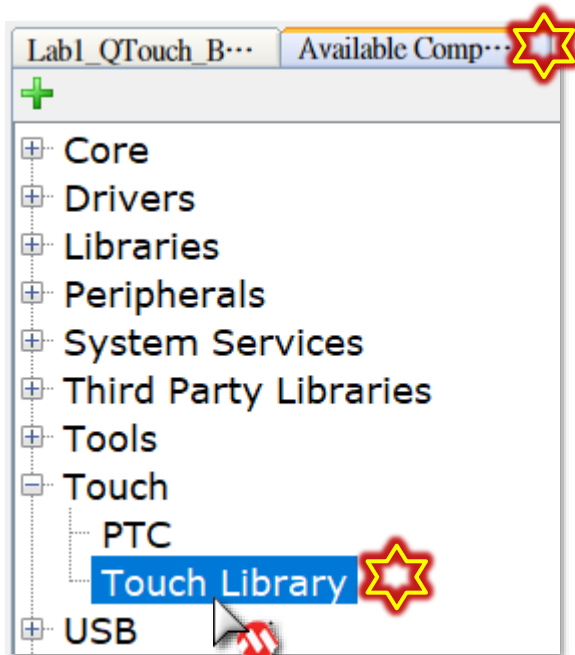
- Create a new project and add SERCOM5 in MHC for UART debug message.



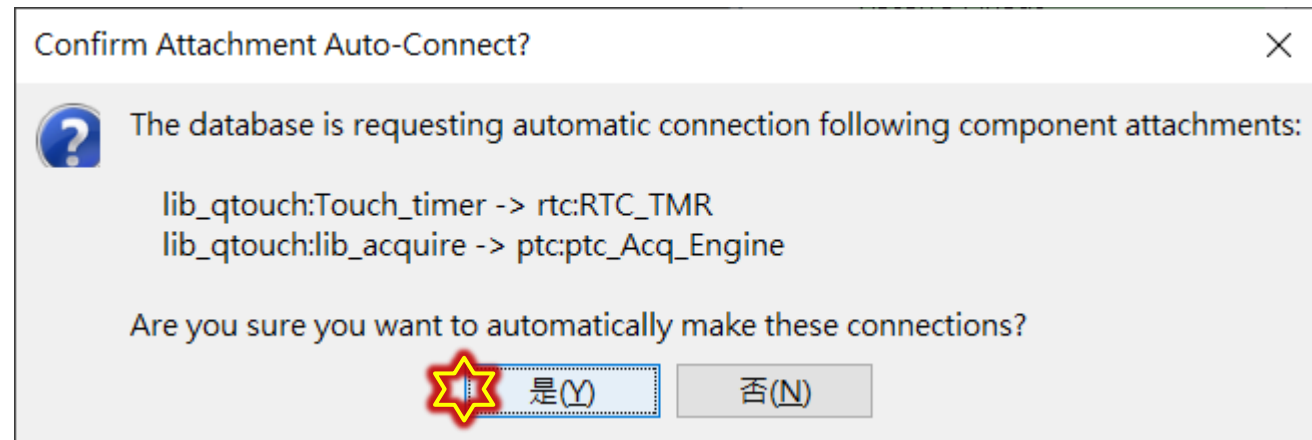
Pin Number	Pin ID	Custom Name	Function
37	PB22		SERCOM5_PAD2
38	PB23		SERCOM5_PAD3

✂ Lab1-1 Qtouch Buttons

- Find **Touch Library** under **Touch** of **Available Component** and add it to your project.
- Please wait a while after double click to add the **Touch Library**.
- **RTC** and **PTC** will request to **Auto-Activation** together and **Auto-Connection**, please simple click “Yes” and “Yes”.

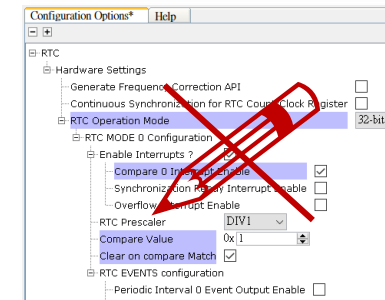
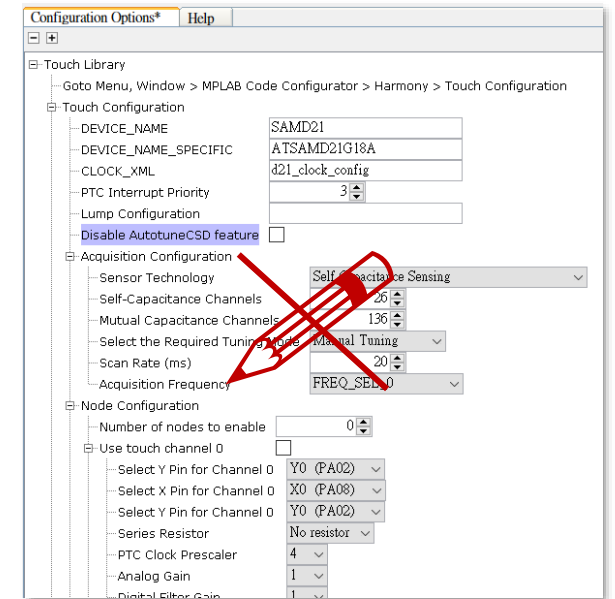
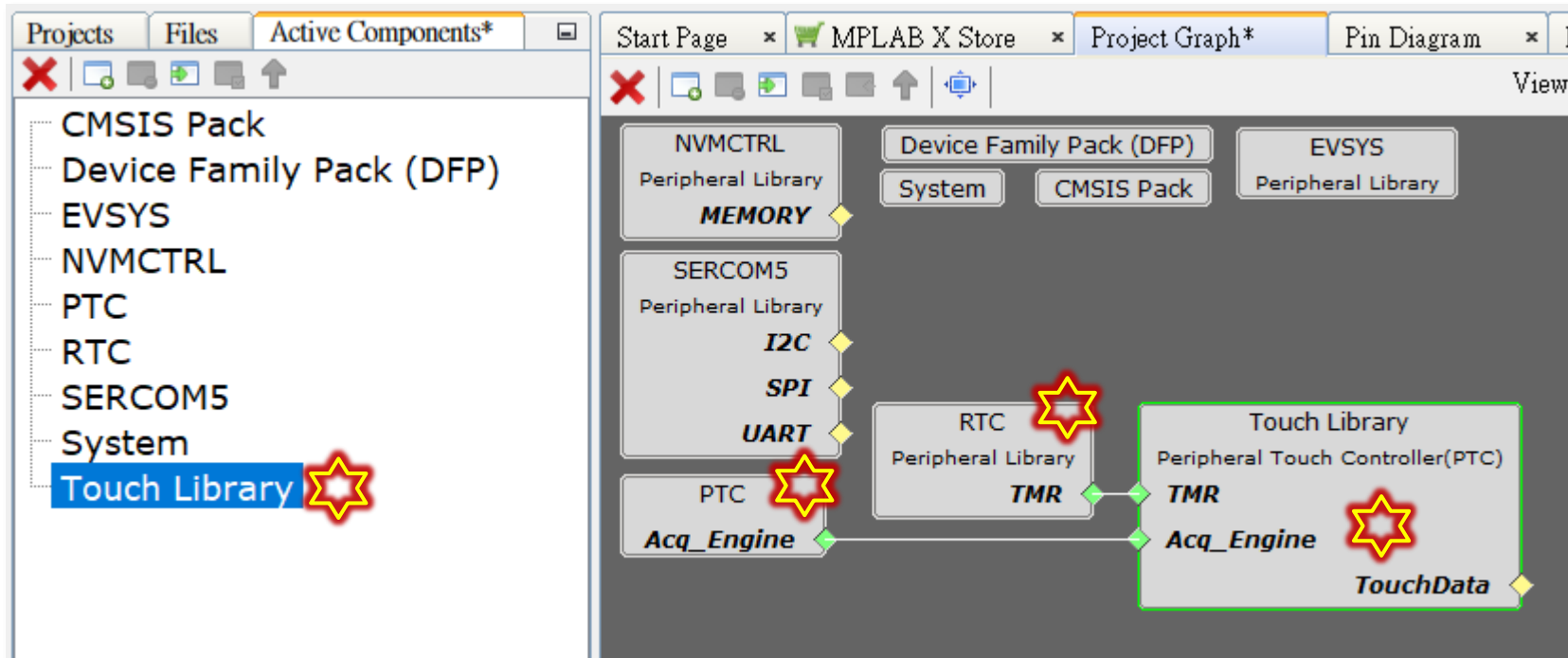


Double Click to add this



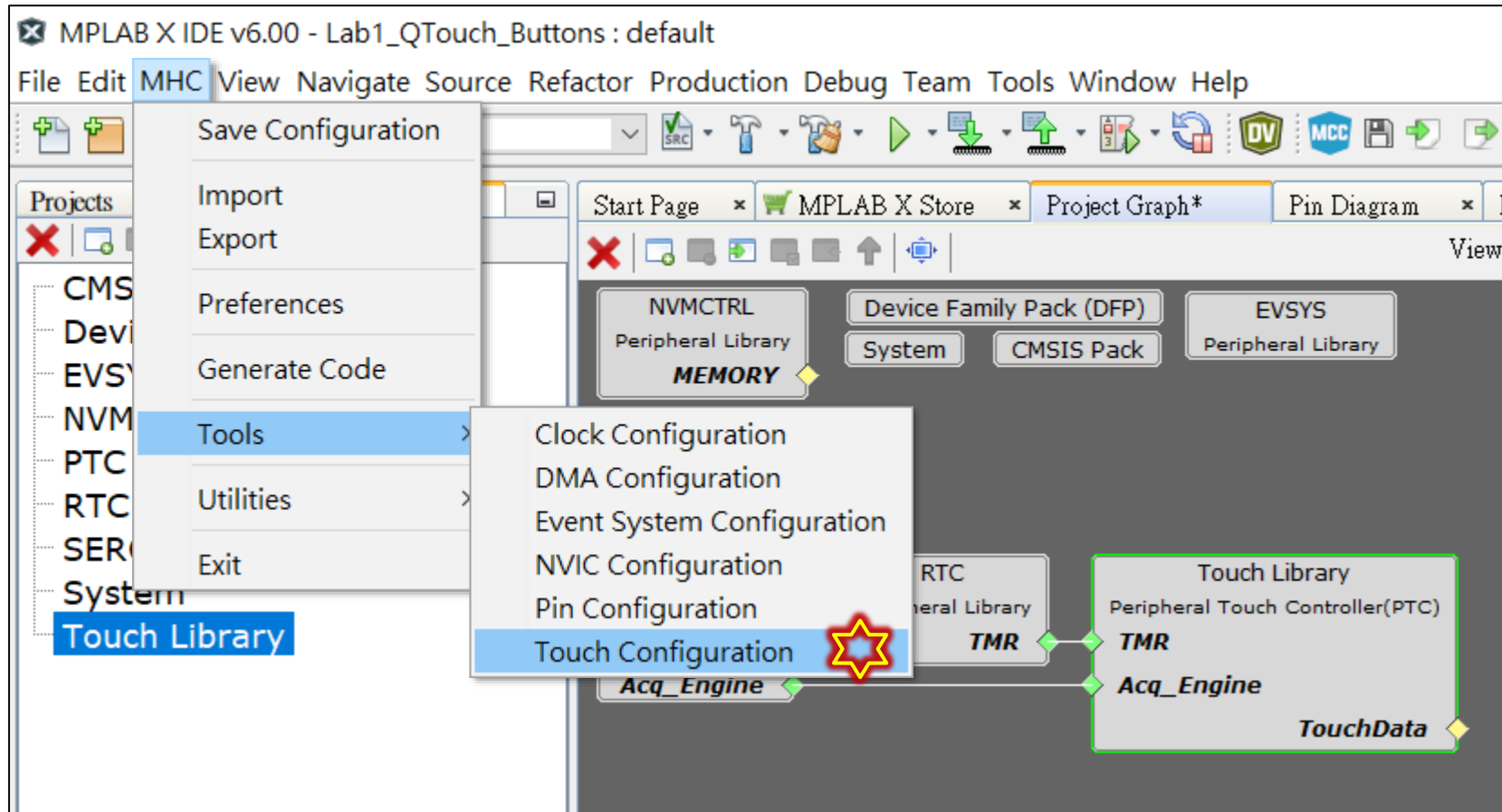
✂ Lab1-1 Qtouch Buttons

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



⚙️ Lab1-1 Qtouch Buttons

- Open Touch Configuration



⚙️ Lab1-1 Qtouch Buttons

- In **Sensor** Tab, click on Button Icon and input **Number of Buttons** to **3**.
- Click Add then a Warning dialog will show up to notice the clock of CPU and peripheral will be auto set as default Touch Function request if select **“Yes”**

The image shows the Touch Configurator software interface. In the top-left window, the 'Sensors' tab is active. A yellow star icon is placed over the 'Click' button. A dialog box is open for configuring a new sensor. It shows 'Technology: Self Capacitance Sensing' and 'Number of Buttons: 3'. A yellow star icon is placed over the 'Add' button. A blue arrow points from this dialog to the right, where a 'WARNING' dialog box is shown. The warning dialog contains the following text: 'The clock for CPU and peripherals will be set as follows if YES option is selected. If NO is selected, then user needs to configure the clock for PTC and RTC peripherals. Refer to notification tab for details.' Below the text is a table with clock configurations. At the bottom of the warning dialog, there are 'Yes' and 'No' buttons, with a yellow star icon placed over the 'Yes' button. A blue arrow points from the 'Yes' button to the bottom window. The bottom window shows the 'Touch Configurator' with the 'Sensors' tab selected. The 'Number of Buttons' is set to 3. A blue arrow points from the 'Add' button to the bottom window, where the 'Number of Buttons' is set to 3. The bottom window also shows the 'Pins', 'Parameters', 'Notifications', and 'Summary' tabs. A blue arrow points from the 'Add' button to the bottom window, where the 'Number of Buttons' is set to 3.

Touch Configurator * Project Graph*

Sensors

Click

Technology: ? Self Capacitance Sensing

Number of Buttons: ? 3 3

Add Cancel

WARNING

The clock for CPU and peripherals will be set as follows if YES option is selected.

If NO is selected, then user needs to configure the clock for PTC and RTC peripherals.

Refer to notification tab for details.

Oscillator	Prescaler Divisor	Frequency
OSC8M	2	4MHz
DFLL48M	1	48MHz(GCLK3, open loop, multi factor 1500)
Generic clock(Peripherals)	Source	Frequency
GCLK0(CPU)	DFLL48M	48MHz
GCLK1(Timer)	OSCULP32K	2.048KHz
GCLK2(PTC,UART)	OSC8M	4MHz
Wait states		
NVM states	2	

Yes No

Yes

Touch Configurator * Project Graph*

Sensors

Pins

Parameters

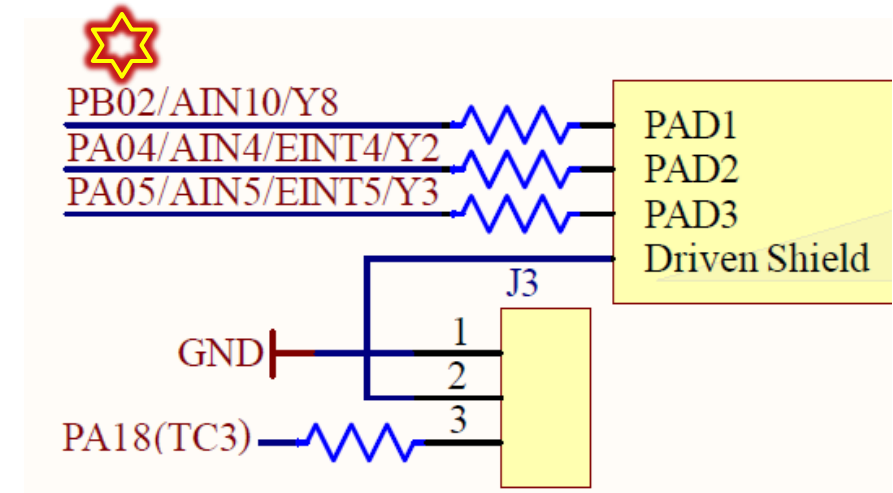
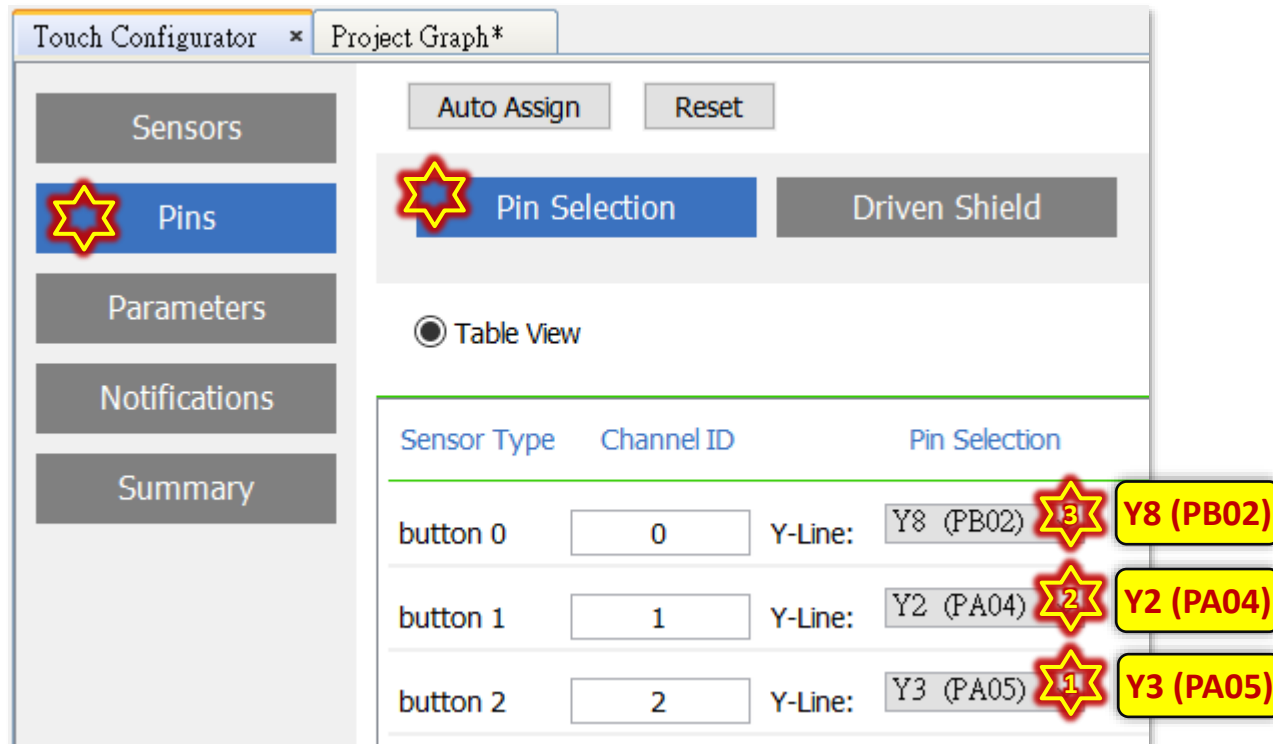
Notifications

Summary

0 1 2

🔧 Lab1-1 Qtouch Buttons

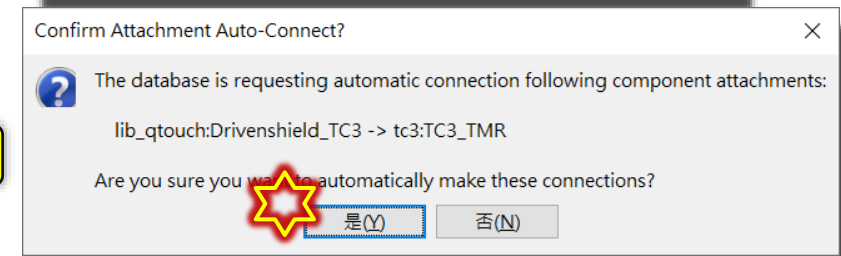
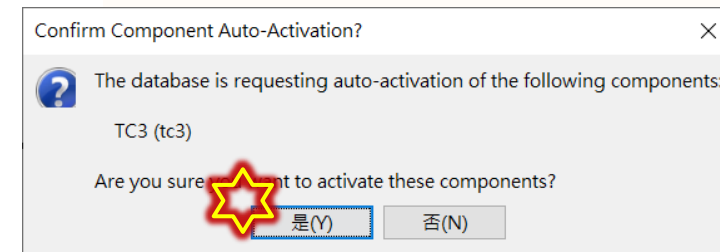
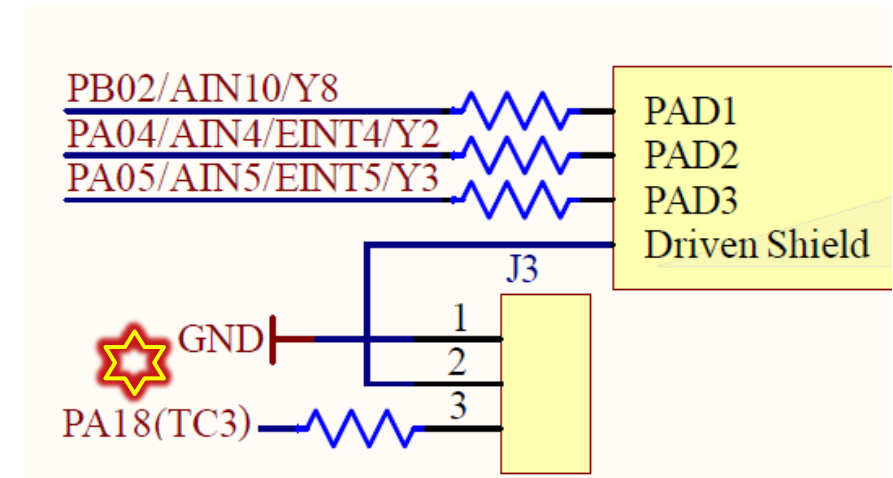
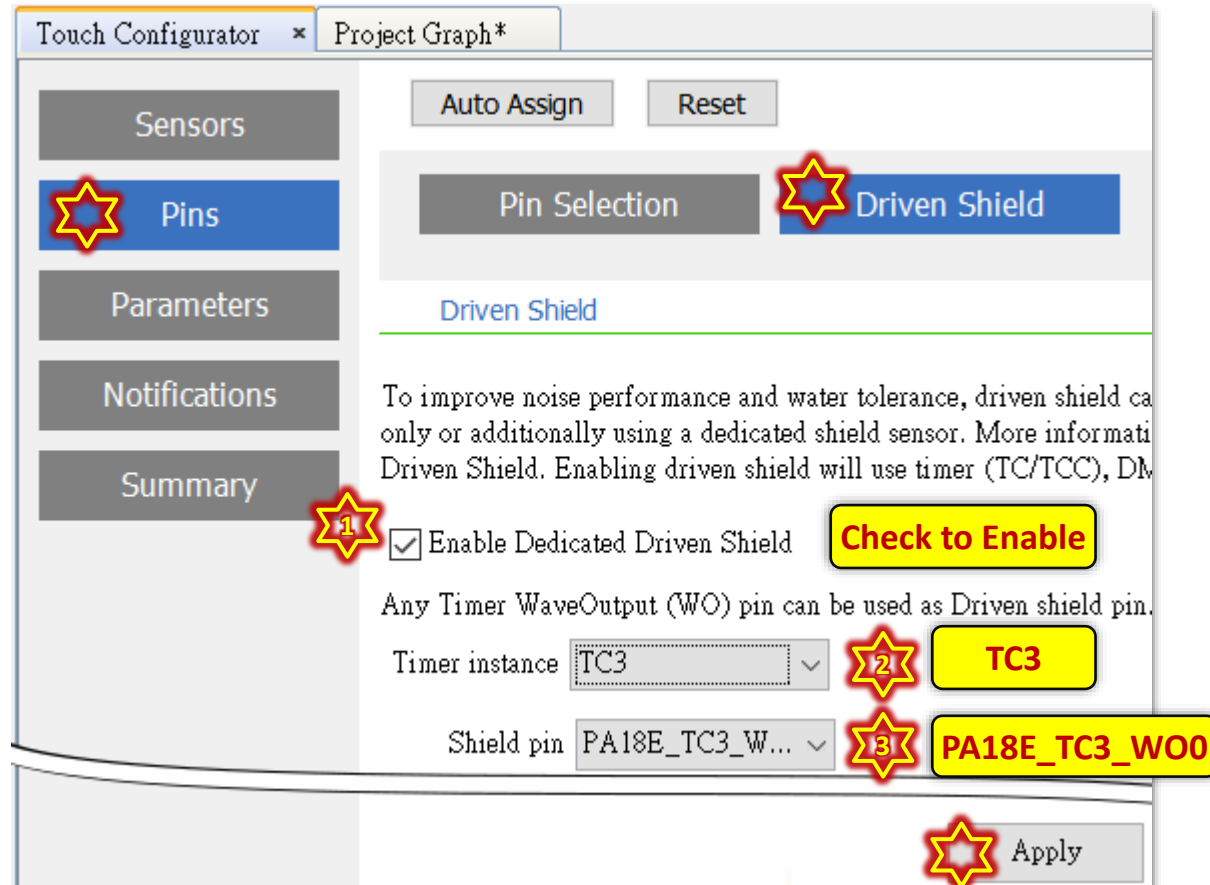
- In **Pins** Tab, Click on **Pin Selection** and configure button pins in below sequence.
 - Button2 (Channel ID 2) : Y3 (PA05)
 - Button1 (Channel ID 1) : Y2 (PA04)
 - Button0 (Channel ID 0) : Y8 (PB02)



✖ Lab1-1 Qtouch Buttons

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

- Enable Dedicated Driven Shield : ☒
- Timer instance : TC3
- Shield pin : PA18E_TC3_WO0



✖ Lab1-1 Qtouch Buttons

- In **Parameters** Tab, configure **Sensor AKS**(Adjacent **K**ey **S**uppression) group as below.
 - Button0 and Button1 : AKS Group 1
 - Button2 : AKS Group 2

The screenshot shows the Touch Configurator software interface. The left sidebar has tabs for Sensors, Pins, Parameters (selected), Notifications, and Summary. The main area has tabs for Channel, Sensor, and Tune. The Channel tab is active, showing PTC CLOCK FREQUENCY settings (GCLK/CLKPER: 4.0000 MHz, Prescaler: 4, PTC Clock: 1.0000 MHz) and CHANNEL PARAMETERS. A table lists three sensors: button 0, button 1, and button 2. Each sensor row has columns for Channel ID, Over Samples, Gain, Analog Gain, Series Resistor, PTC clock, Threshold, Hysterisis, and Sensor AKS. Buttons 0 and 1 are configured for AKS Group 1, and Button 2 is configured for AKS Group 2. Red stars highlight the AKS Group 1 and 2 settings in the table. Yellow callout boxes on the right label these as 'AKS Group 1' and 'AKS Group 2'.

PTC CLOCK FREQUENCY

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

GCLK/CLKPER: 4.0000 MHz → Prescaler: 4 → PTC Clock: 1.0000 MHz

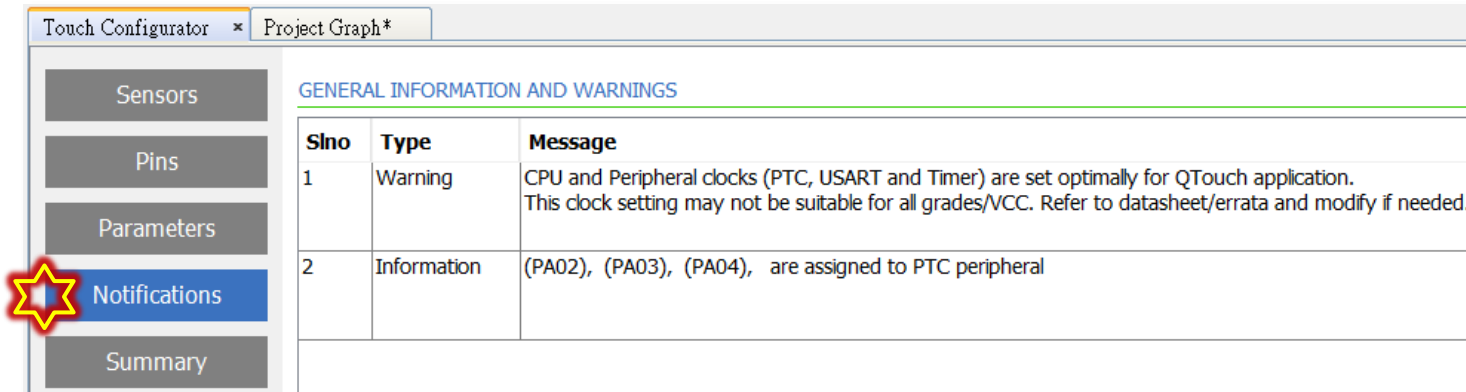
CHANNEL PARAMETERS

☐ Change all (Note: Enabling Change all, and changing a parameter will modify it for all sensors in the design. This does not apply to AKS)

Channel ID	Over Samples	Gain	Analog Gain	Series Resistor	PTC clock	Threshold	Hysterisis	Sensor AKS
Sensor List								
Sensor Id: button 0								
0	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 1
Sensor Id: button 1								
1	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 1
Sensor Id: button 2								
2	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	AKS Group 2

⚙️ Lab1-1 Qtouch Buttons

- Check **Notifications** and **Summary** Tabs to figure out current configuration of Touch.

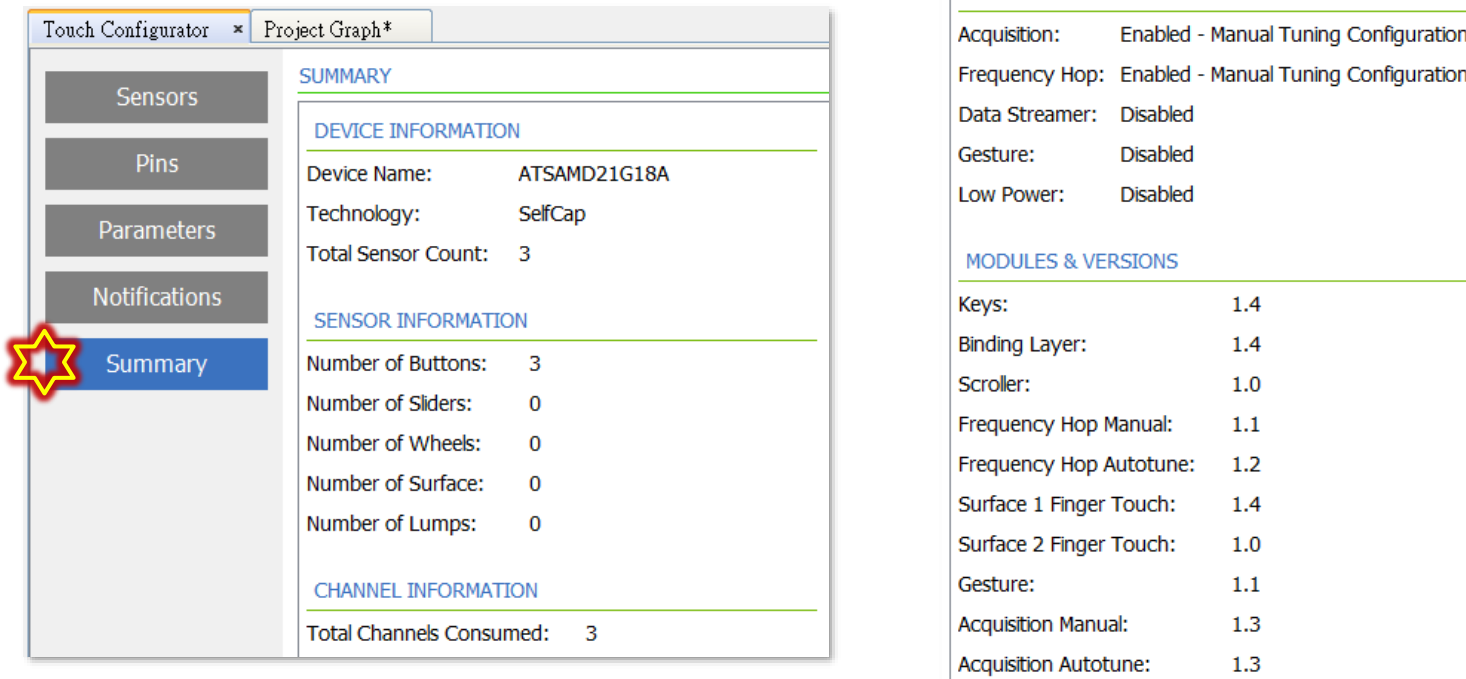


Touch Configurator * Project Graph*

Sensors
Pins
Parameters
Notifications
Summary

GENERAL INFORMATION AND WARNINGS

Sno	Type	Message
1	Warning	CPU and Peripheral clocks (PTC, USART and Timer) are set optimally for QTouch application. This clock setting may not be suitable for all grades/VCC. Refer to datasheet/errata and modify if needed.
2	Information	(PA02), (PA03), (PA04), are assigned to PTC peripheral



Touch Configurator * Project Graph*

Sensors
Pins
Parameters
Notifications
Summary

SUMMARY

DEVICE INFORMATION

Device Name: ATSAM21G18A
Technology: SelfCap
Total Sensor Count: 3

SENSOR INFORMATION

Number of Buttons: 3
Number of Sliders: 0
Number of Wheels: 0
Number of Surface: 0
Number of Lumps: 0

CHANNEL INFORMATION

Total Channels Consumed: 3

OTHER INFORMATION

Acquisition: Enabled - Manual Tuning Configuration
Frequency Hop: Enabled - Manual Tuning Configuration
Data Streamer: Disabled
Gesture: Disabled
Low Power: Disabled

MODULES & 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

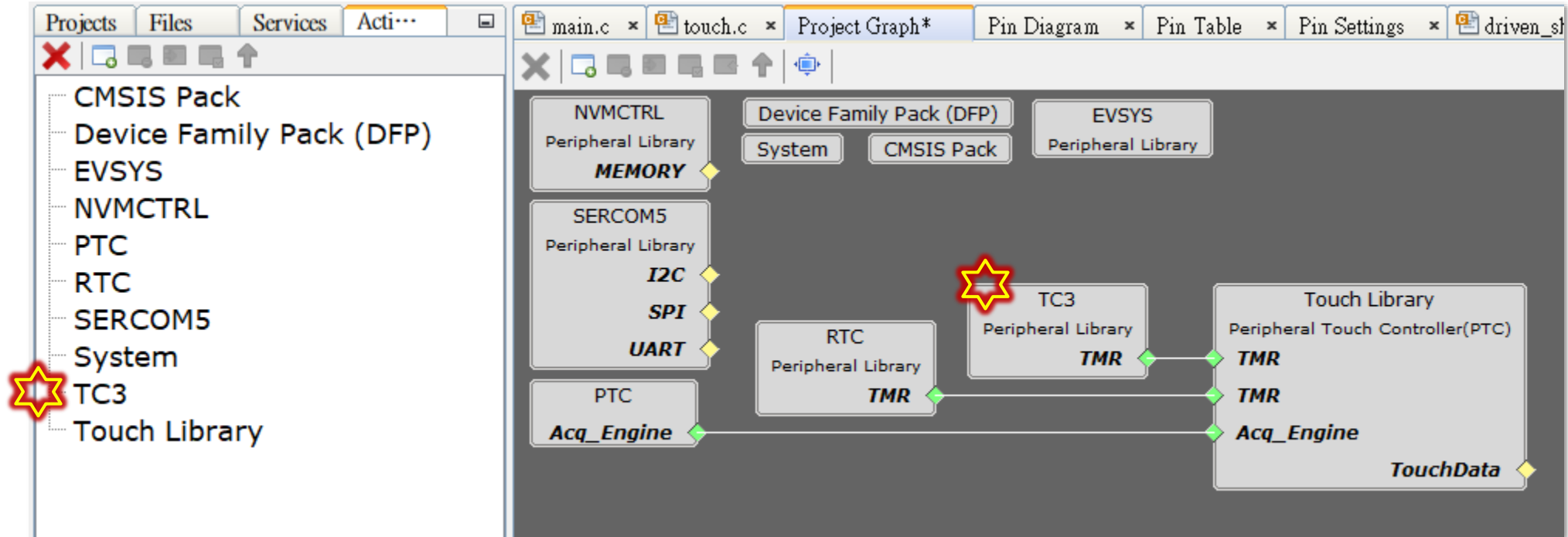
✂ Lab1-1 Qtouch Buttons

- Configure LEDs and Buttons in Harmony Pin Configuration as below.

main.c × Project Graph Pin Diagram × Pin Table × Pin Settings ×									
Order: Ports ▾ 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 ▾
23	PA14	★ BT1	GPIO ▾	Digital	In ▾	Low	☐	☐	NORMAL ▾
24	PA15	★ BT2	GPIO ▾	Digital	In ▾	Low	☐	☐	NORMAL ▾
26	PA17	★ LED3	GPIO ▾	Digital	Out ▾	Low	☐	☐	NORMAL ▾
29	PA20	★ LED1	GPIO ▾	Digital	Out ▾	Low	☐	☐	NORMAL ▾
30	PA21	★ LED2	GPIO ▾	Digital	Out ▾	Low	☐	☐	NORMAL ▾
47	PB02		PTC_Y8 ▾	Analog	High Impedance ▾	n/a	☐	☐	NORMAL ▾
37	PB22		SERCOM5_PAD2 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾
38	PB23		SERCOM5_PAD3 ▾	Digital	High Impedance ▾	n/a	☐	☐	NORMAL ▾

⚙️ Lab1-1 Qtouch Buttons

- Check [Harmony Graph](#) to see the difference after added Driven Shield.



Lab1-1 Qtouch Buttons

API functions in Touch.c

<code>uint8_t measurement_done_touch</code>	Touch acquisition done flag
<code>touch_process ()</code>	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.
<code>touch_init ()</code>	Initialization PTC, timer, and datastreamer modules of touch library.
<code>touch_timer_config ()</code>	Initialization necessary RTC timer for Touch in touch_init()
<code>touch_sensors_config ()</code>	Initialization of touch key sensors in touch_init()
<code>get_sensor_node_signal ()</code>	Get sensor node acquisition signals.
<code>update_sensor_node_signal ()</code>	Update sensor node acquisition signals.
<code>get_sensor_node_reference ()</code>	Get sensor node channel reference.
<code>update_sensor_node_reference ()</code>	Update sensor node channel reference.
<code>get_sensor_cc_val ()</code>	Get sensor node compensation capacitor value.
<code>update_sensor_cc_val ()</code>	Update sensor node compensation capacitor value.
<code>get_sensor_state ()</code>	Get sensor node state
<code>update_sensor_state ()</code>	Update sensor node state
<code>calibrate_node ()</code>	Sensor node calibration

API functions in driven_shield.c

<code>drivenshield_configure ()</code>	Initialization Driven Shield for Touch in touch_init()
--	--

Lab1-1 Qtouch 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 == 1)
    {
        measurement_done_touch = 0;
        // 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
    }
}
```

Lab1-1 Qtouch 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("  QTouch Button\r\n");
    myprintf("-----\r\n");
    myprintf("AKS  1    1    2\r\n");
    ...
}
```

⚙️ Lab1-1 Qtouch 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;
            }

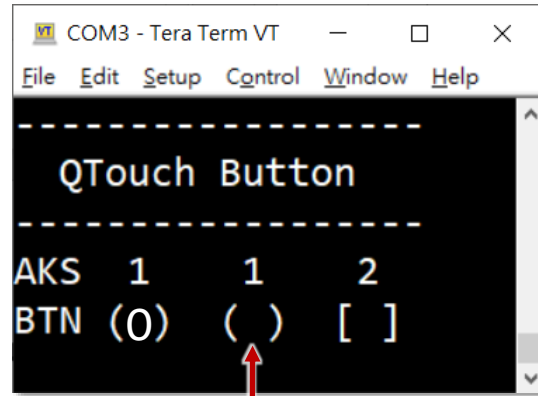
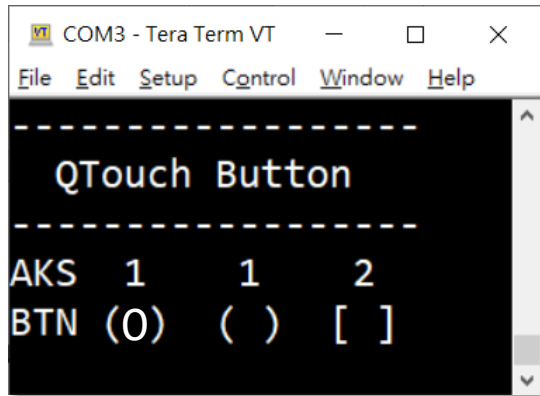
            switch( get_sensor_state(2) )
            {
                case QTM_KEY_STATE_DETECT:  KeyStatus[2]=true;  LED3_Set();  break;
                case QTM_KEY_STATE_NO_DET:   KeyStatus[2]=false; LED3_Clear(); break;
            }

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

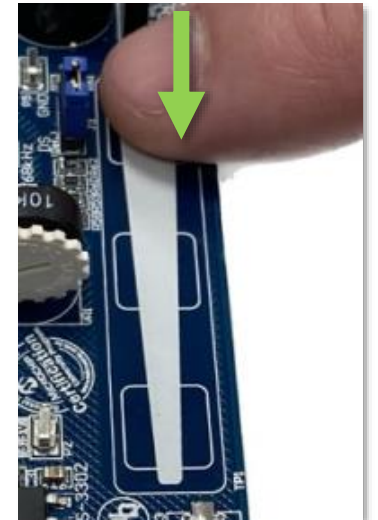
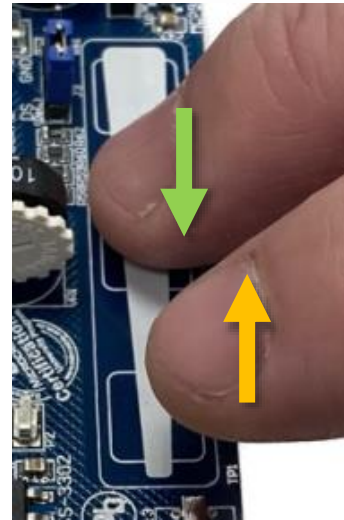
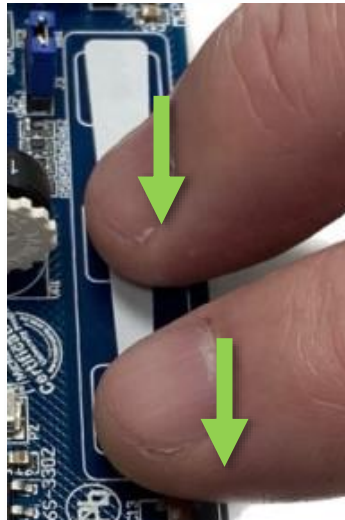
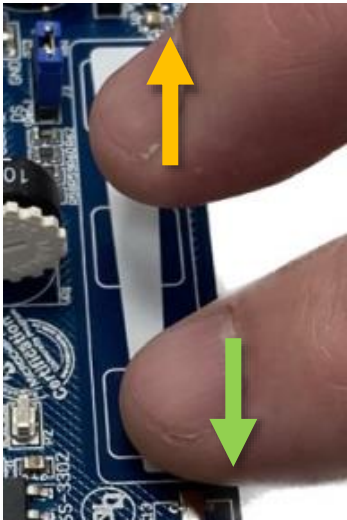
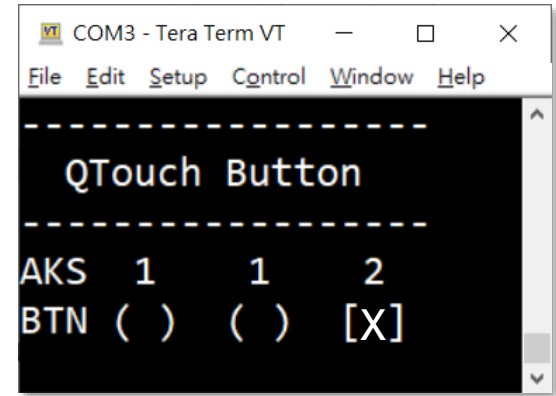
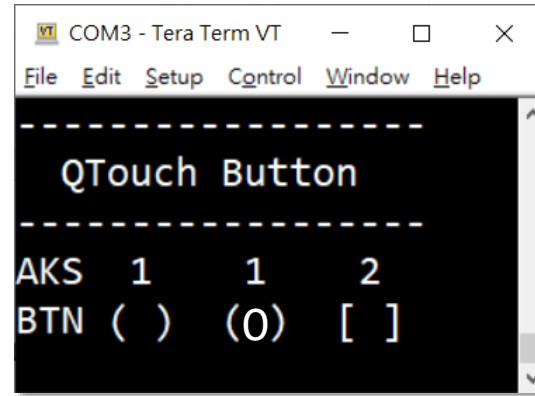
        }
        ...
    }
}
```

⚙️ Lab1-1 Qtouch Buttons

- Result of Touch Key implementation.



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



Lab1-1 Qtouch 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);
        }

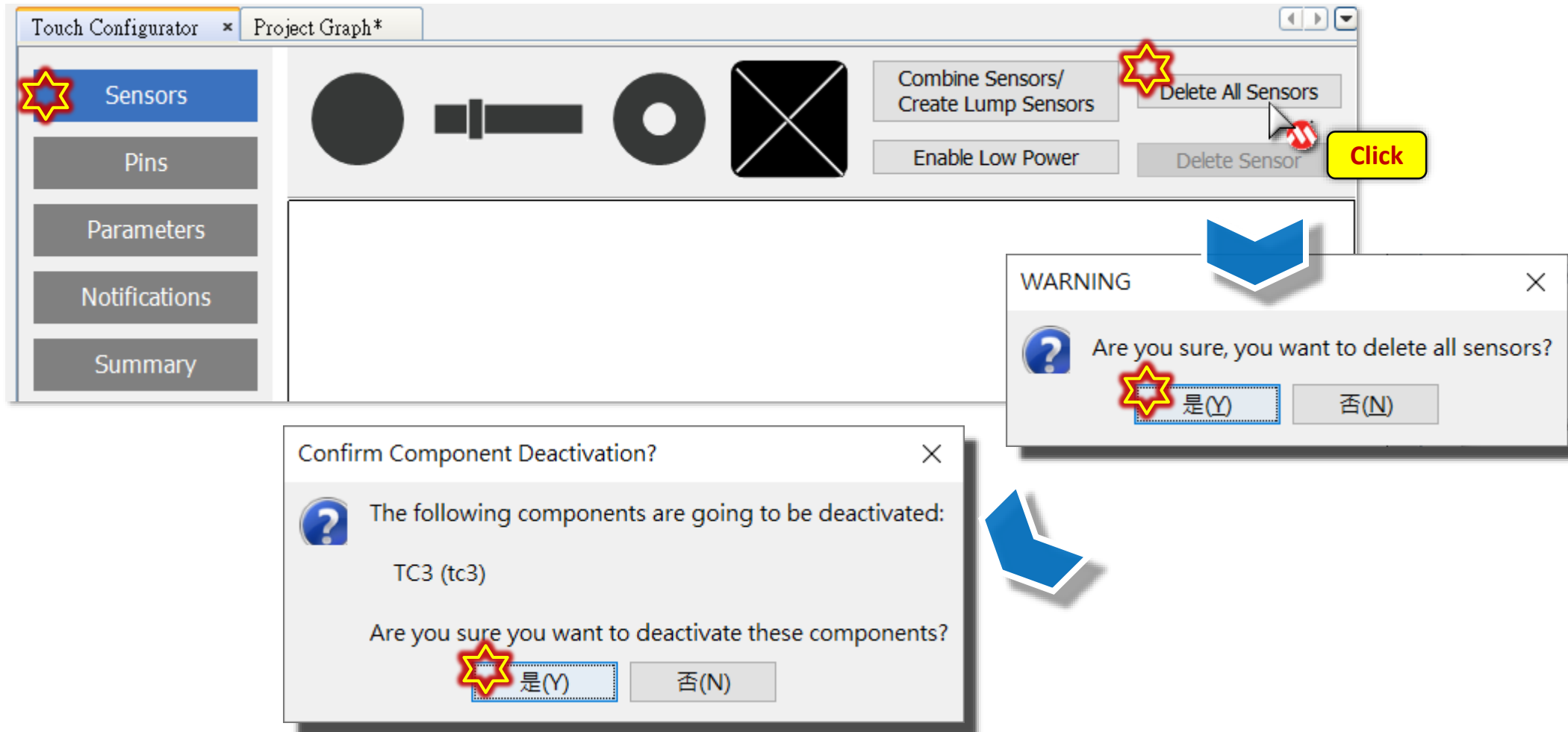
        if( measurement_done_touch == 1 )
        {
            measurement_done_touch = 0;
            switch( get_sensor_state(0) )
            {
                case QTM_KEY_STATE_DETECT:  KeyStatus[0]=true;  LED1_Set();  break;
                case QTM_KEY_STATE_NO_DET:   KeyStatus[0]=false; LED1_Clear();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

Lab1-2 Qtouch Slider

✖ Lab1-2 Qtouch Slider

- Return Harmony Touch Configurator **if you would like continue from Lab1-1.**
- In **Sensor** Tab, please click on **Delete All Sensors**.



✖ Lab1-2 Qtouch Slider

- Click on Slider(Scroller) Icon, default **Number of Sliders : 1** and **Number of Channels : 3**
- Click Add, if you are first configuring in Touch Configurator, a Warning dialog will show up to notice the clock will be auto set as default Touch Function request if select “Yes”

WARNING Only first time to configure in Touch Configurator

The clock for CPU and peripherals will be set as follows if YES option is selected.

If NO is selected, then user needs to configure the clock for PTC and RTC peripherals.

Refer to notification tab for details.

Oscillator	Prescaler Divisor	Frequency
OSC8M	2	4MHz
DFLL48M	1	48MHz(GCLK3, open loop, multi factor 1500)
Generic clock(Peripherals)	Source	Frequency
GCLK0(CPU)	DFLL48M	48MHz
GCLK1(Timer)	OSCULP32K	2.048KHz
GCLK2(PTC,UART)	OSC8M	4MHz
Wait states		
NVM states	2	

Yes No

Yes

Touch Configurator * Project Graph*

Sensors

Pins

Parameters

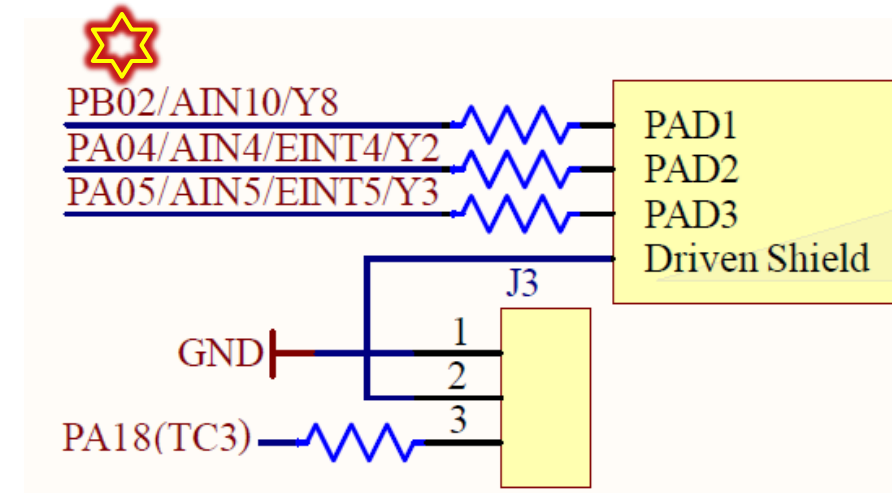
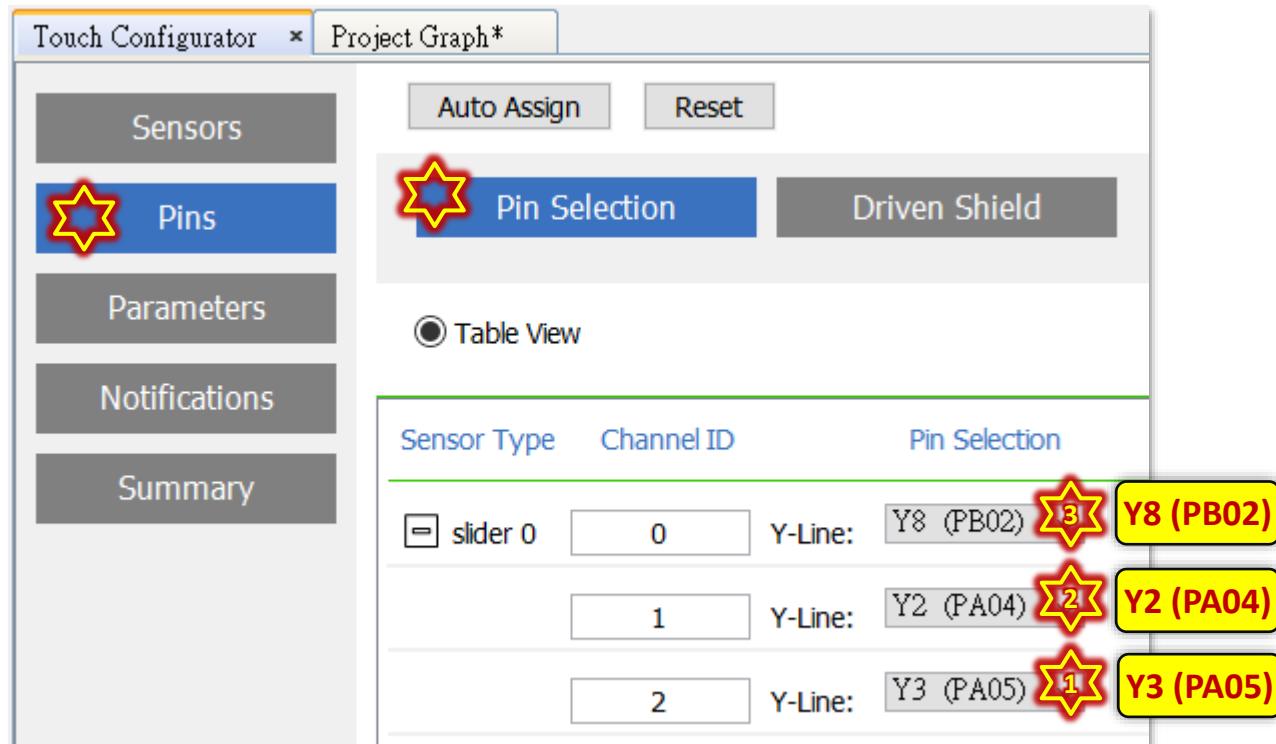
Notifications

Summary

0(3)

✂ Lab1-2 Qtouch Slider

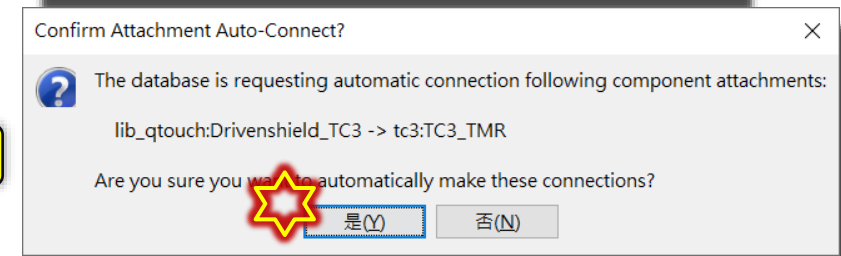
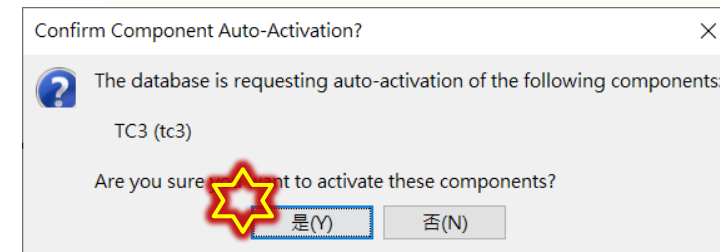
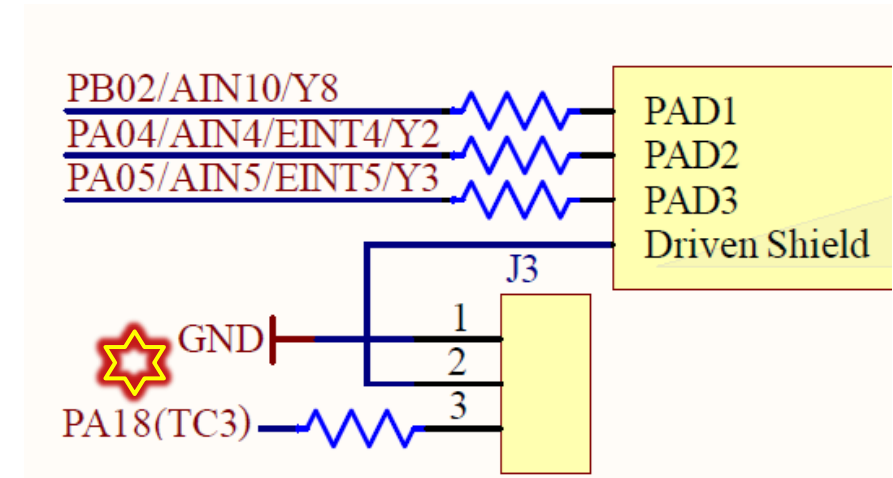
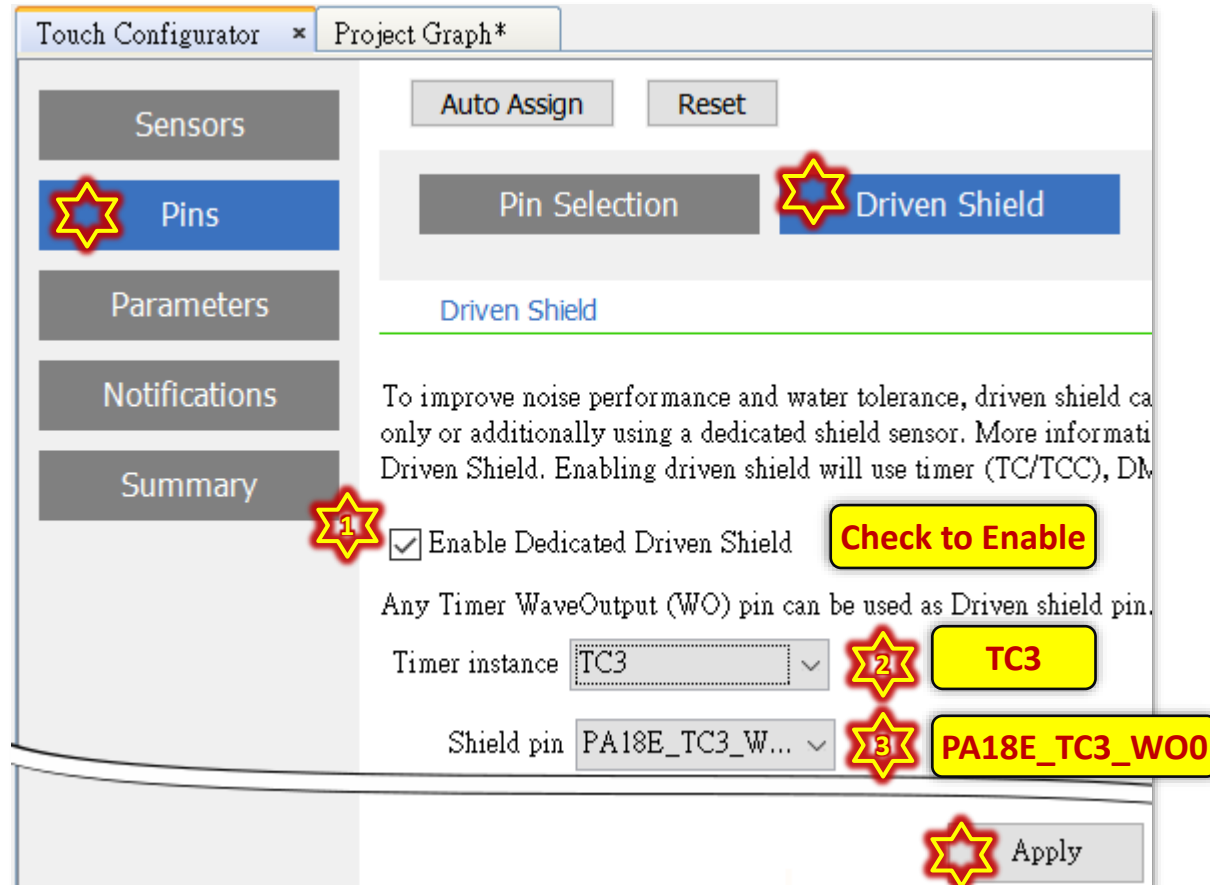
- In **Pins** Tab, Click on **Pin Selection** and configure button pins in below sequence.
 - Slider0 (Channel ID 2) : Y3 (PA05)
 - (Channel ID 1) : Y2 (PA04)
 - (Channel ID 0) : Y8 (PB02)



✖ Lab1-2 Qtouch Slider

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

- Enable Dedicated Driven Shield : ☒
- Timer instance : TC3
- Shield pin : PA18E_TC3_WO0



✖ Lab1-2 Qtouch Slider

- In **Parameters** Tab, disable **Sensor AKS**(Adjacent **K**ey **S**uppression) as below.
 - Channel 0/1/2 : No AKS

Touch Configurator × Project Graph*

Parameters

CHANNEL PARAMETERS

☐ Change all (Note: Enabling Change all, and changing a parameter will modify it for all sensors in the design. This does not apply to AKS)

Channel ID	Over Samples	Gain	Analog Gain	Series Resistor	PTC clock	Threshold	Hysteresis	Sensor AKS
Sensor List								
Sensor Id: slider 0								
0	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	No AKS
1	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	No AKS
2	16 samples	1	1	No resistor	1.0000 MHz	20	25 %	No AKS

PTC CLOCK FREQUENCY

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

GCLK/CLKPER: 4.0000 MHz

Prescaler: 4, 8, 16, 32

PTC Clock: 1.0000 MHz, 0.5000 MHz, 0.2500 MHz, 0.1250 MHz

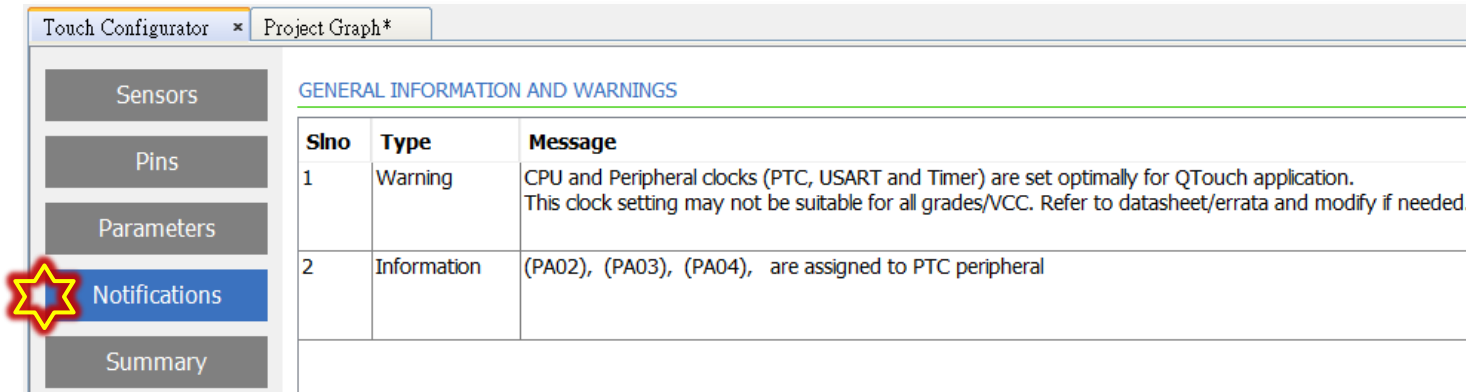
No AKS

No AKS

No AKS

⚙️ Lab1-2 Qtouch Slider

- Check **Notifications** and **Summary** Tabs to figure out current configuration of Touch.

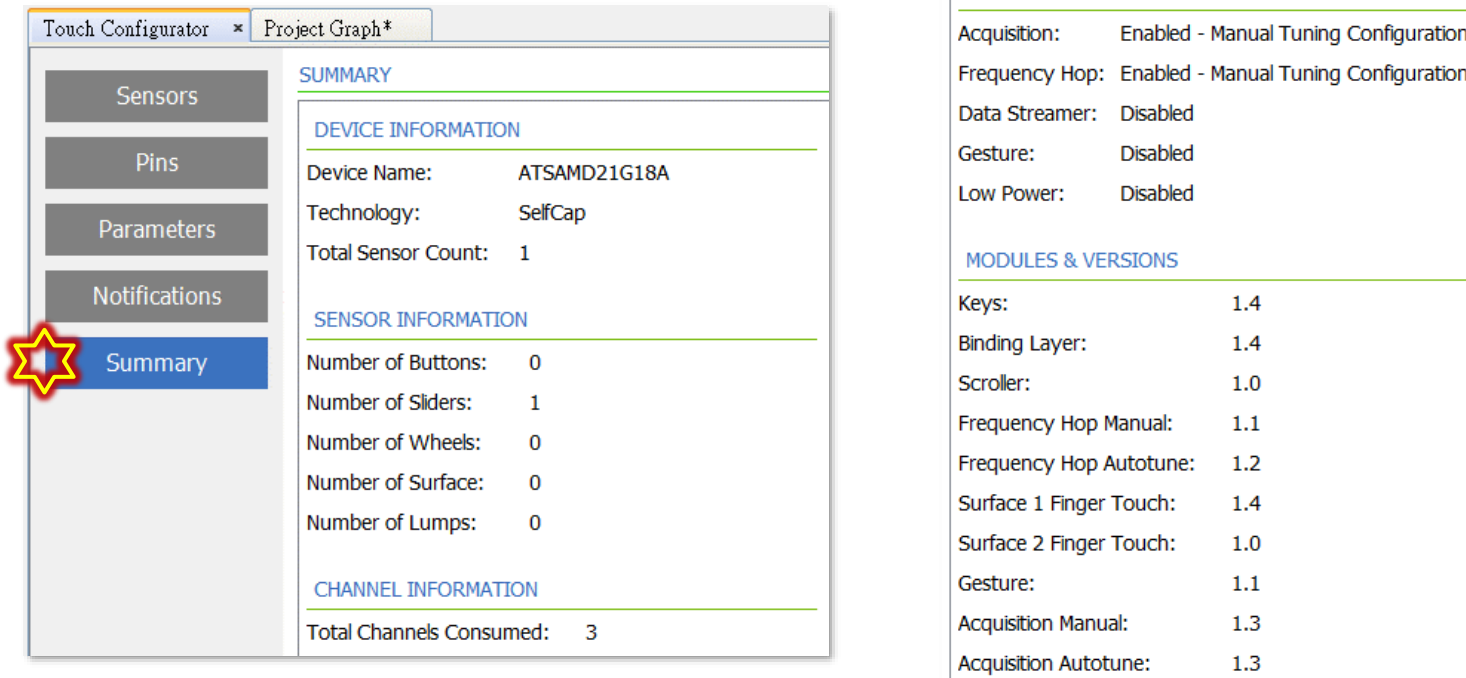


Touch Configurator * Project Graph*

Sensors
Pins
Parameters
Notifications
Summary

GENERAL INFORMATION AND WARNINGS

Sno	Type	Message
1	Warning	CPU and Peripheral clocks (PTC, USART and Timer) are set optimally for QTouch application. This clock setting may not be suitable for all grades/VCC. Refer to datasheet/errata and modify if needed.
2	Information	(PA02), (PA03), (PA04), are assigned to PTC peripheral



Touch Configurator * Project Graph*

Sensors
Pins
Parameters
Notifications
Summary

SUMMARY

DEVICE INFORMATION

Device Name: ATSAM21G18A
Technology: SelfCap
Total Sensor Count: 1

SENSOR INFORMATION

Number of Buttons: 0
Number of Sliders: 1
Number of Wheels: 0
Number of Surface: 0
Number of Lumps: 0

CHANNEL INFORMATION

Total Channels Consumed: 3

OTHER INFORMATION

Acquisition: Enabled - Manual Tuning Configuration
Frequency Hop: Enabled - Manual Tuning Configuration
Data Streamer: Disabled
Gesture: Disabled
Low Power: Disabled

MODULES & 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

✖ Lab1-2 Qtouch Slider

- Configure LEDs and Buttons in Harmony Pin Configuration as below. **(New Project Only)**

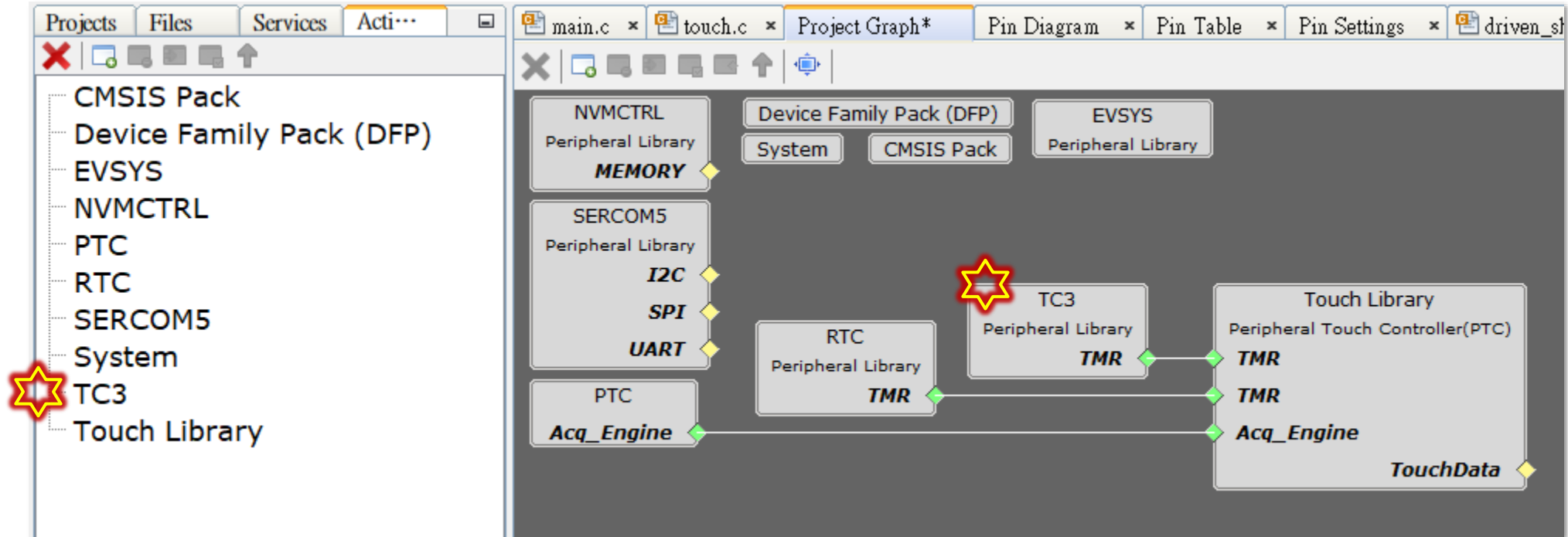
main.c * Project Graph Pin Diagram * Pin Table * Pin Settings *

Order: Ports 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
23	PA14	BT1	GPIO	Digital	In	Low	☐	☐	NORMAL
24	PA15	BT2	GPIO	Digital	In	Low	☐	☐	NORMAL
26	PA17	LED3	GPIO	Digital	Out	Low	☐	☐	NORMAL
29	PA20	LED1	GPIO	Digital	Out	Low	☐	☐	NORMAL
30	PA21	LED2	GPIO	Digital	Out	Low	☐	☐	NORMAL
47	PB02		PTC_Y8	Analog	High Impedance	n/a	☐	☐	NORMAL
37	PB22		SERCOM5_PAD2	Digital	High Impedance	n/a	☐	☐	NORMAL
38	PB23		SERCOM5_PAD3	Digital	High Impedance	n/a	☐	☐	NORMAL

✖ Lab1-2 Qtouch Slider

- Check [Harmony Graph](#) to see the difference after added Driven Shield.



Lab1-2 Qtouch 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()

⚙️ Lab1-1 Qtouch 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 == 1)
    {
        measurement_done_touch = 0;
        // process touch data
    }
}

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
    }
}
```

```
scroller_status = get_scroller_state(0);
scroller_position = get_scroller_position(0);

//Example: 8 bit scroller resolution. Modify as per requirement.
scroller_position = scroller_position >> 5;

//LED_OFF
if ( 0u != scroller_status)
{
    switch (scroller_position)
    {
        case 0: //LED0_ON
            break;
        case 1: //LED1_ON
            break;
        case 2: //LED2_ON
            break;
        case 3: //LED3_ON
            break;
        case 4: //LED4_ON
            break;
        case 5: //LED5_ON
            break;
        case 6: //LED6_ON
            break;
        case 7: //LED7_ON
            break;
        default:
            //LED_OFF
            break;
    }
}
```

Lab1-2 Qtouch 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("  QTouch Slider\r\n");
    myprintf("-----\r\n");
    myprintf("_____");
    ...
}
```

⚙️ Lab1-2 Qtouch 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;
                }
            }
        }
        ...
    }
}
```

Lab1-2 Qtouch Slider

- Implement manual Touch Calibration feature.

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

    while ( true )
    {
        touch_process();

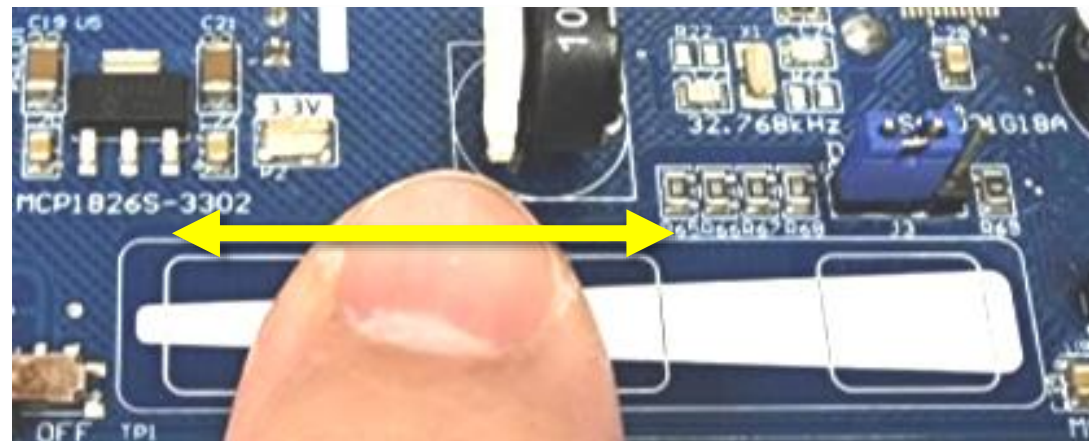
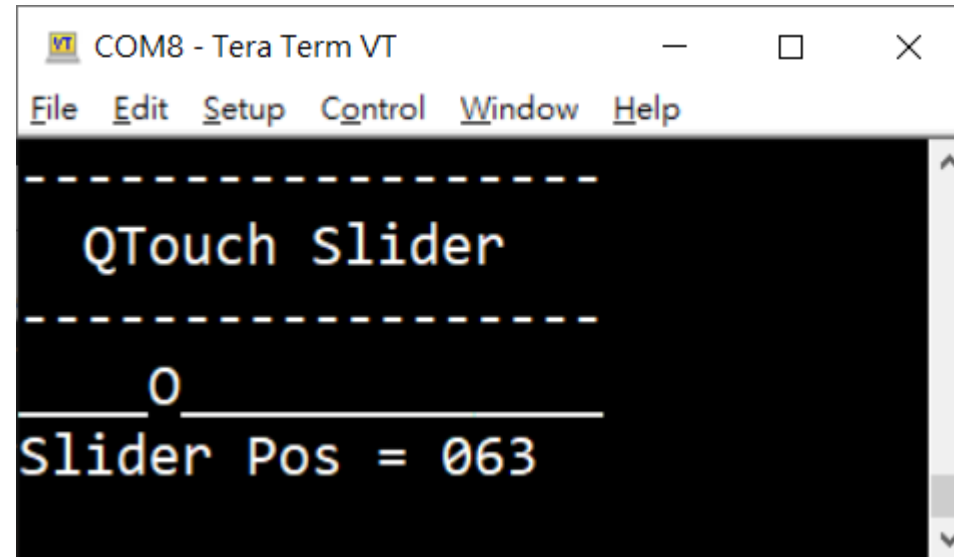
        if(BT1_Get()==0)
        {
            while(BT1_Get()==0);
            calibrate_node(0);
            calibrate_node(1);
            calibrate_node(2);
        }

        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;
                    ...
                }
            }
        }
    }
}
```

✖ Lab1-2 Qtouch Slider

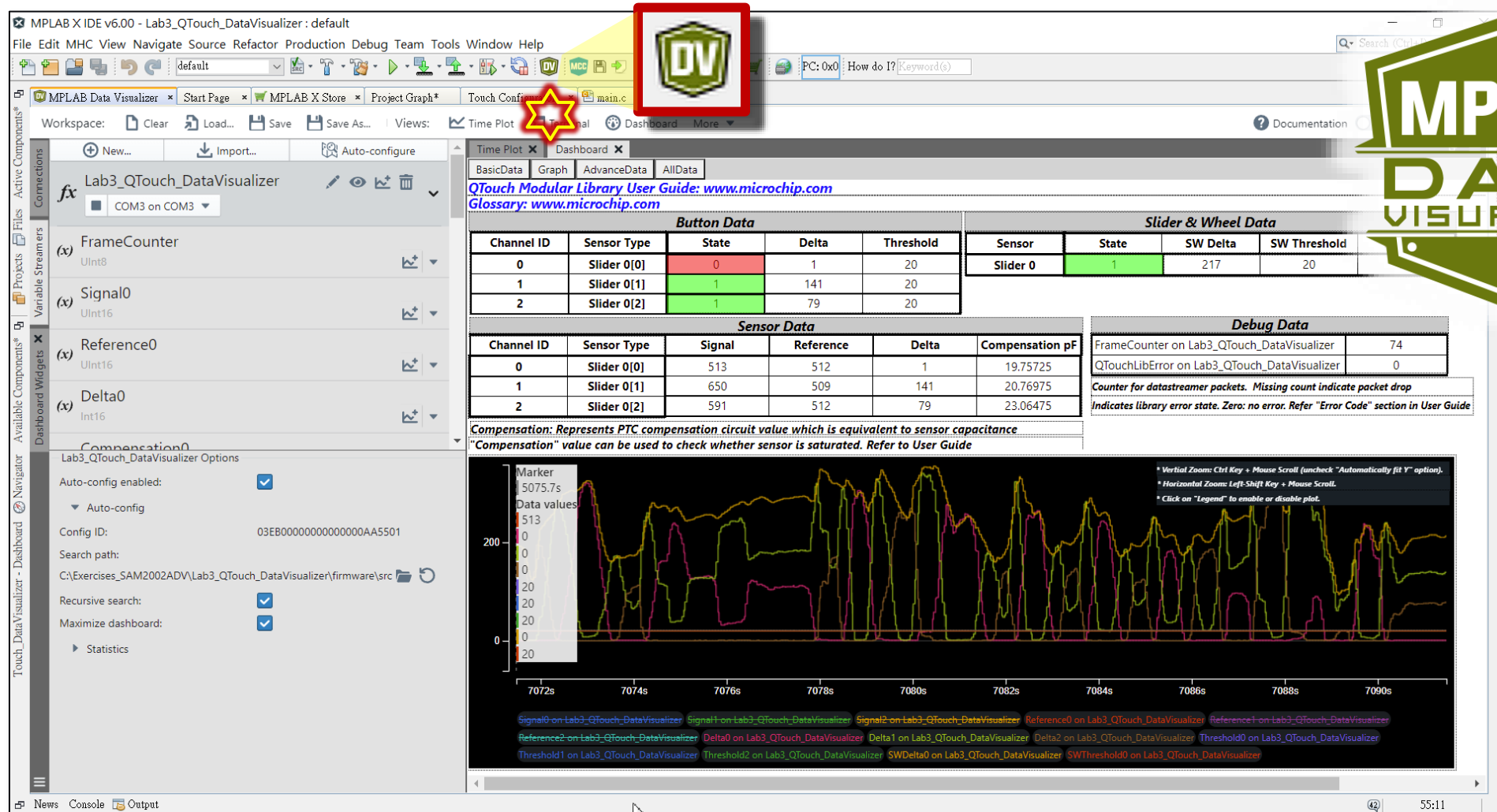
- Result of Touch Slider(Scroller) implementation.



Lab1-3 Data Visualizer for Touch

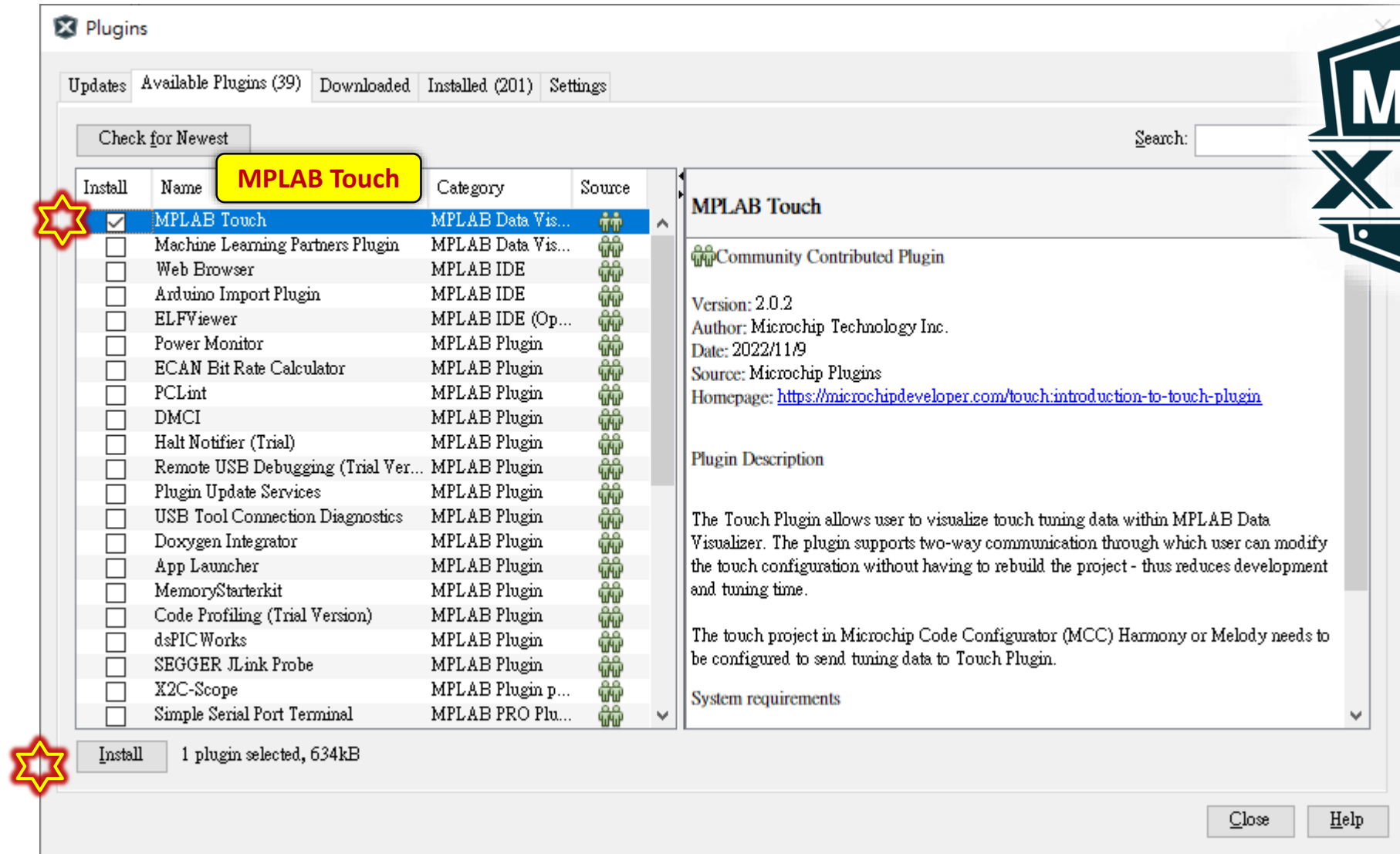
⚡ Lab1-3 Data Visualizer for Touch

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



⚙️ Lab1-3 Data Visualizer for Touch

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



⚙️ Lab1-3 Data Visualizer for Touch

- Return Harmony Touch Configurator **if you would like continue from Lab1-2.**
- In **Parameters** Tab, please click on **Tune**.
- **Enable Touch Tuning Connection** and select to **Touch Visualization (unidirectional...)**

The screenshot shows the 'Parameters' tab of the Harmony Touch Configurator. The interface includes a sidebar with 'Parameters' highlighted (marked with a yellow star and '1'). The main area has a 'TOUCH TUNING' section with a 'Tune' button (marked with a yellow star and '2'). Below this, the 'Enable Touch Tuning Connection' checkbox is checked (marked with a yellow star and '3'). The 'Touch Visualization (unidirectional, requires UART Tx)' radio button is selected (marked with a yellow star and '4'). A yellow callout box labeled 'Touch Visualization' points to this option. The bottom section, 'CHARGE TIME AUTO TUNE', has an unchecked 'Enable Charge Time Autotune' checkbox.

Touch Configurator * Project Graph*

Sensors Channel Sensor Sliders and w... Tune

Pins

Parameters

Notifications

Summary

TOUCH TUNING

☒ Enable Touch Tuning Connection

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

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

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 require [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

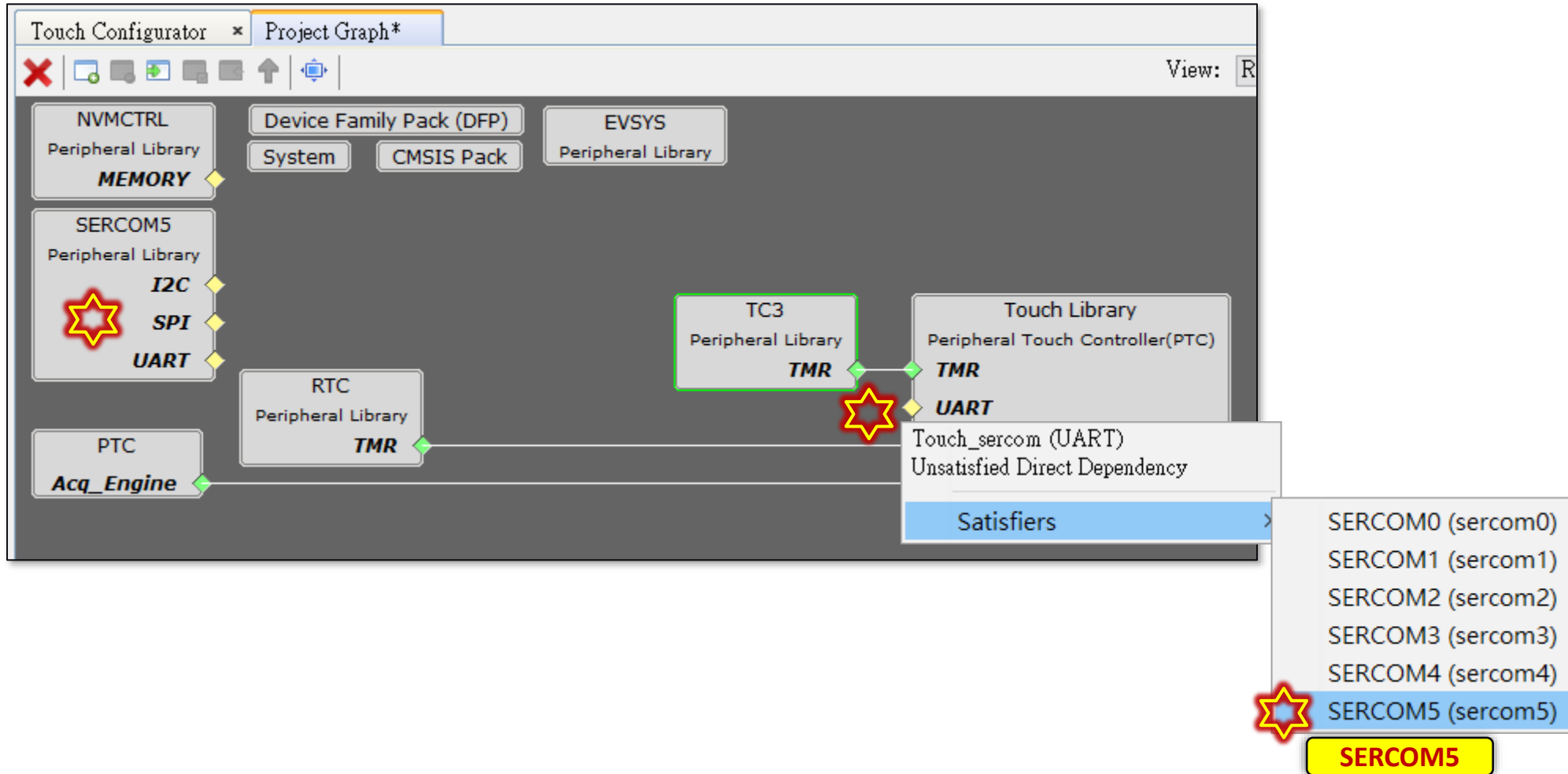
This should be enabled and used for project development and validation and be disabled for production code to remove overhead, power consumption and flash/RAM footp

CHARGE TIME AUTO TUNE

☐ Enable Charge Time Autotune

⚙️ Lab1-3 Data Visualizer for Touch

- A new **UART** connection will request for Touch Library, please connect it to **SERCOM5**



⚙️ Lab1-3 Data Visualizer for Touch

- Configure **SERCOM5** to use Blocking Mode **if you are continuing from Lab1-2.**

The screenshot displays the Touch Configurator interface. The top window, 'Project Graph*', shows a block diagram of the system components. The components include NVMCTRL (Peripheral Library MEMORY), Device Family Pack (DFP) (System, CMSIS Pack), EVSYS (Peripheral Library), PTC (Acq_Engine), RTC (Peripheral Library TMR), SERCOM5 (Peripheral Library UART), TC3 (Peripheral Library TMR), and Touch (Peripheral Library TMR, UART, TMR, Acq_Engine). The SERCOM5 block is highlighted with a red star, and the TC3 block is also highlighted with a red star. The Touch block is highlighted with a blue star.

The bottom window, 'Configuration Options*', shows the configuration for SERCOM5. The 'Select SERCOM Operation Mode' is set to 'USART with internal Clock'. The 'Operating Mode' is set to 'Blocking mode', which is highlighted with a red star and a yellow 'Blocking Mode' button. The 'Receive Enable' and 'Transmit Enable' checkboxes are checked. The 'Frame Format' is set to 'USART frame'. The 'Baud Rate in Hz' is set to 115,200. The 'Parity Mode' is set to 'No Parity'. The 'Character Size' is set to '8 Bits'. The 'Stop Bit Mode' is set to 'One Stop Bit'. The 'Receive Pinout' is set to 'SERCOM PAD[3] is used for data reception', which is highlighted with a blue star. The 'Transmit Pinout' is set to 'PAD[2] = TxD; PAD[3] = XCK'. The 'Enable Run in Standby' checkbox is unchecked.

⚙️ Lab1-3 Data Visualizer for Touch

- Modify `myprintf()` to use `TransmitComplete()` function of SERCOM **Blocking Mode** and **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 )
{
    ...

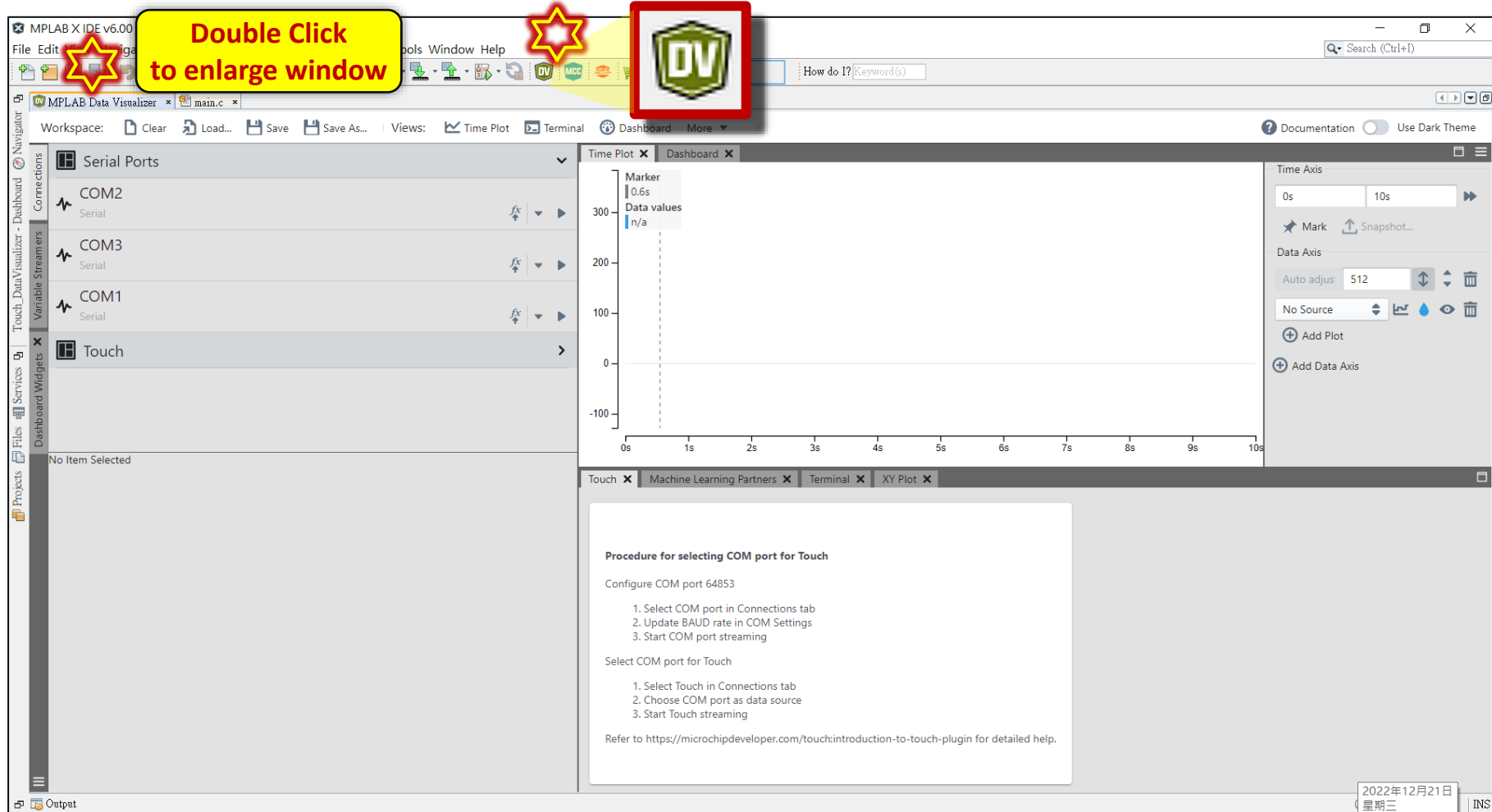
    while ( true )
    {
        touch_process();

        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 )
```

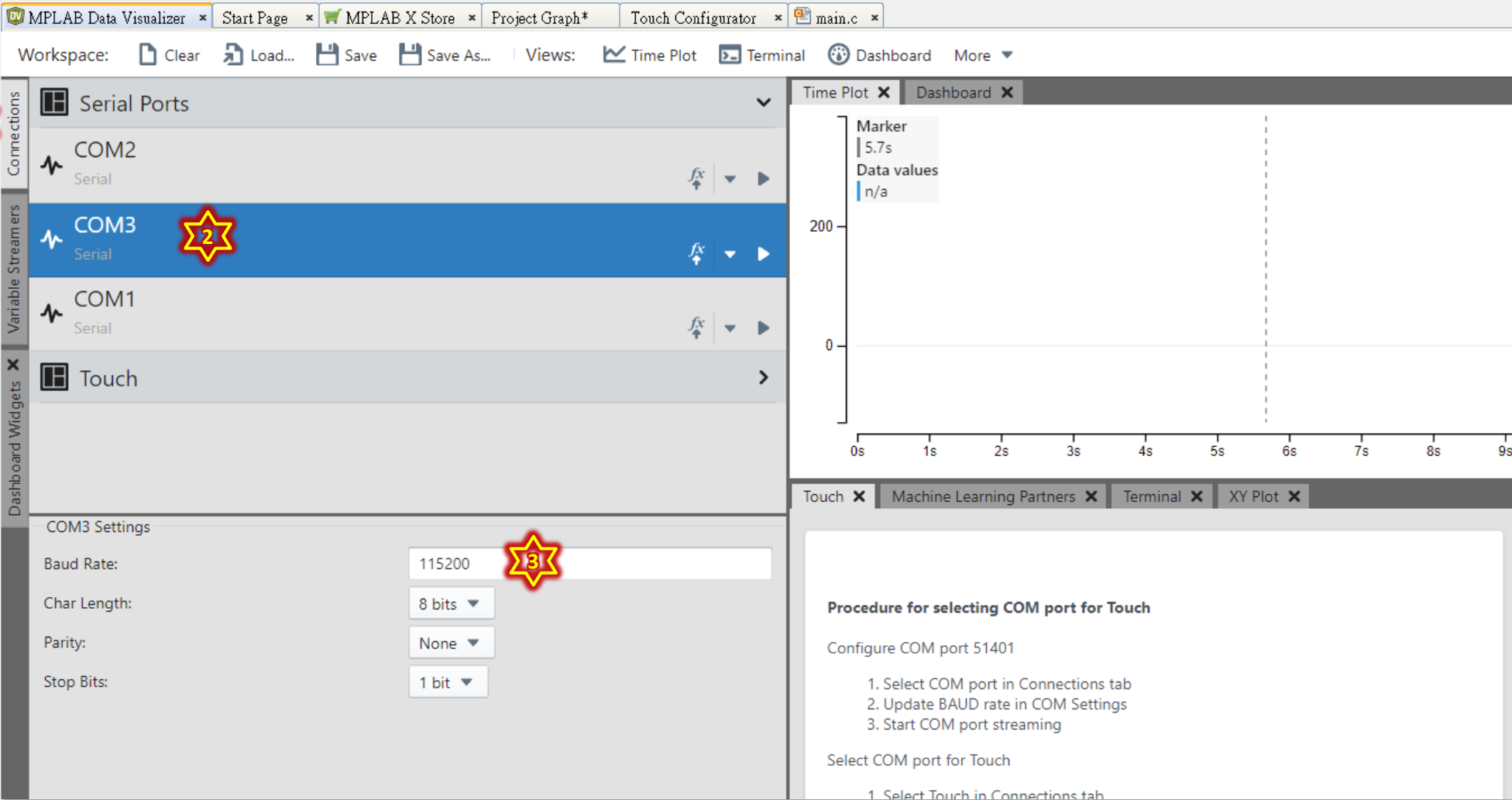
⚙️ Lab1-3 Data Visualizer for Touch

- 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.



Lab1-3 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.



The screenshot shows the MPLAB Data Visualizer interface with the following components:

- Connections Tab:** Located on the left, it lists three serial ports: COM2, COM3, and COM1. COM3 is highlighted in blue and marked with a yellow star labeled '2'.
- COM3 Settings:** Located at the bottom left, it shows the Baud Rate set to 115200, Char Length to 8 bits, Parity to None, and Stop Bits to 1 bit. This section is marked with a yellow star labeled '3'.
- Time Plot:** Located on the right, it shows a graph with a vertical dashed line at 5.7s and a data value of n/a.
- Terminal:** Located at the bottom right, it contains a procedure for selecting the COM port for Touch.

Procedure for selecting COM port for Touch

Configure COM port 51401

1. Select COM port in Connections tab
2. Update BAUD rate in COM Settings
3. Start COM port streaming

Select COM port for Touch

1. Select Touch in Connections tab

⚙️ Lab1-3 Data Visualizer for Touch

- Select Protocol Source of Touch Session to use the same **COM Port** in above.
- After select COM Port of Touch Session, then it will start to communication.

The screenshot displays the MPLAB Data Visualizer interface. The 'Connections' tab is active, showing a list of serial ports: COM2, COM3, and COM1. COM3 is selected. Below this, the 'Touch' section is expanded, showing a 'Touch Session' configuration. A dropdown menu is open for 'Protocol Source', showing options: 'No Source', 'COM1 on COM1', 'COM2 on COM2', and 'COM3 on COM3'. A red star with the number '2' is placed over the 'COM3 on COM3' option, indicating the correct selection. A red star with the number '1' is placed over the 'COM3' entry in the 'Connections' list. The right side of the interface shows a 'Time Plot' and a 'Dashboard' tab. The 'Time Plot' shows a graph with a vertical dashed line at 5.7s. The 'Dashboard' tab shows a 'Procedure for selecting COM port for Touch' section with instructions: 'Configure COM port 51401', 'Select COM port for Touch', and '1. Select Touch in Connections tab'.

Workspace: Clear Load... Save Save As... Views: Time Plot Terminal Dashboard More

Connections

Serial Ports

COM2 Serial

COM3 Serial

COM1 Serial

Touch

Touch Session

COM3 on COM3

Filter...

No Source

COM1 on COM1

COM2 on COM2

COM3 on COM3

115200

8 bits

None

1 bit

Time Plot

Marker

5.7s

Data values

n/a

200

0

0s 1s 2s 3s 4s 5s 6s 7s 8s 9s

Touch Machine Learning Partners Terminal XY Plot

Procedure for selecting COM port for Touch

Configure COM port 51401

Select COM port for Touch

1. Select Touch in Connections tab

⚙️ Lab1-3 Data Visualizer for Touch

- Click to Variable Streamers, then click on **Auto-configure** to add a new Streamer.
- Please 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...', and 'Auto-configure'. The 'Auto-configure' button is highlighted with a yellow star labeled '2'. On the left sidebar, the 'Variable Streamers' tab is selected, indicated by a yellow star labeled '1'. A dropdown menu is open, showing options for data sources: 'No Source', 'Touch Session on Touch', 'COM1 on COM1', 'COM2 on COM2', and 'COM3 on COM3'. The 'COM3 on COM3' option is highlighted with a yellow star labeled '3'. Below the dropdown, the 'Auto-configure Options' section shows 'Auto-config enabled' checked. A yellow warning box with the text 'Missing config files' is prominently displayed. The 'Config ID' is '03EB000000000000AA5501' and the 'Search path' is 'C:\Exercises_SAM2002ADV\Lab3_QTouch_DataVisualizer\firmware\Lab3_QTouch_DataVisualizer.X'. The right panel shows a 'Time Plot' with a y-axis from 0 to 200 and an x-axis from 0s to 9s. A legend indicates 'Marker' at 0.2s and 'Data values' as 'n/a'. At the bottom right, a 'Procedure for selecting COM port for Touch' is provided, listing steps: 1. Select COM port in Connections tab, 2. Update BAUD rate in COM Settings, 3. Start COM port streaming.

Missing config files

Procedure for selecting COM port for Touch

Configure COM port 51401

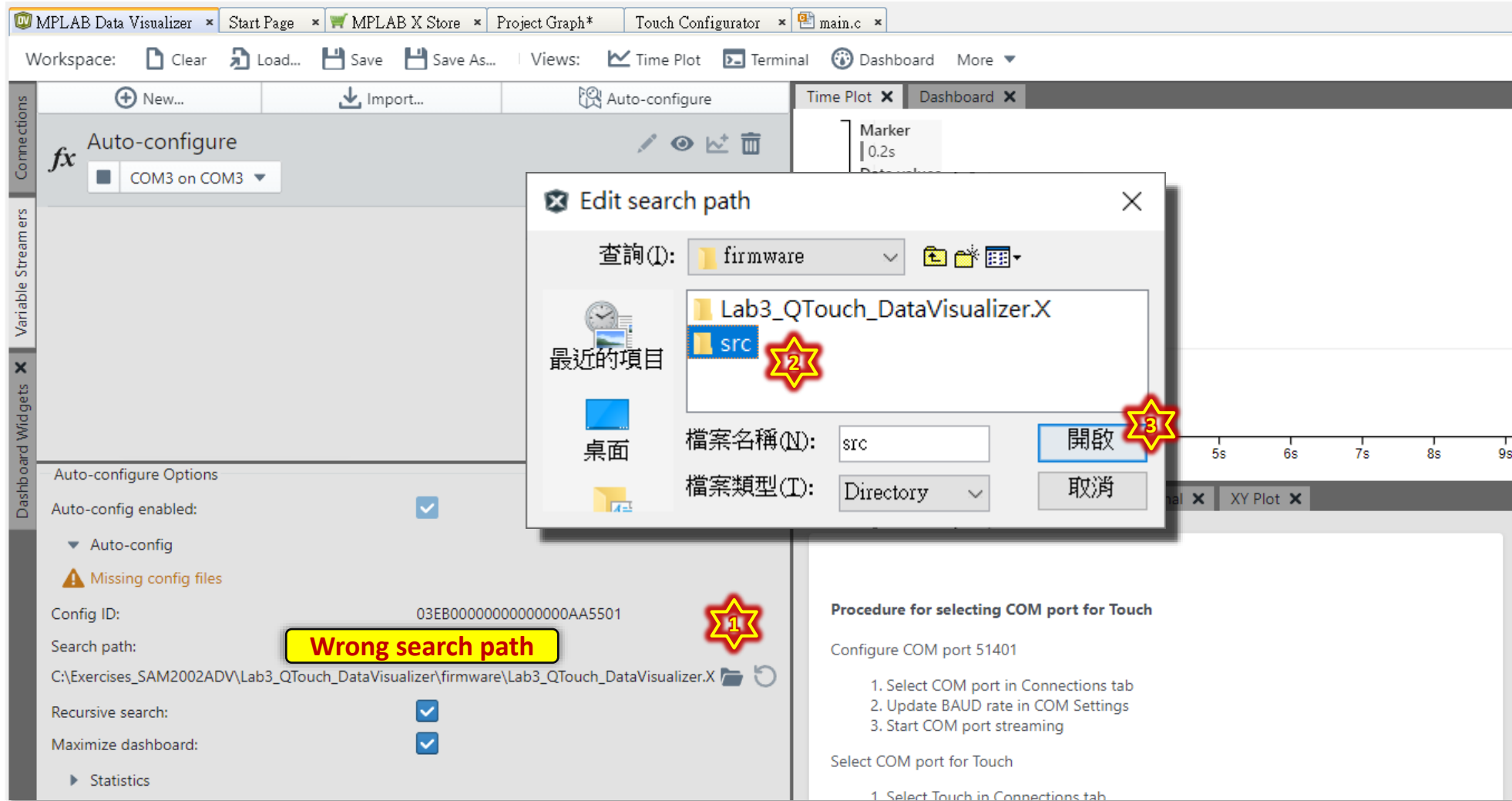
1. Select COM port in Connections tab
2. Update BAUD rate in COM Settings
3. Start COM port streaming

Select COM port for Touch

1. Select Touch in Connections tab

✖ Lab1-3 Data Visualizer for Touch

- The default Streamer Configuration file **Search Path** is wrong, please click on Icon  to change to upper level of folder and set **Search Path** on **\src** folder of project.



The screenshot shows the MPLAB Data Visualizer interface. The 'Auto-configure' tab is active, displaying the 'Search path' field with the value 'C:\Exercises_SAM2002ADV\Lab3_QTouch_DataVisualizer\firmware\Lab3_QTouch_DataVisualizer.X'. A yellow star icon with the number '1' is placed next to this field, and a yellow box with the text 'Wrong search path' is overlaid on it.

An 'Edit search path' dialog box is open, showing a list of recent projects. The 'src' folder is highlighted with a yellow star icon and the number '2'. The 'Open' button is highlighted with a yellow star icon and the number '3'.

The 'Edit search path' dialog box includes the following fields:

- 查詢(I): firmware
- Lab3_QTouch_DataVisualizer.X
- src
- 檔案名稱(N): src
- 檔案類型(T): Directory
- 開啟
- 取消

The 'Auto-configure Options' section shows:

- Auto-config enabled: ☒
- Auto-config: ☒
- Missing config files: 
- Config ID: 03EB000000000000AA5501
- Search path: C:\Exercises_SAM2002ADV\Lab3_QTouch_DataVisualizer\firmware\Lab3_QTouch_DataVisualizer.X
- Recursive search: ☒
- Maximize dashboard: ☒
- Statistics: ☐

The 'Procedure for selecting COM port for Touch' section includes the following steps:

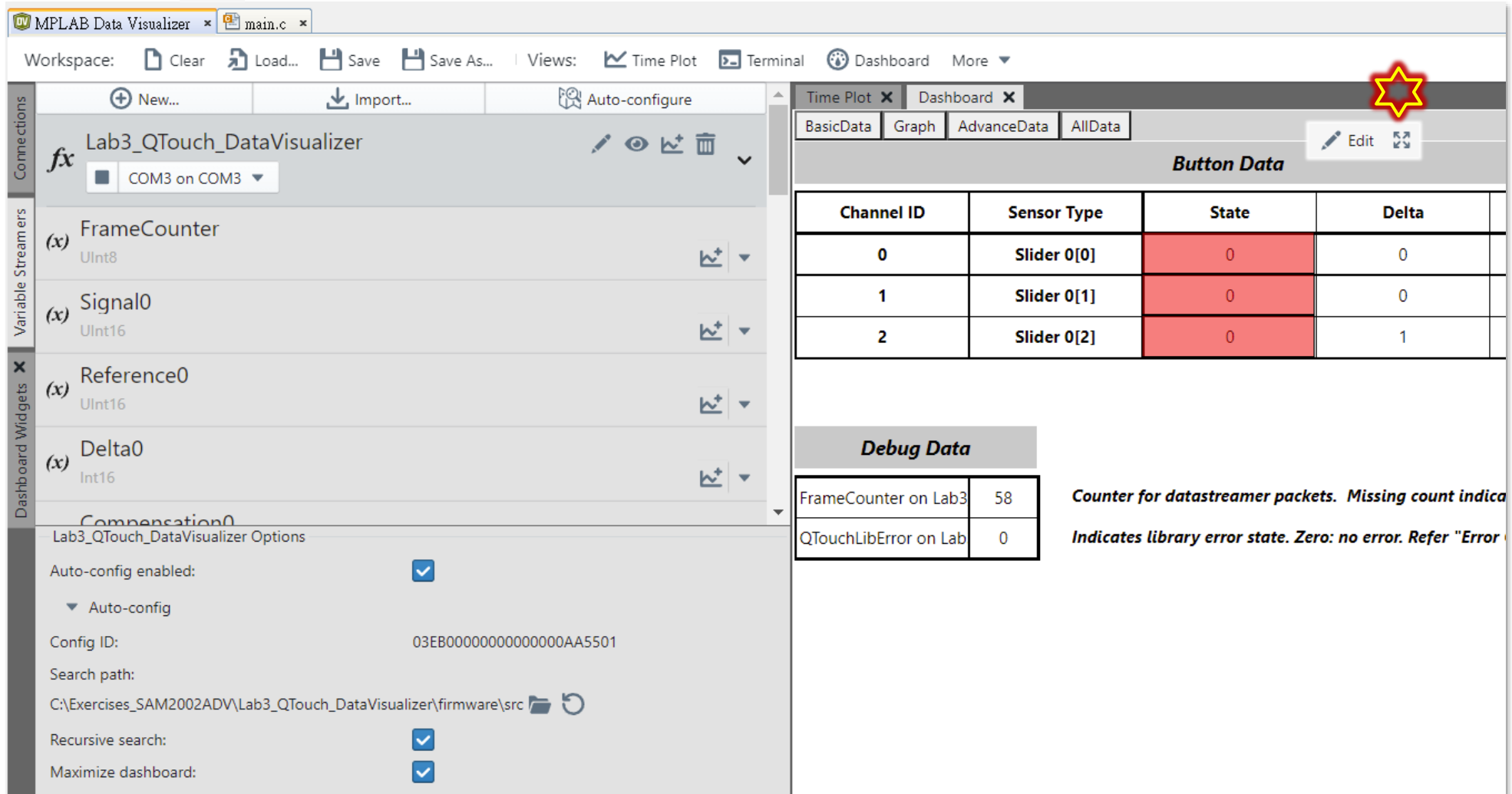
1. Select COM port in Connections tab
2. Update BAUD rate in COM Settings
3. Start COM port streaming

Select COM port for Touch

1. Select Touch in Connections tab

⚙️ Lab1-3 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.



The screenshot displays the MPLAB Data Visualizer interface. The top menu bar includes options like 'Workspace', 'Clear', 'Load...', 'Save', 'Save As...', 'Views', 'Time Plot', 'Terminal', 'Dashboard', and 'More'. The left sidebar shows 'Connections' with 'Lab3_QTouch_DataVisualizer' and 'COM3 on COM3'. Below this, 'Variable Streamers' lists 'FrameCounter' (UInt8), 'Signal0' (UInt16), 'Reference0' (UInt16), and 'Delta0' (Int16). The 'Dashboard Widgets' section shows 'Compensation0' and 'Lab3_QTouch_DataVisualizer Options' with checkboxes for 'Auto-config enabled', 'Recursive search', and 'Maximize dashboard'. The main dashboard area has tabs for 'Time Plot', 'Dashboard', 'BasicData', 'Graph', 'AdvanceData', and 'AllData'. The 'Dashboard' tab is active, showing a 'Button Data' table and a 'Debug Data' table. A yellow star icon is highlighted in the top right corner of the dashboard area.

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

Debug Data	
FrameCounter on Lab3	58
QTouchLibError on Lab	0

Counter for datastreamer packets. Missing count indicates library error state. Zero: no error. Refer "Error"

Lab1-3 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	0	20
1	Slider 0[1]	0	0	20
2	Slider 0[2]	0	0	20

Slider & Wheel Data

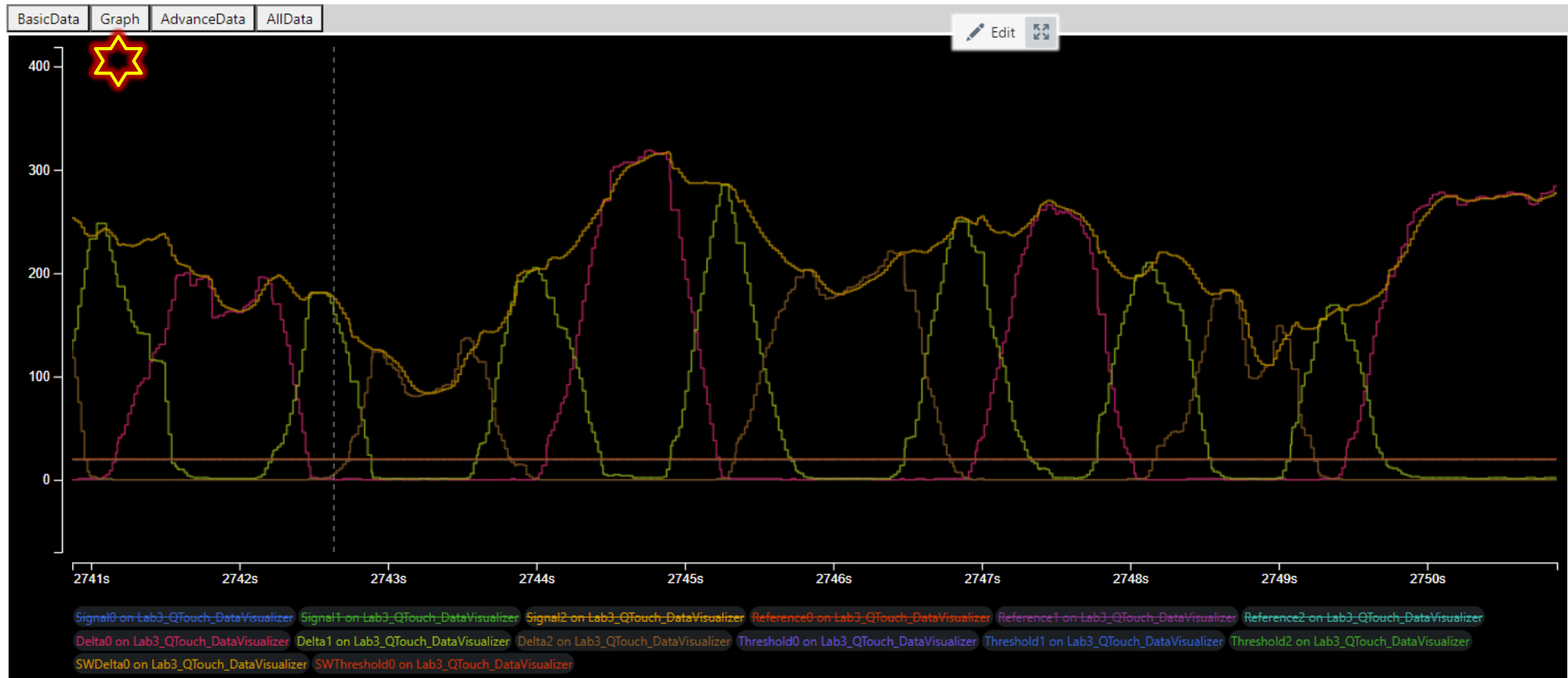
Sensor	State	SW Delta	SW Threshold	Position
Slider 0	0	0	20	163

Debug Data

FrameCounter on Lab3	177	Counter for datastreamer packets. Missing count indicate packet drop
QTouchLibError on Lab	0	Indicates library error state. Zero: no error. Refer "Error Code" section in User Guide

⚙️ Lab1-3 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.

Lab1-3 Data Visualizer for Touch

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



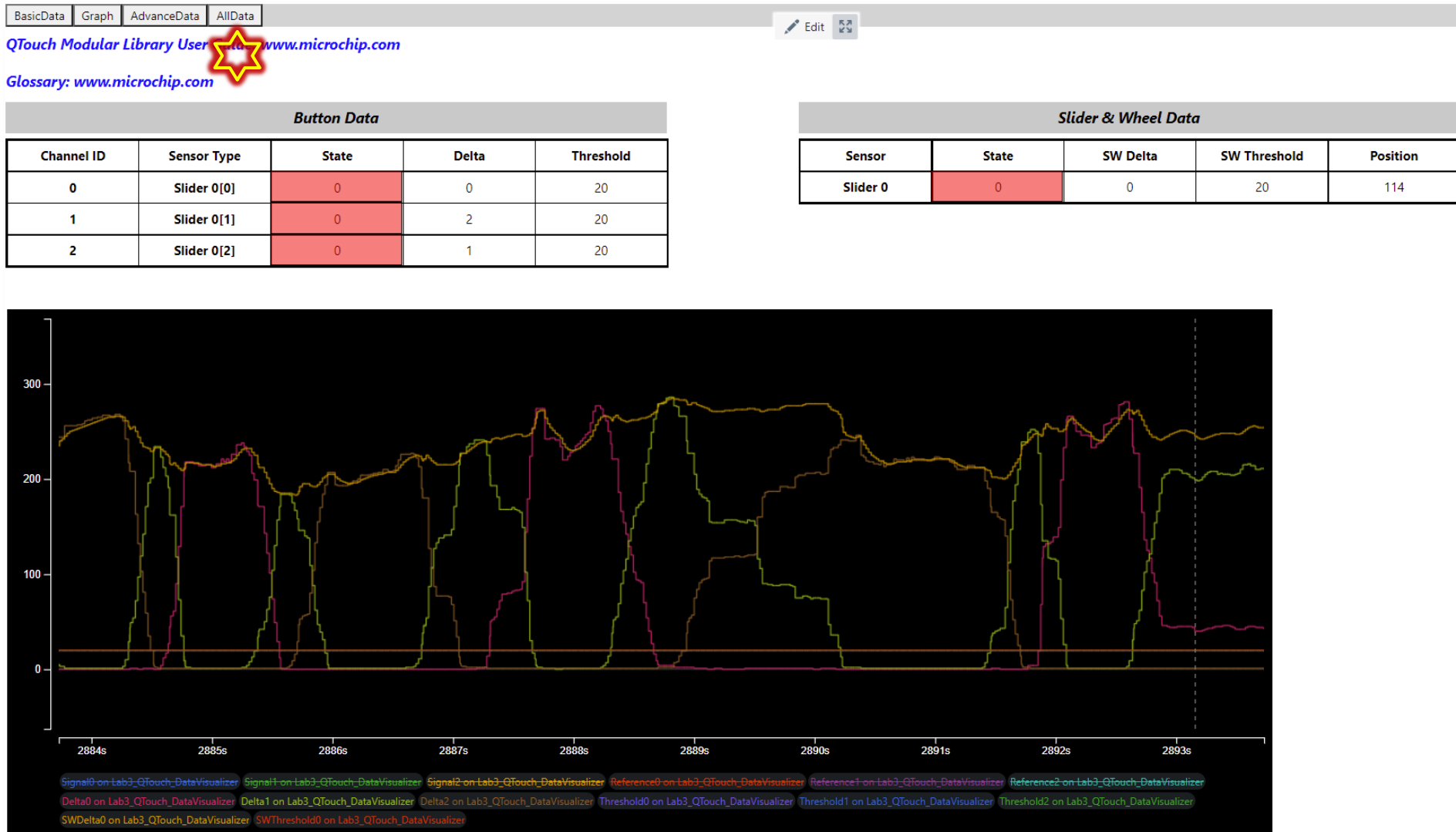
BasicData	Graph	AdvanceData	AllData		
Sensor Data					
Channel ID	Sensor Type	Signal	Reference	Delta	Compensation pF
0	Slider 0[0]	522	512	10	19.75725
1	Slider 0[1]	699	509	190	20.76975
2	Slider 0[2]	520	511	9	23.06475

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

⚙️ Lab1-3 Data Visualizer for Touch

- All Data window of Touch Dashboard.



✖ Lab1-3 Data Visualizer for Touch

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

The screenshot shows the MPLAB Data Visualizer interface. The 'Load...' dialog box is open, displaying the 'DV' folder. The file 'Lab3_QTouch_DataVisualizer.dvws' is selected. The background shows the main interface with the 'Dashboard' view selected, displaying various data tables.

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

Slider & Wheel Data

Sensor	State	SW Delta	SW Threshold	Position
Slider 0	0	0	20	114

Sensor Data

Sensor	State
Slider 0	0

Debug Data

Message	Count
Counter on Lab3_QTouch_DataVisualizer	126
Error on Lab3_QTouch_DataVisualizer	0

Load... Dialog Box

查詢(I): DV

檔案名稱(N): Lab3_QTouch_DataVisualizer.dvws

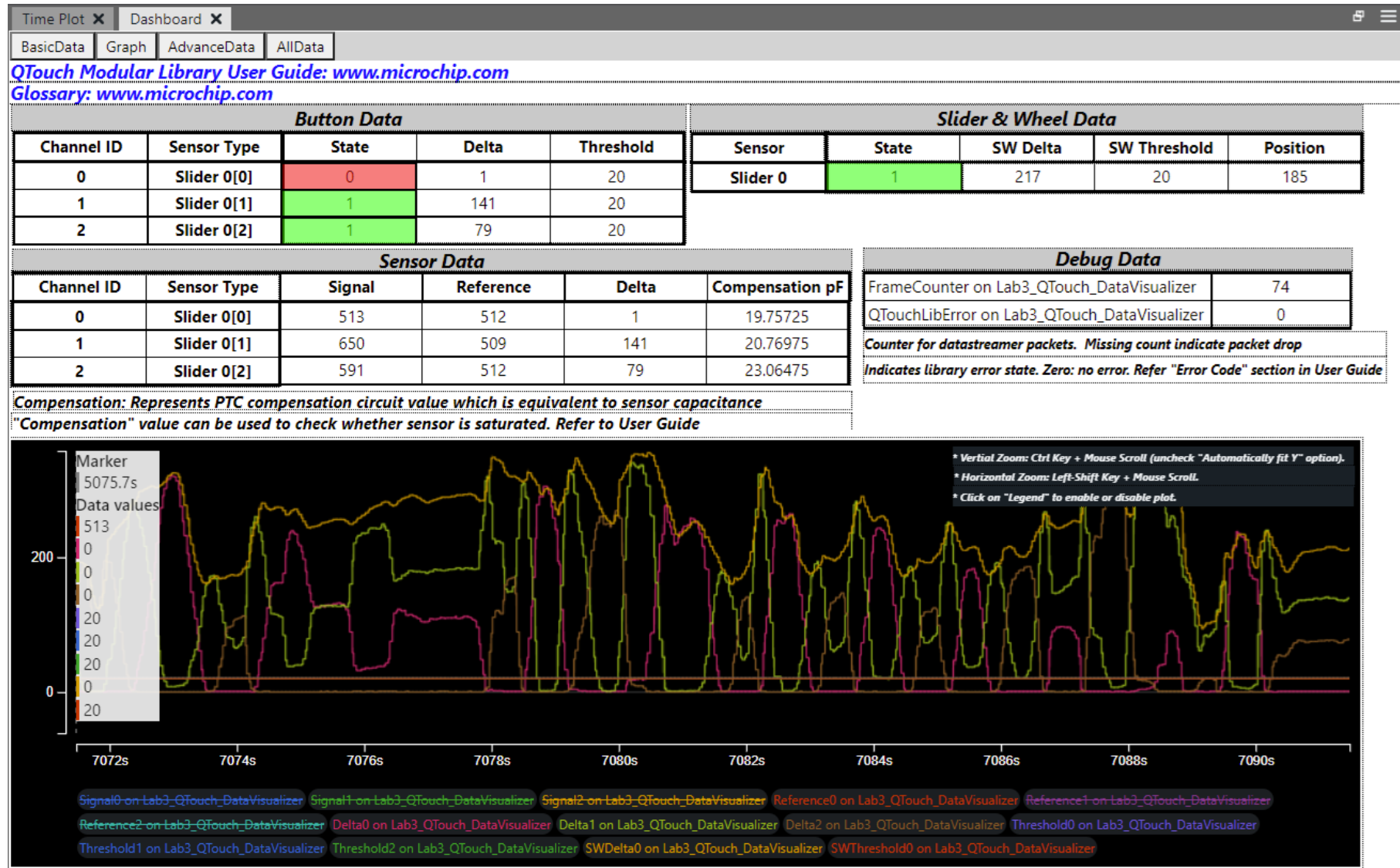
檔案類型(T): Workspace (*.dvws *.json) Files

開啓 (Open)

取消 (Cancel)

✖ Lab1-3 Data Visualizer for Touch

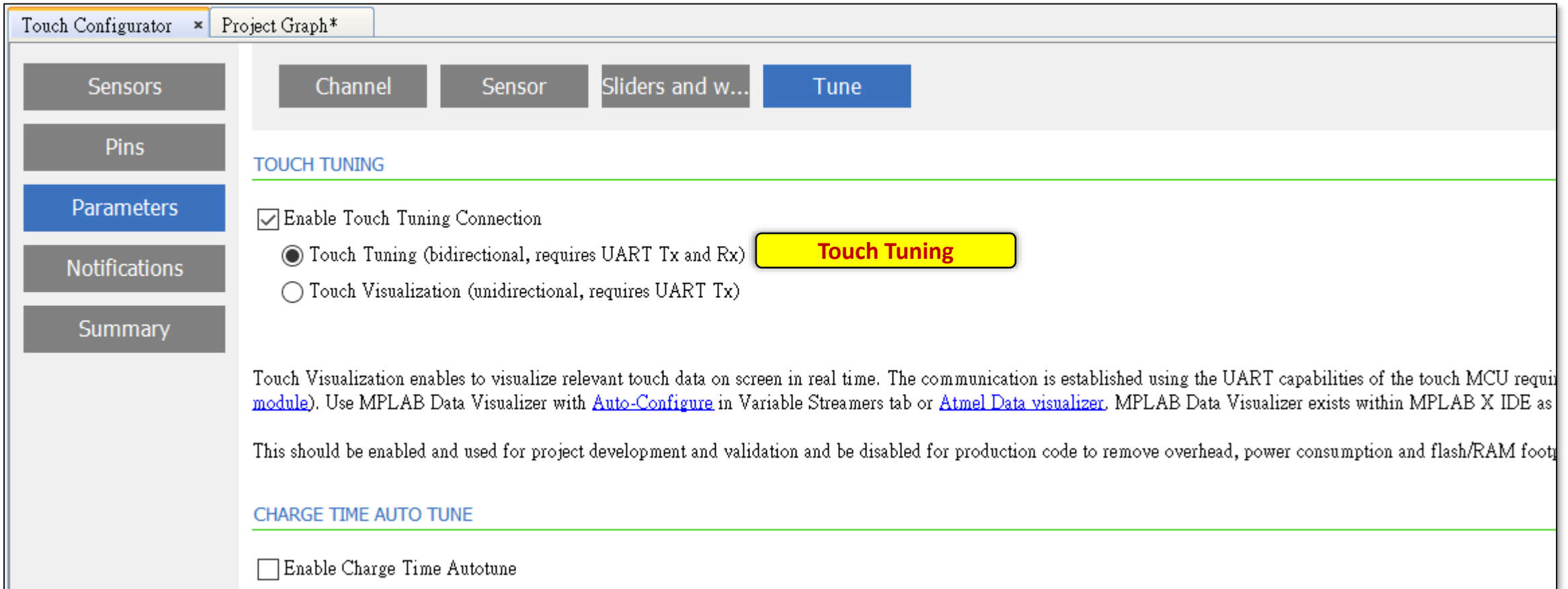
- The customized **All Data** of Dashboard



Lab1-3 Data Visualizer for Touch

Discuss

- How about if select **Touch Tuning** in **Parameters** Tab of Touch Configurator ?



The screenshot shows the Touch Configurator application window. The left sidebar contains tabs: Sensors, Pins, Parameters (selected), Notifications, and Summary. The main area has a top bar with 'Channel', 'Sensor', 'Sliders and w...', and 'Tune' buttons. Below this is the 'TOUCH TUNING' section, which includes a checked checkbox for 'Enable Touch Tuning Connection'. Under this checkbox are two radio button options: 'Touch Tuning (bidirectional, requires UART Tx and Rx)' and 'Touch Visualization (unidirectional, requires UART Tx)'. A yellow button labeled 'Touch Tuning' is positioned to the right of the first radio button. Below these options is a paragraph of text explaining the functionality of Touch Visualization. At the bottom of the window is the 'CHARGE TIME AUTO TUNE' section with an unchecked checkbox for 'Enable Charge Time Autotune'.

Touch Configurator * Project Graph*

Sensors Pins Parameters Notifications Summary

Channel Sensor Sliders and w... Tune

TOUCH TUNING

☒ Enable Touch Tuning Connection

☒ Touch Tuning (bidirectional, requires UART Tx and Rx) **Touch Tuning**

☐ Touch Visualization (unidirectional, requires UART Tx)

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 a [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 a separate 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.

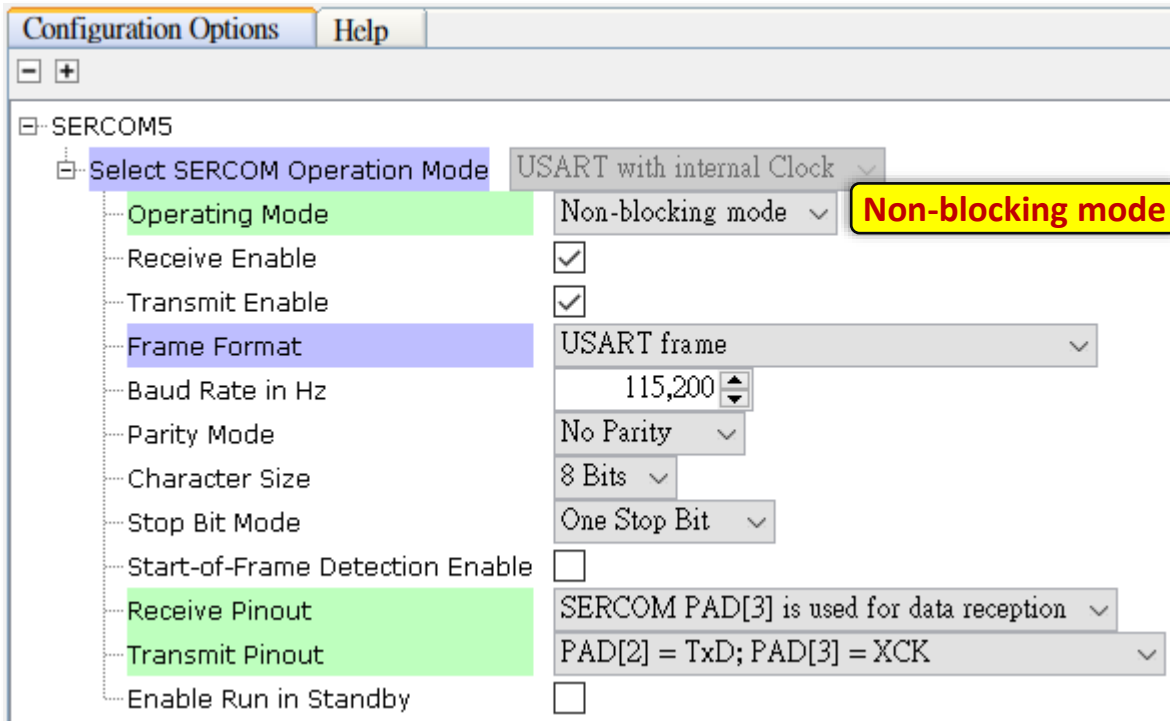
CHARGE TIME AUTO TUNE

☐ Enable Charge Time Autotune

⚙️ Lab1-3 Data Visualizer for Touch

Discuss

- Enable **Touch Tuning** of Touch Library, the SERCOM UART need use **Non-Blocking Mode**.
- Modify **myprintf()** to use **WriteIsBusy()** function of SERCOM **Non-Blocking Mode**.
- Yes ! It's to modify back to the original configuration and source code from Lab 1-2.



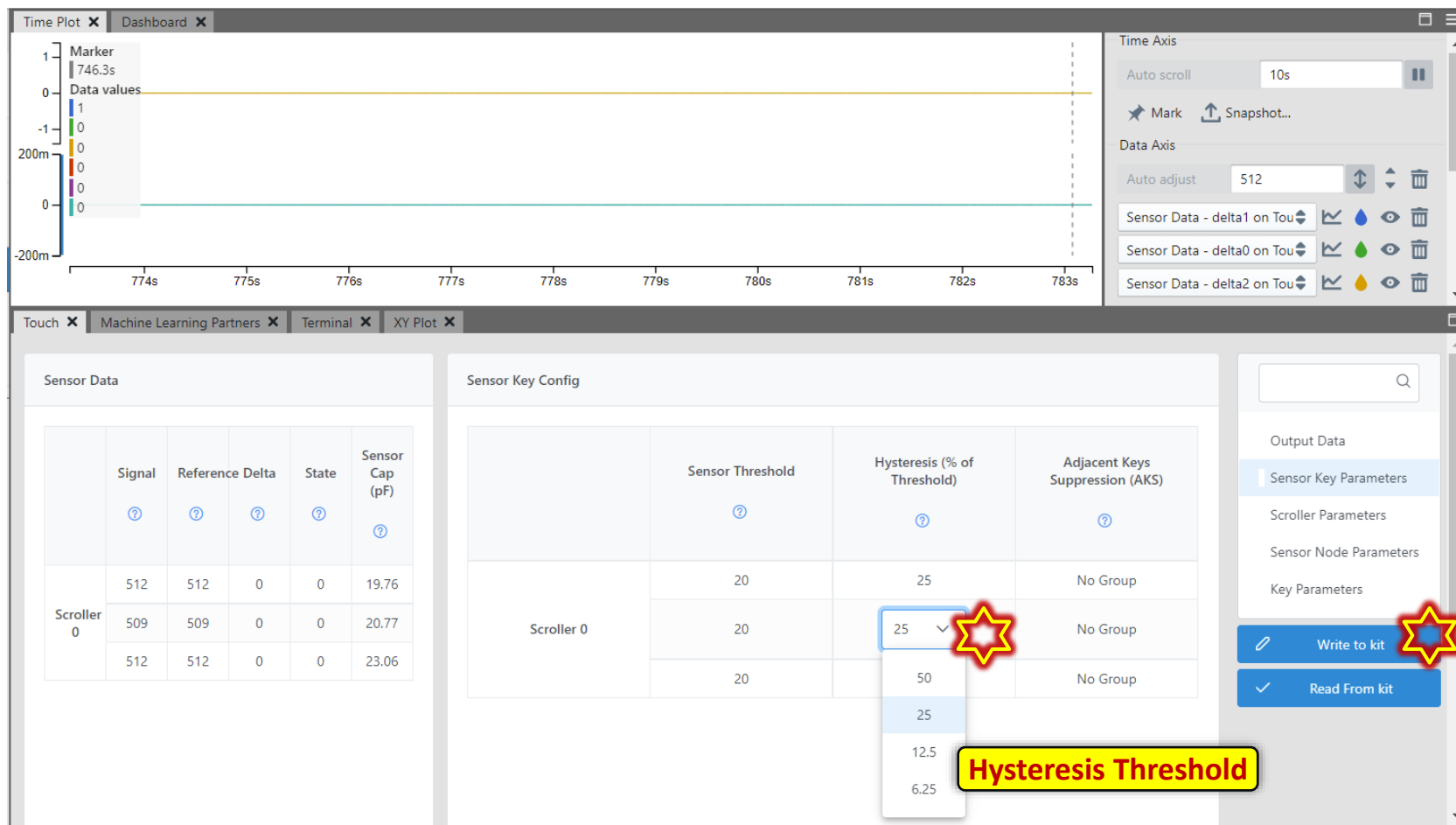
```
...  
void myprintf(const char *format, ...)  
{  
    ...  
    if ((len > 0) && (len < SYS_CONSOLE_PRINT_BUFFER_SIZE))  
    {  
        consolePrintBuffer[len] = '\0';  
        SERCOM5_USART_Write((uint8_t*)consolePrintBuffer, len);  
        while (SERCOM5_USART_WriteIsBusy());  
        //while( !SERCOM5_USART_TransmitComplete() );  
    }  
}
```

Non-blocking mode function

⚙️ Lab1-3 Data Visualizer for Touch

Discuss

- The **Touch Tuning** accept configure parameters in **Data Visualizer** and write back to Touch Library through UART.



USB Fundamentals

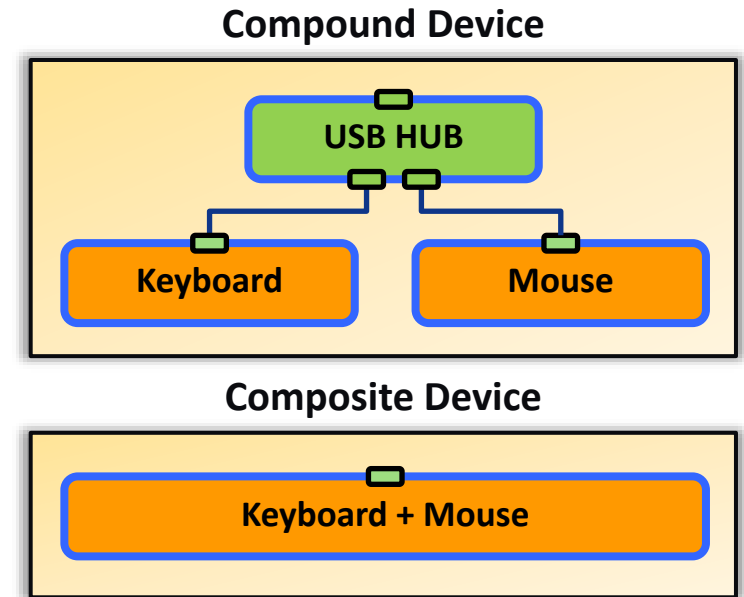
USB, Universal Serial Bus

- 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

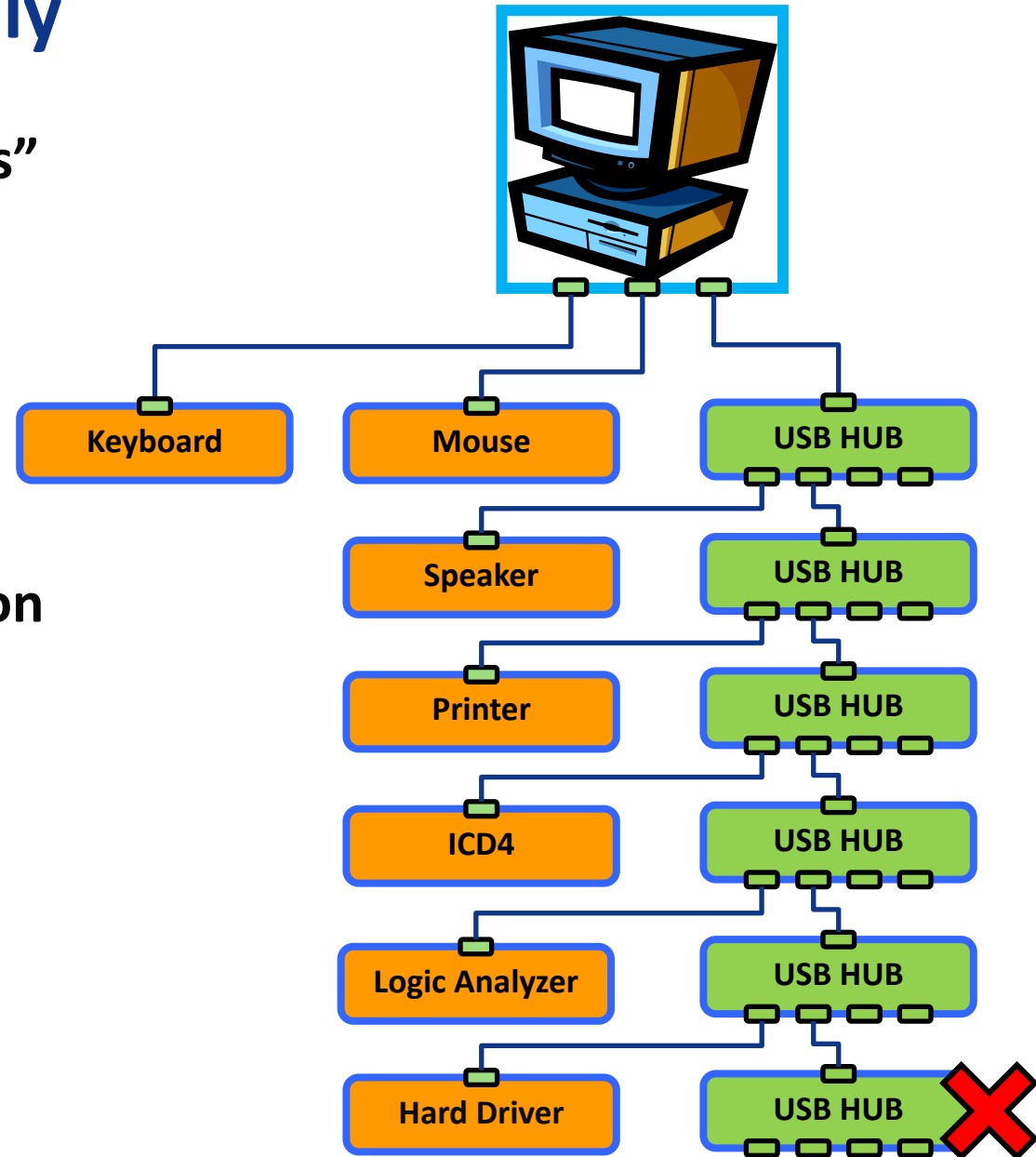
USB Device Types

- **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)



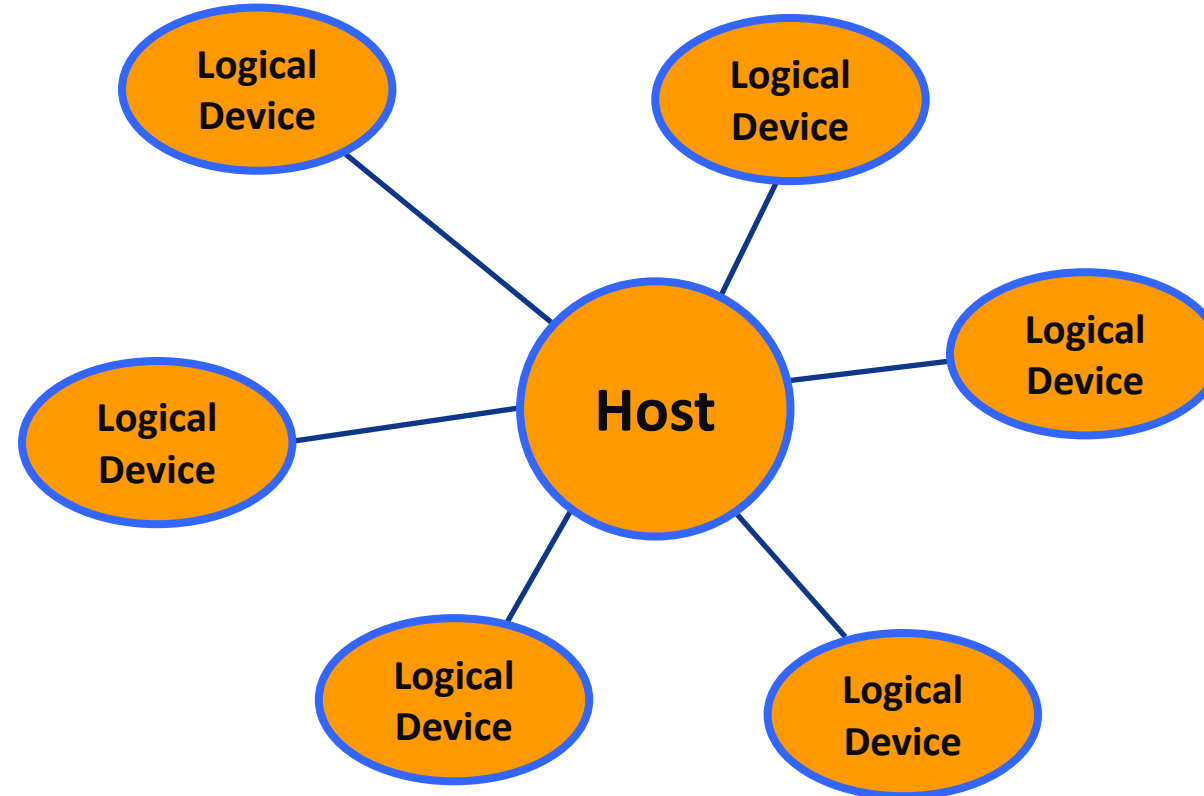
Tiered Star Topology, Physically

- 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.



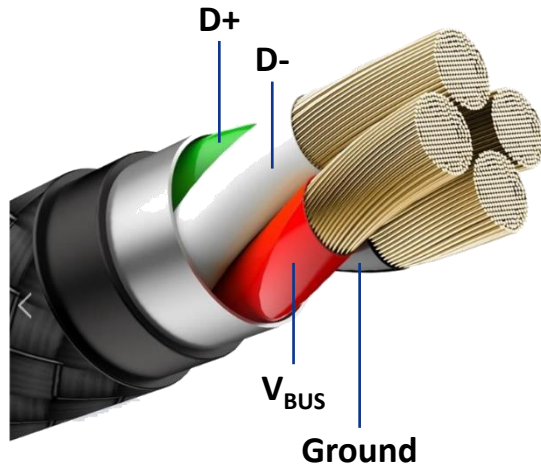
Star Topology, Logical

- 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).



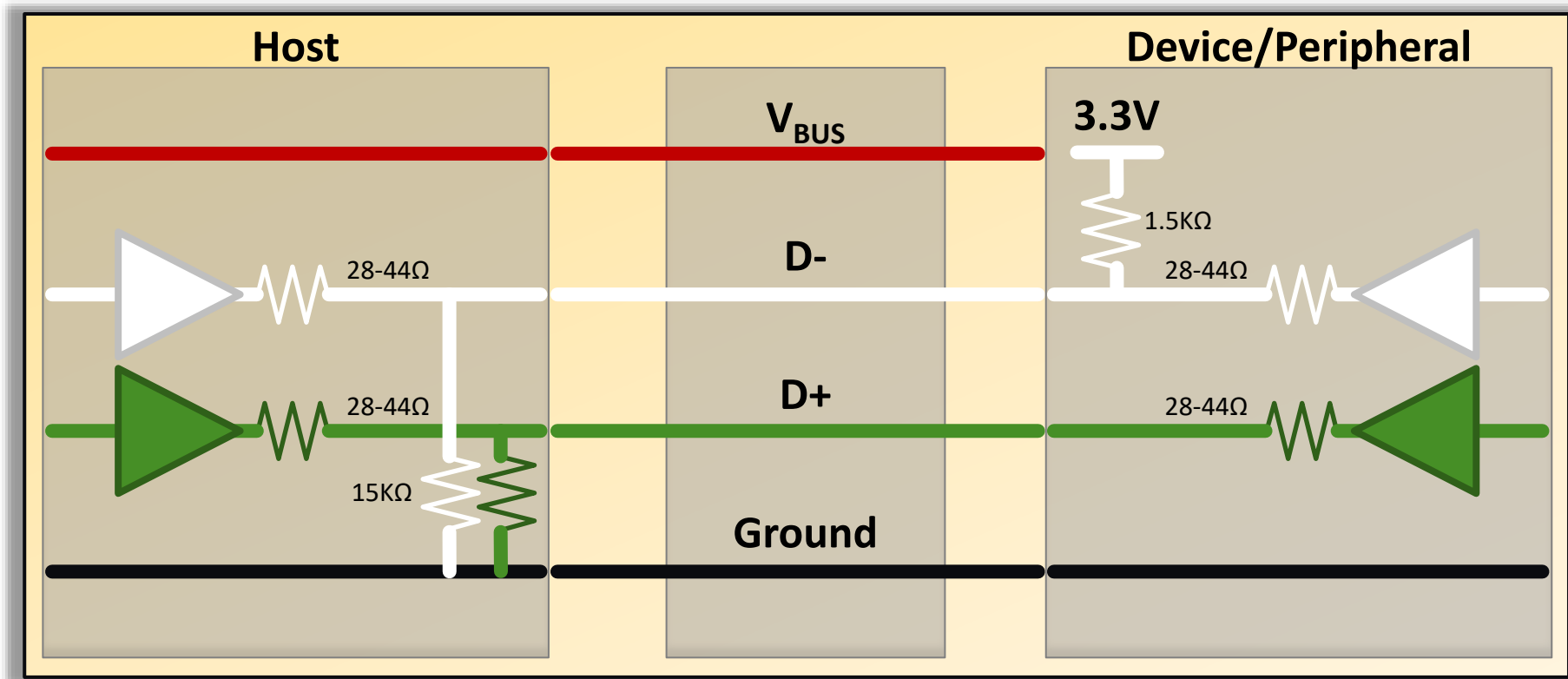
VBUS	~ 5.0V
D+/D-	~ 3.3V

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

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

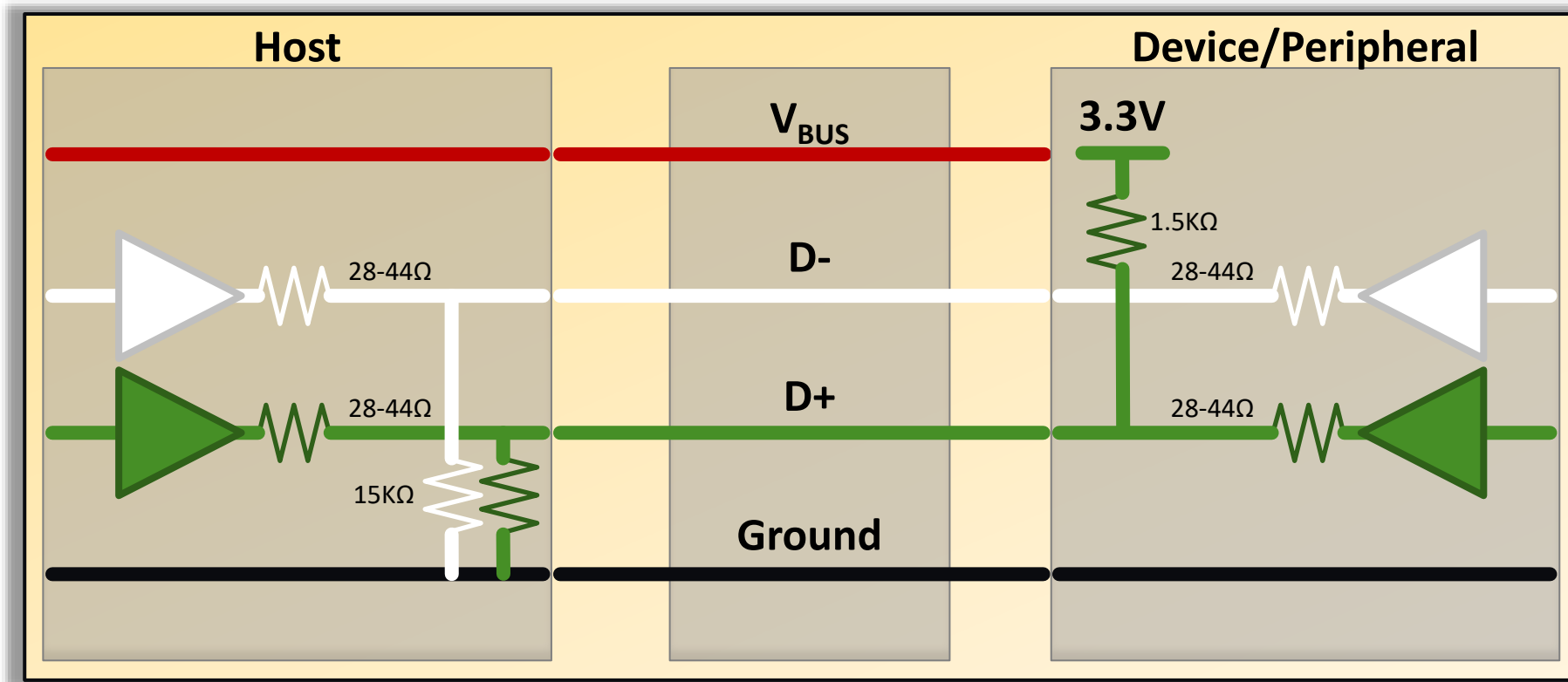
Attached : Low-Speed

- 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.



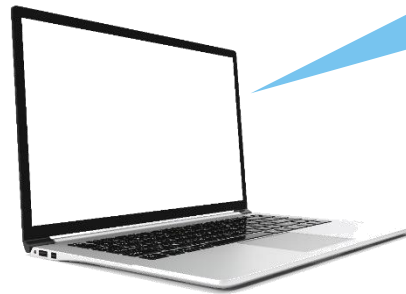
Attached : Full/High-Speed

- 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”.



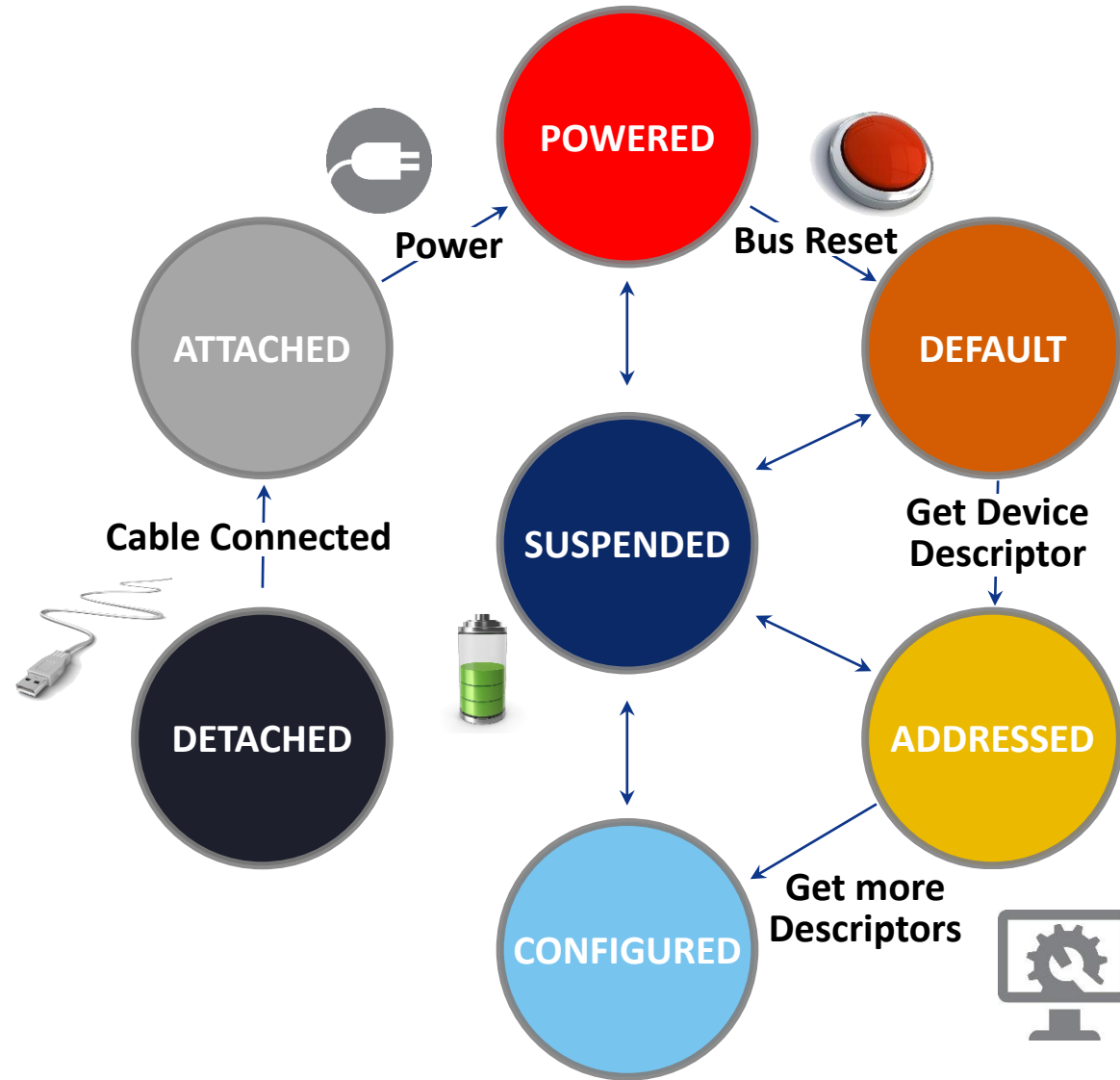
Hi, who are you?

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 ?

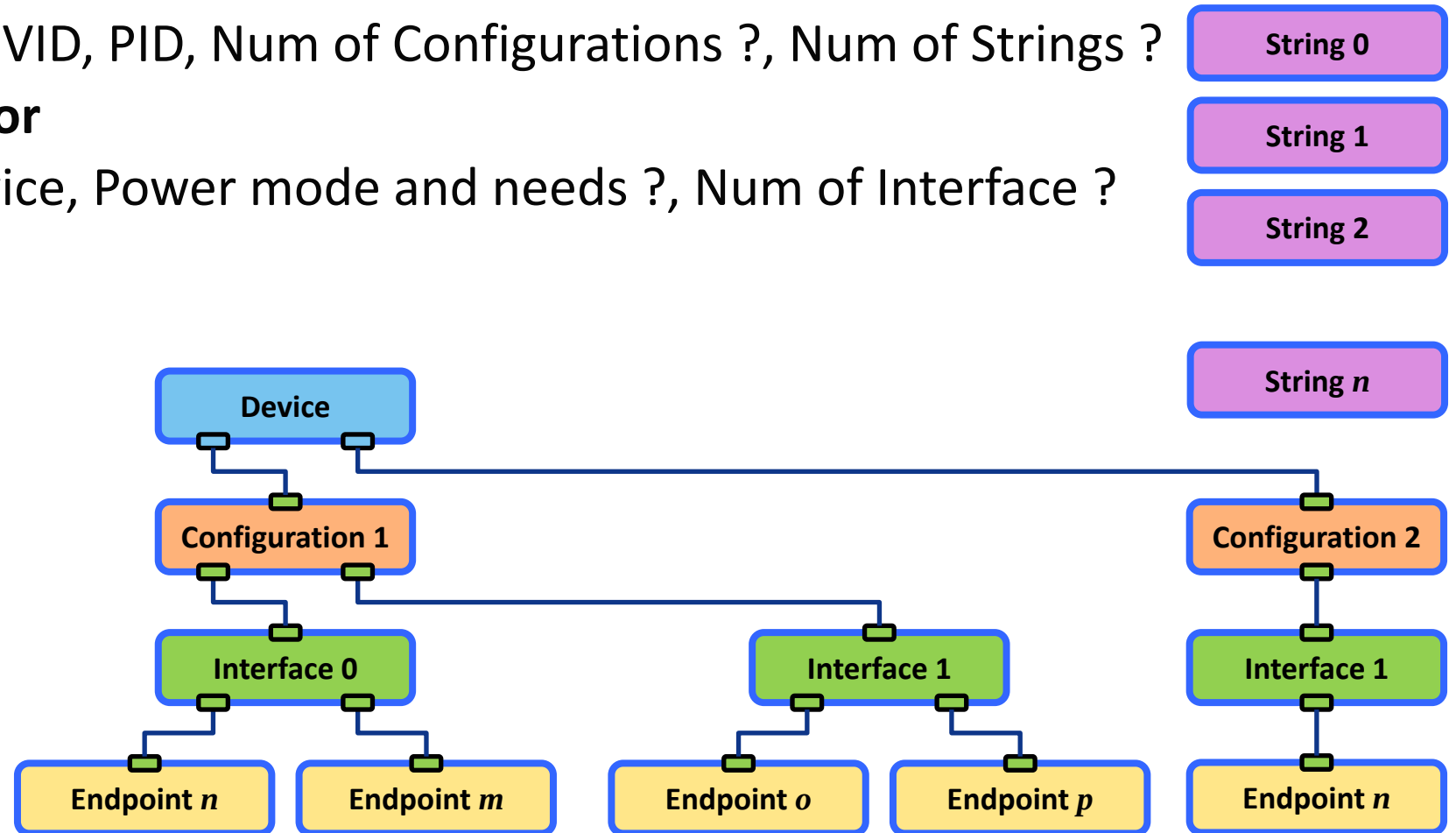
- **Endpoint Descriptor**

- Transfer Type

- **String Descriptor**

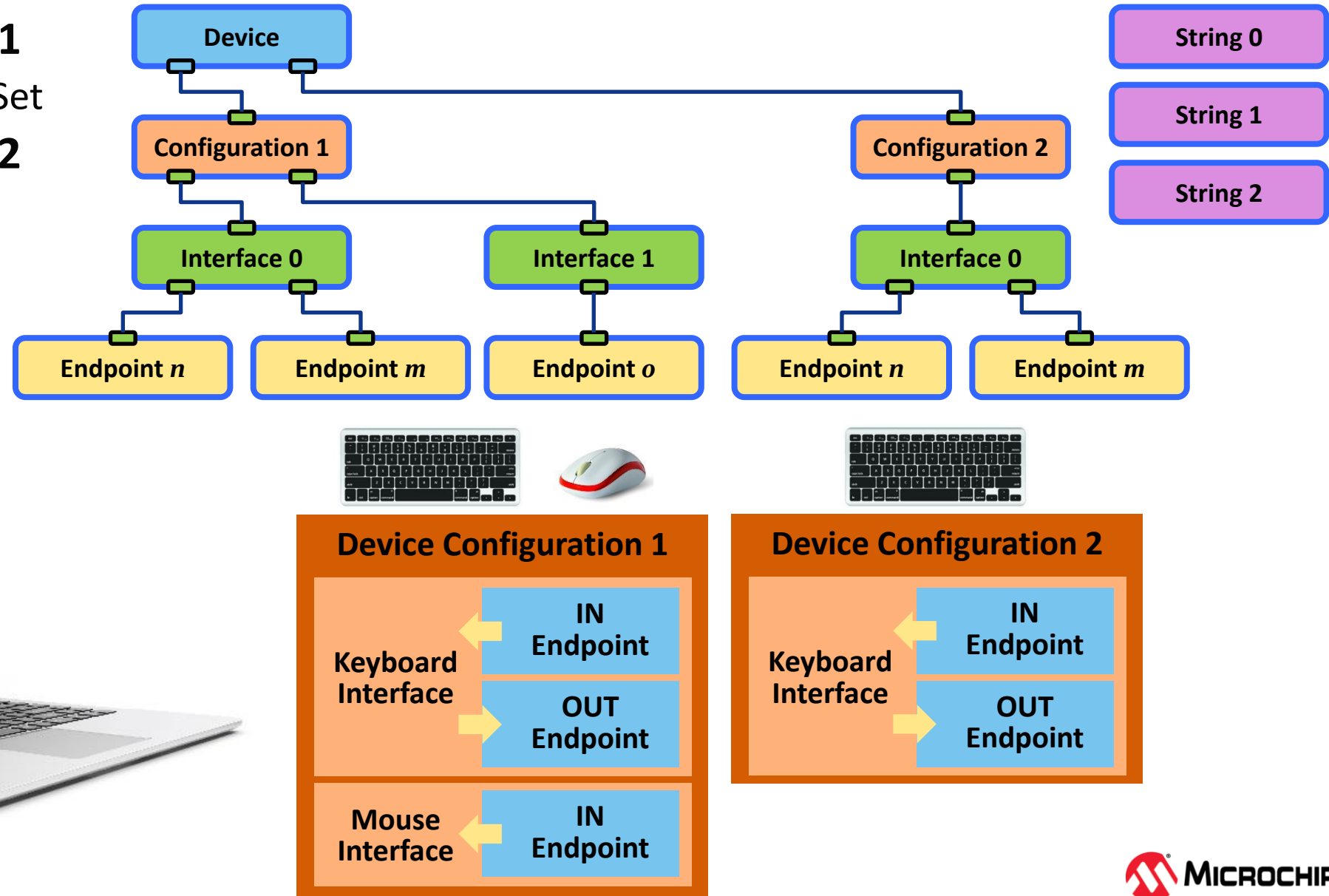
- **Driver Class Specific**

- **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**

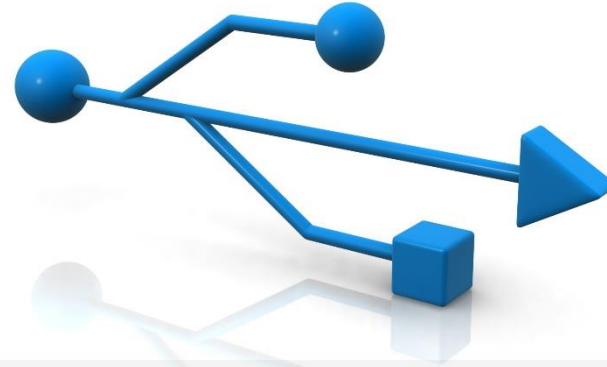
- Periodic Transfer, Guaranteed latency.
- Delivery not Guaranteed.
- Moderate/high throughput.

- **Bulk Transfer**

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

- **Control Transfer**

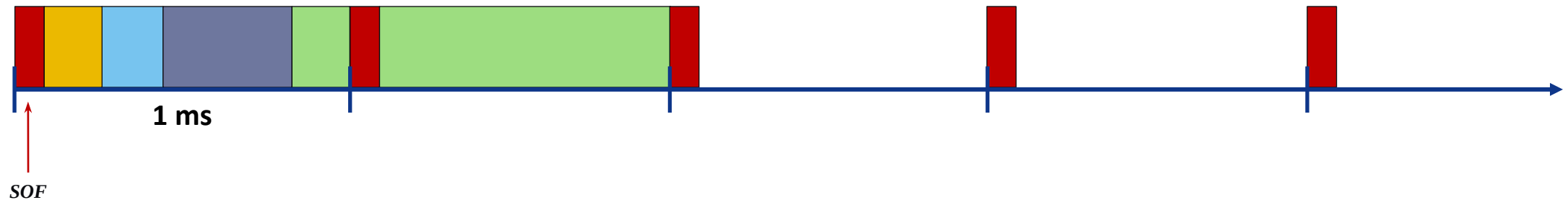
- Used to configure the device.
- All devices use control transfers.
- Guaranteed delivery and Low throughput.



<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

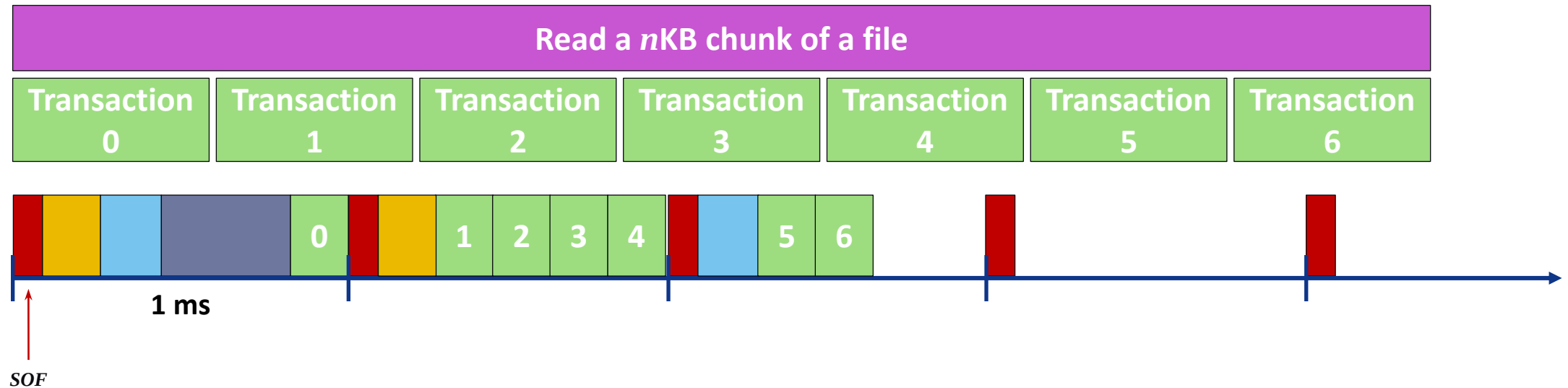
Data Transfer Scheduling

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

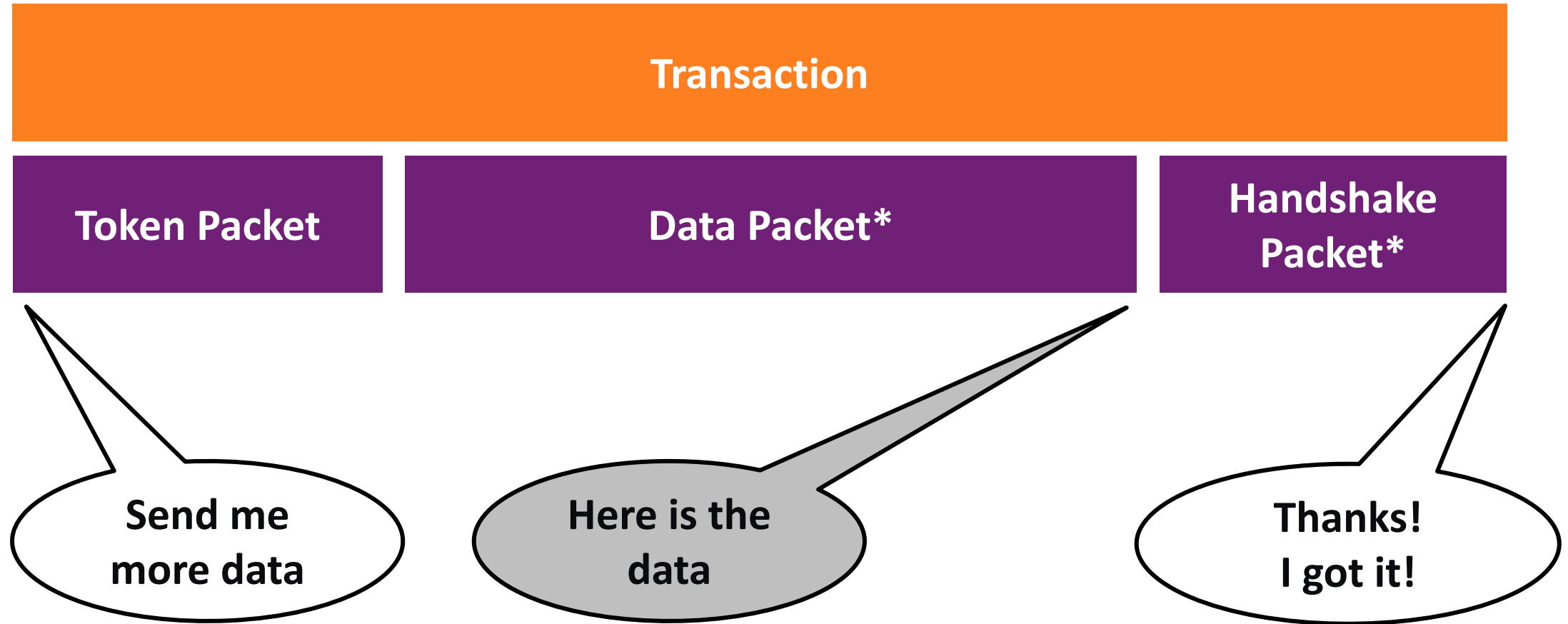


USB Transactions

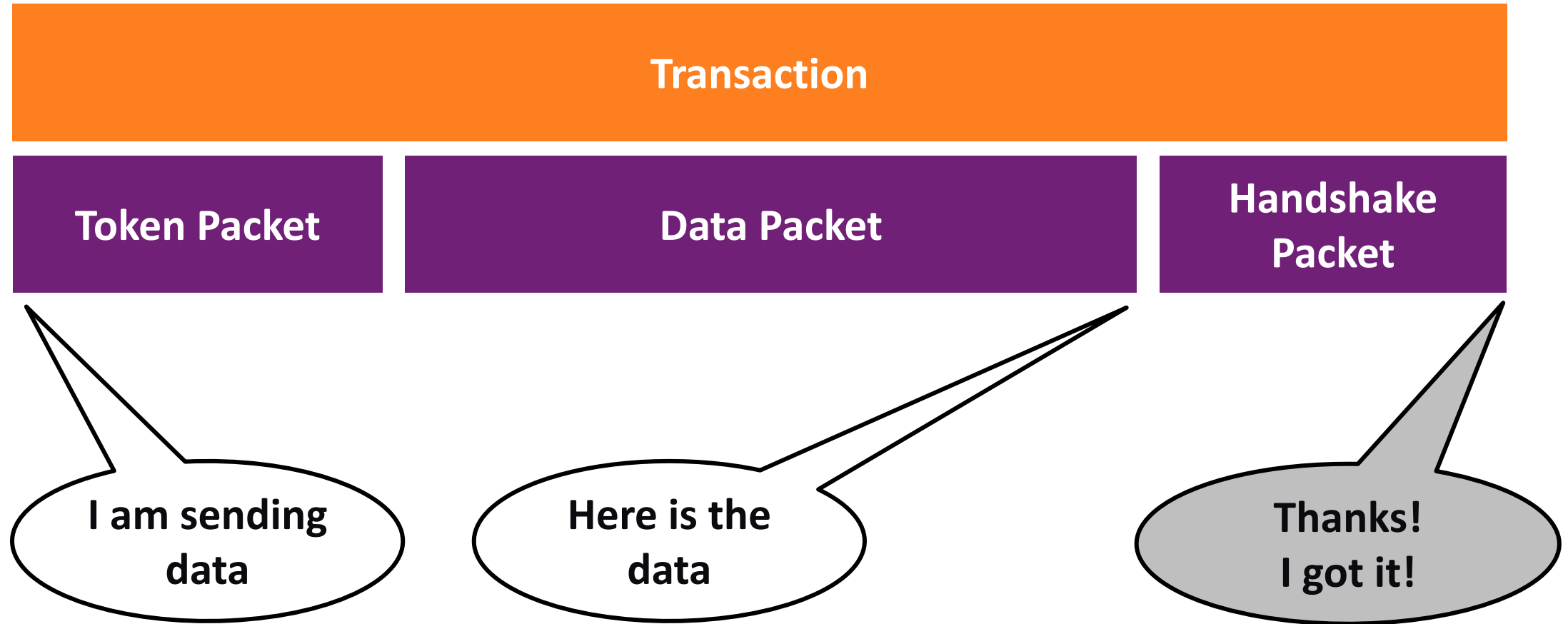
- Specification defined maximum packet size for endpoint.



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)

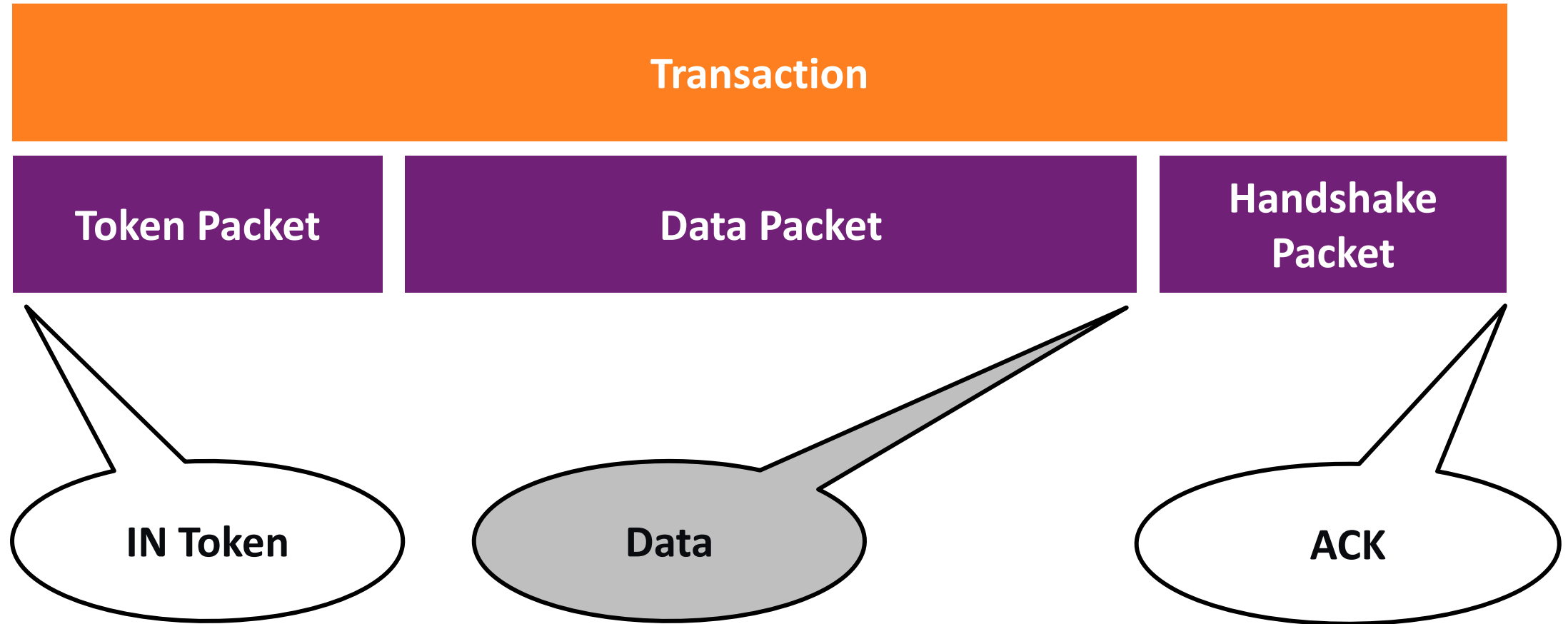
Data Packet

**Handshake
Packet**

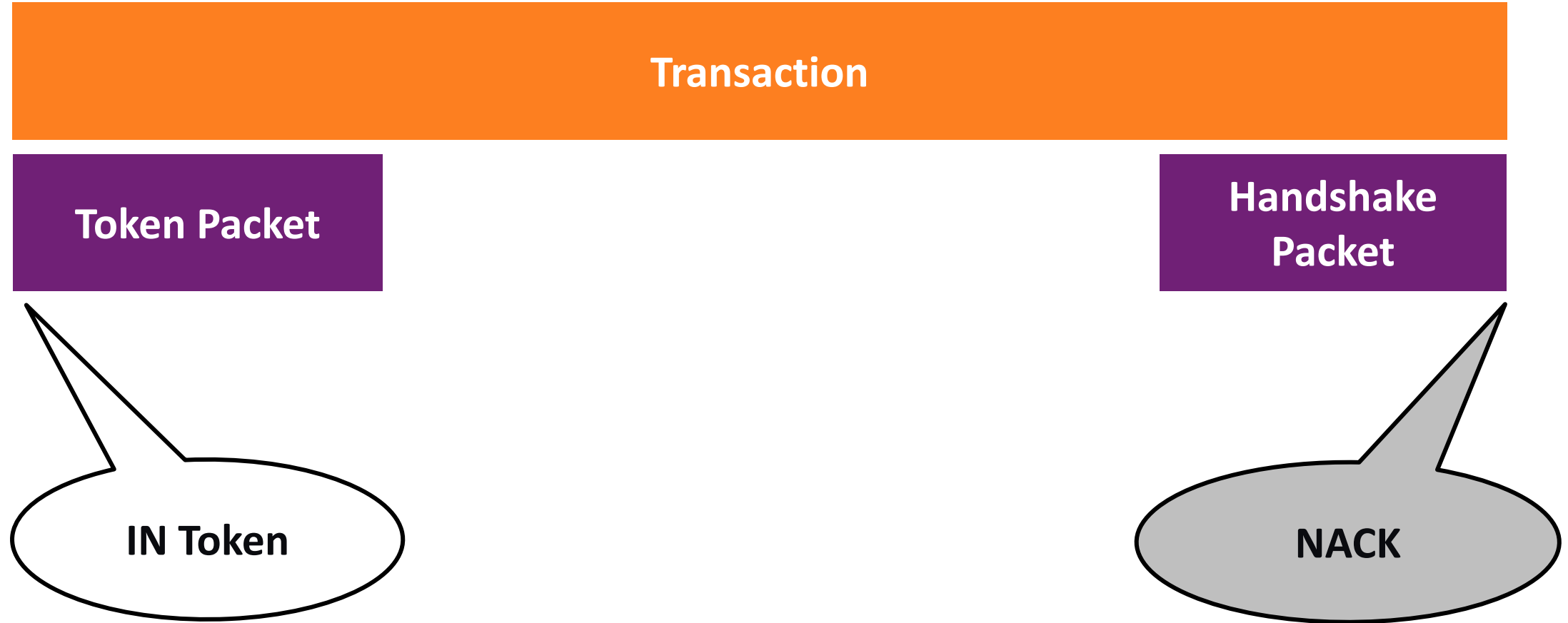
ACK
NACK
STALL
NYET
ERR

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

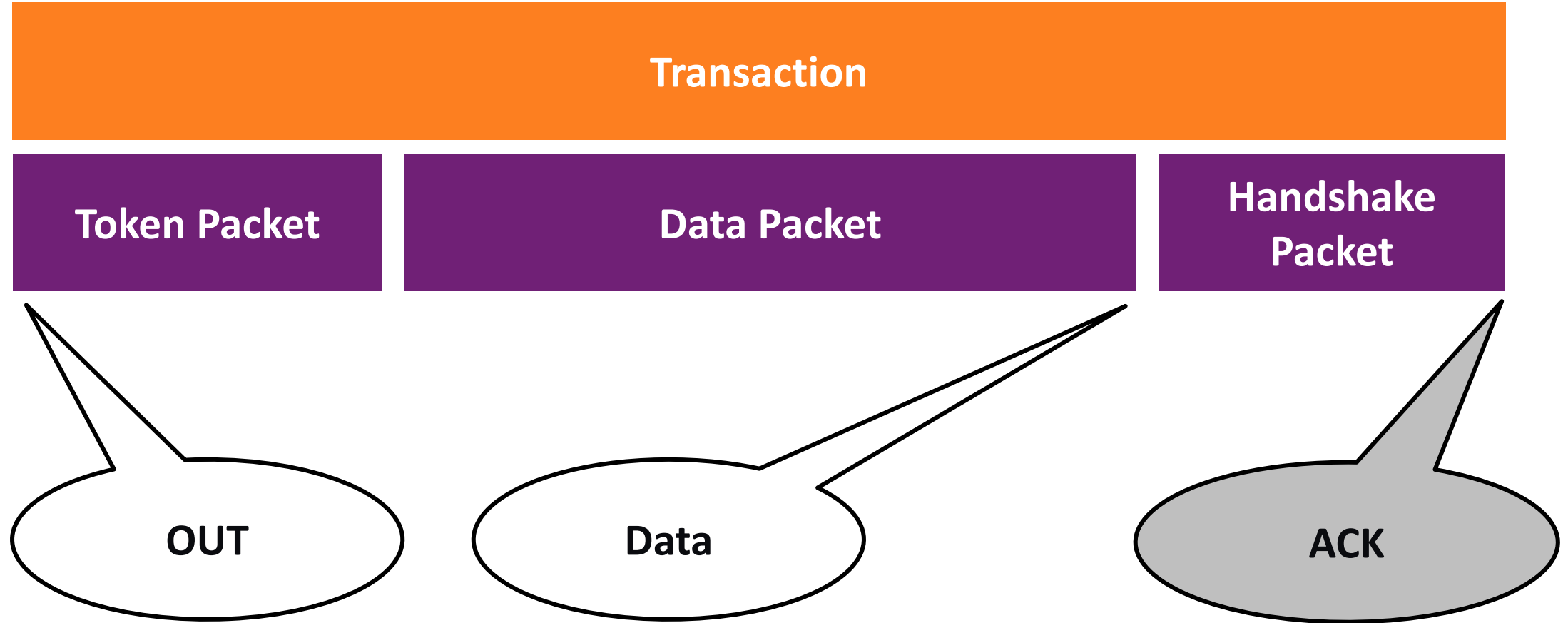
USB Transactions - Require Data, IN - ACK



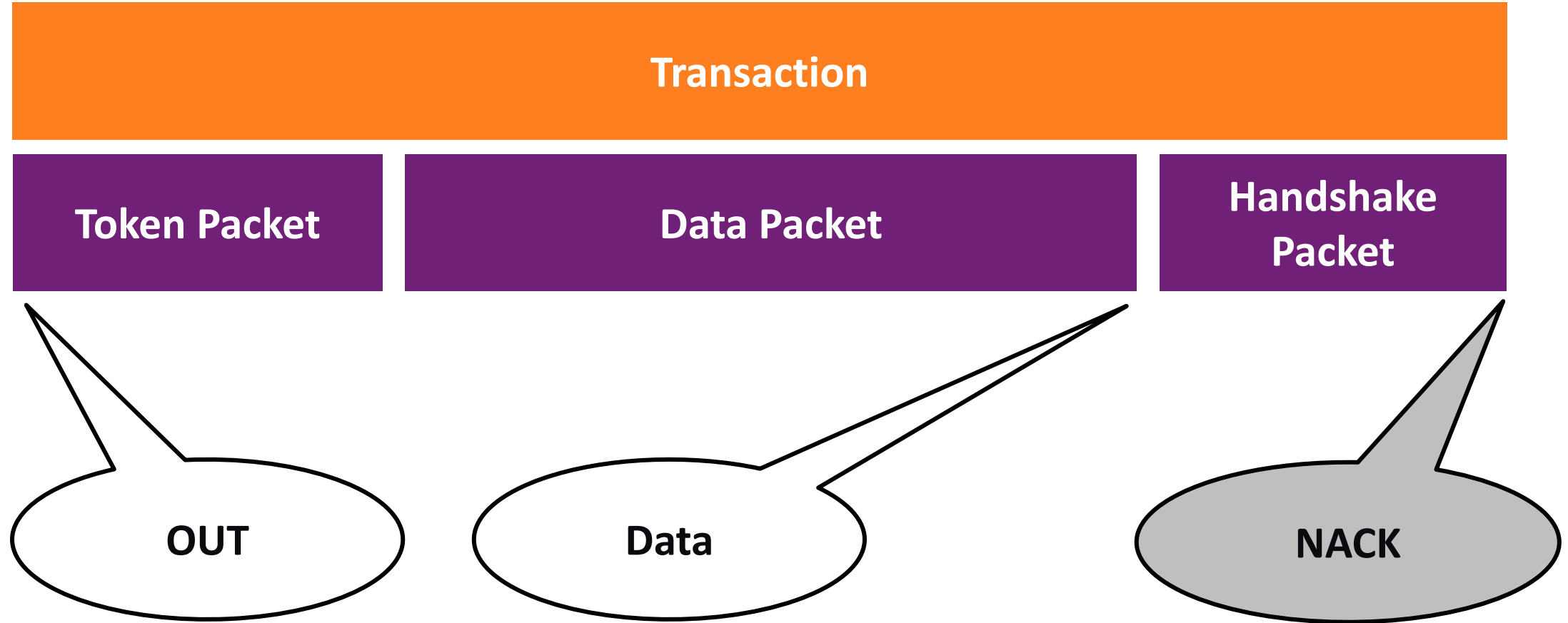
USB Transactions - Require Data, IN - NACK



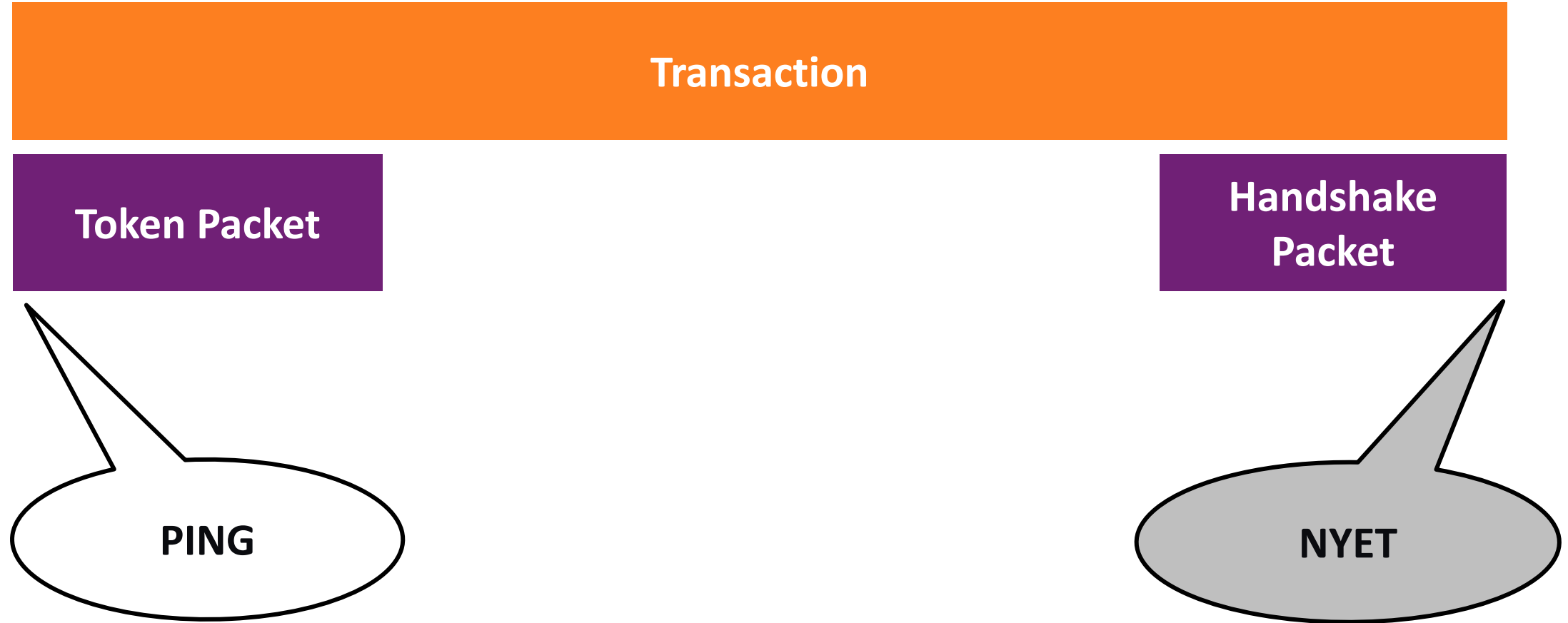
USB Transactions - Send Data, OUT - ACK



USB Transactions - Send Data, OUT - NACK

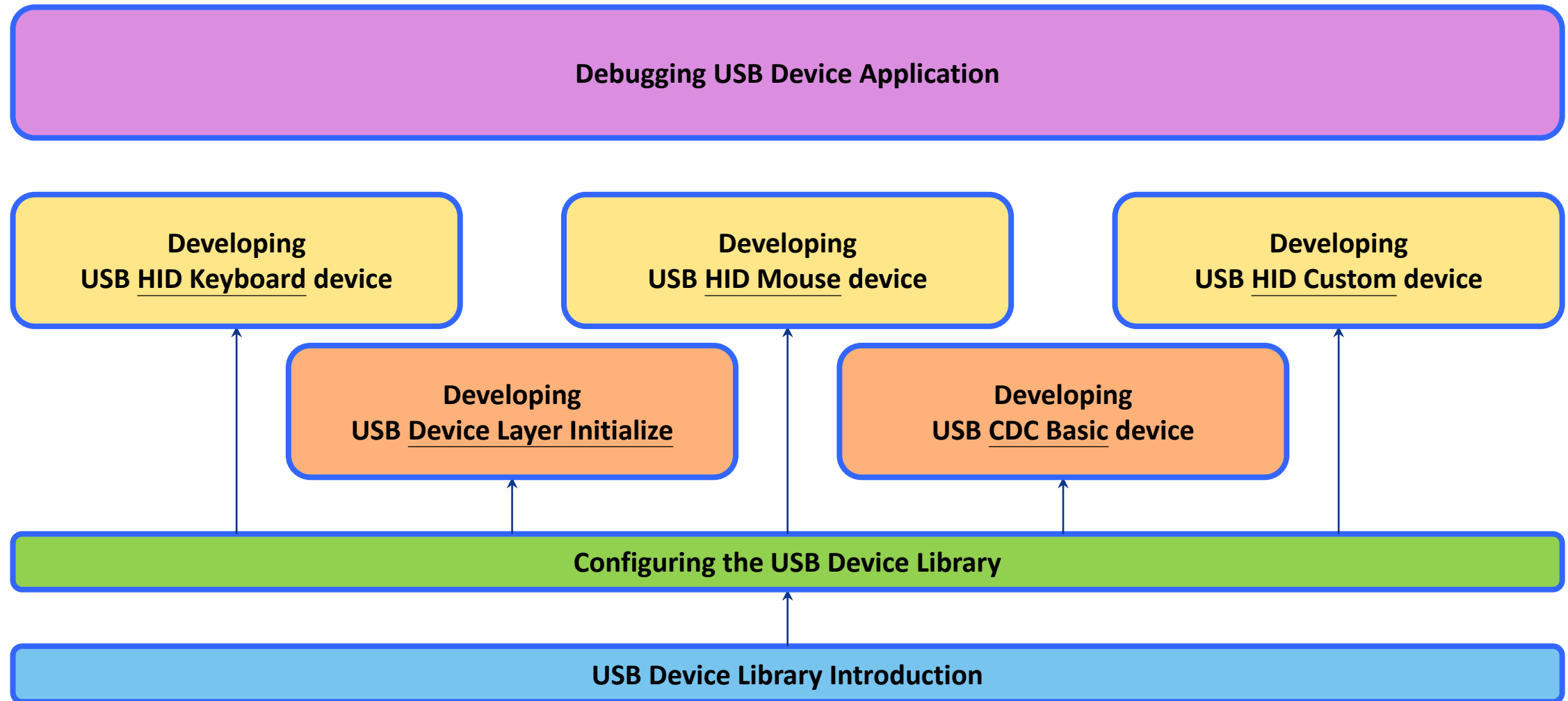


USB Transactions - Send Data, PING - NYET

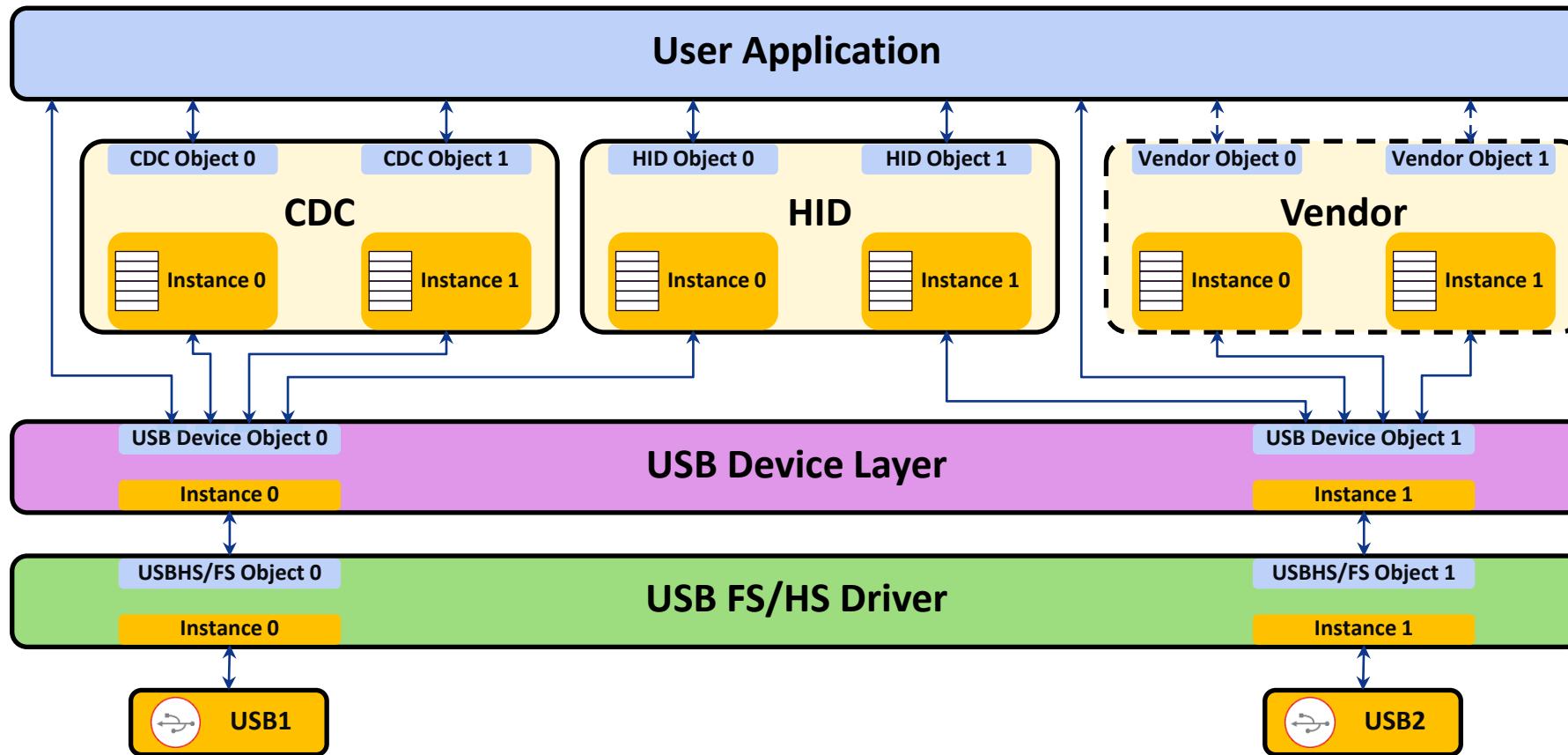


USB Device Library (USB Device Architecture)

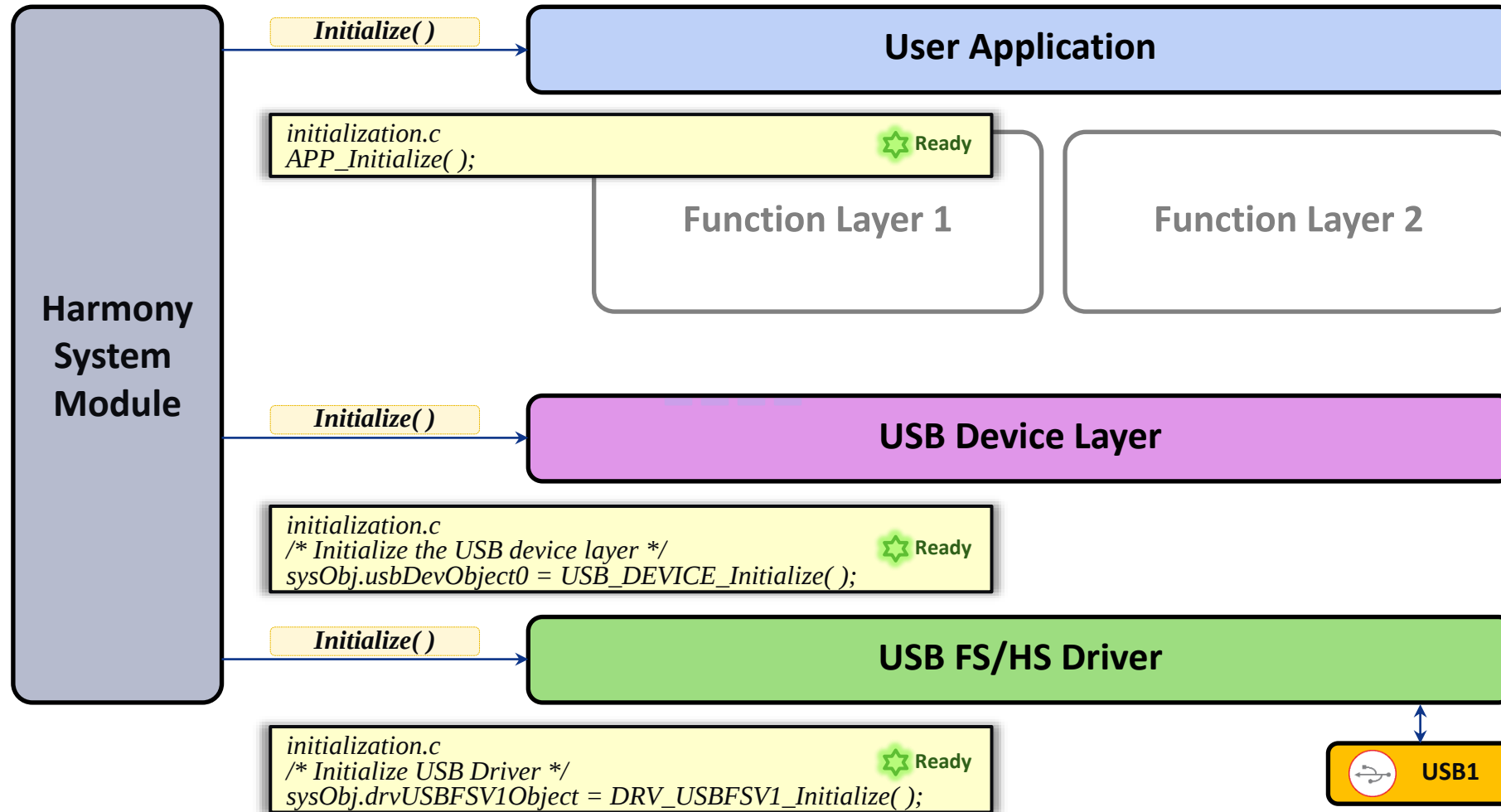
Course Structure for USB Device Library



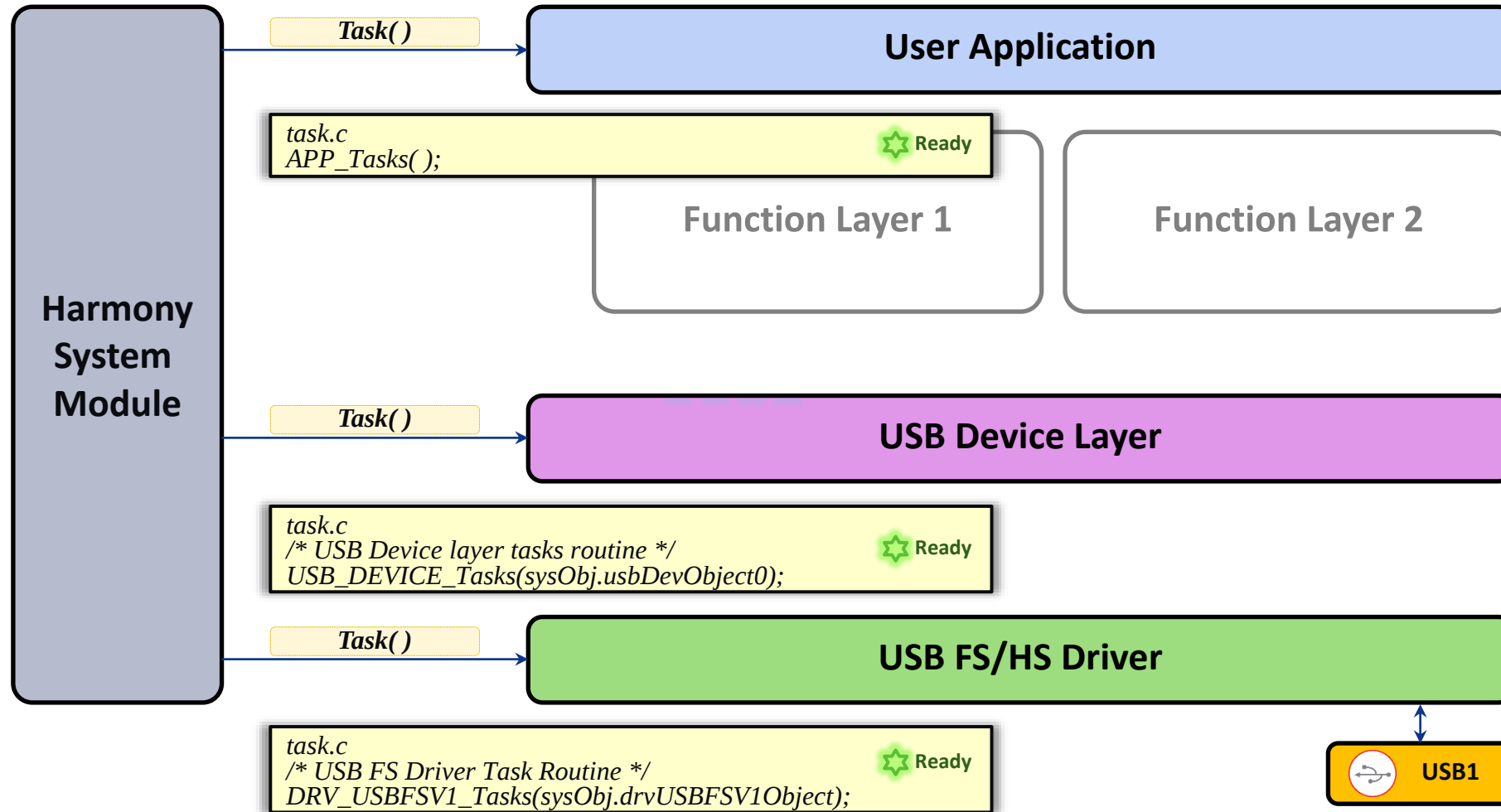
USB Device Library Architecture



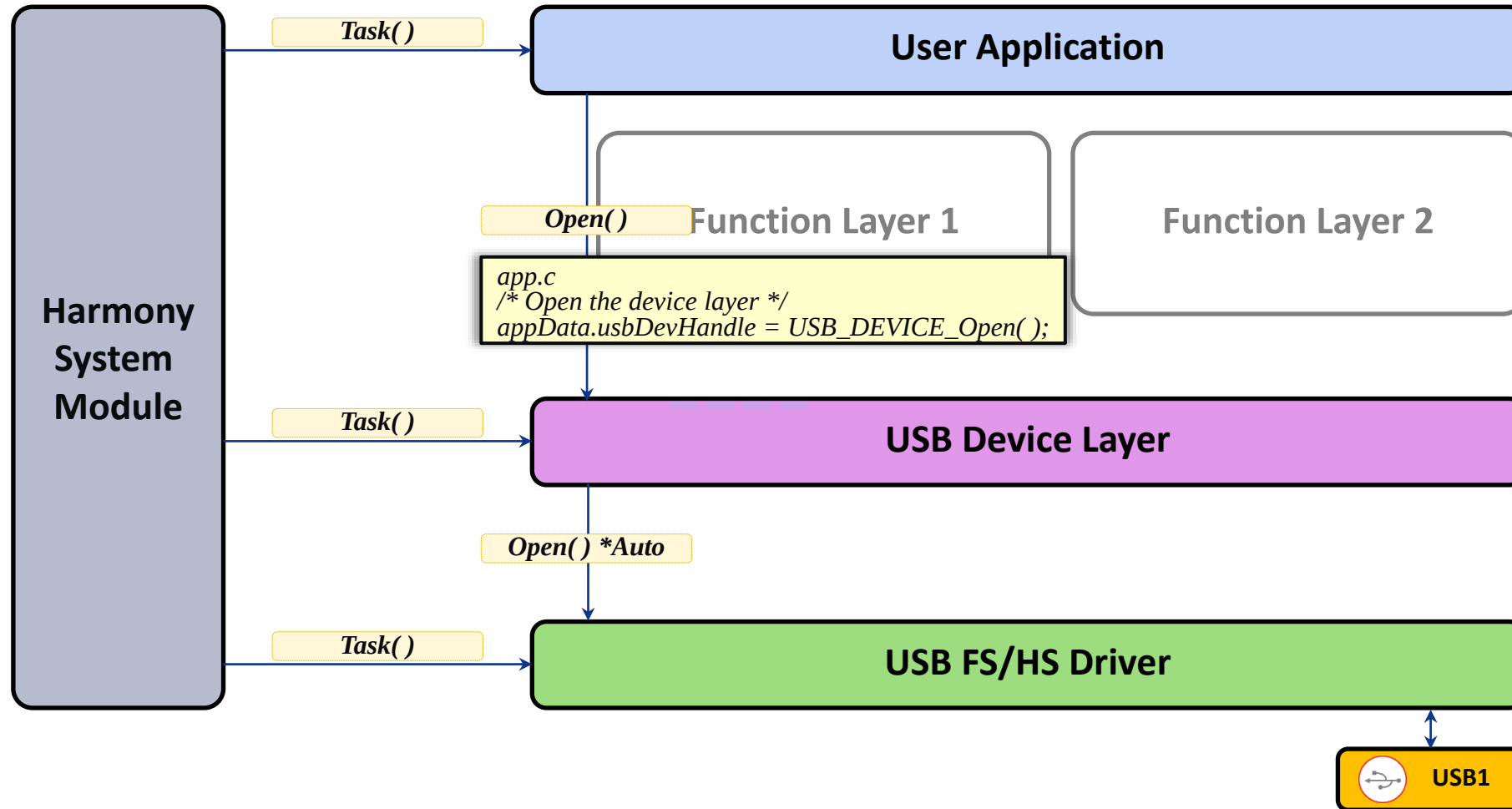
USB Stack Abstraction Model - Initialize & Open



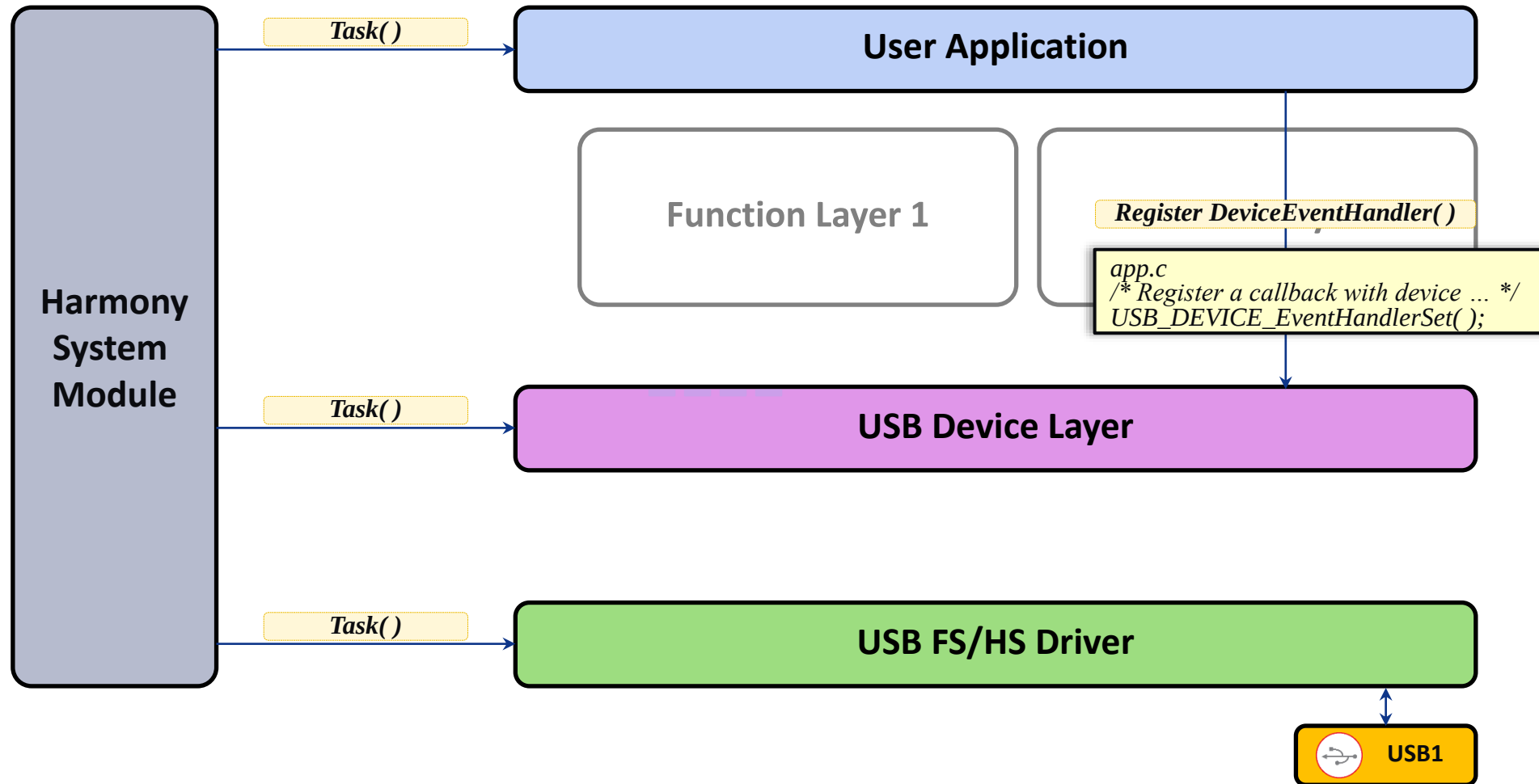
USB Stack Abstraction Model - Initialize & Open



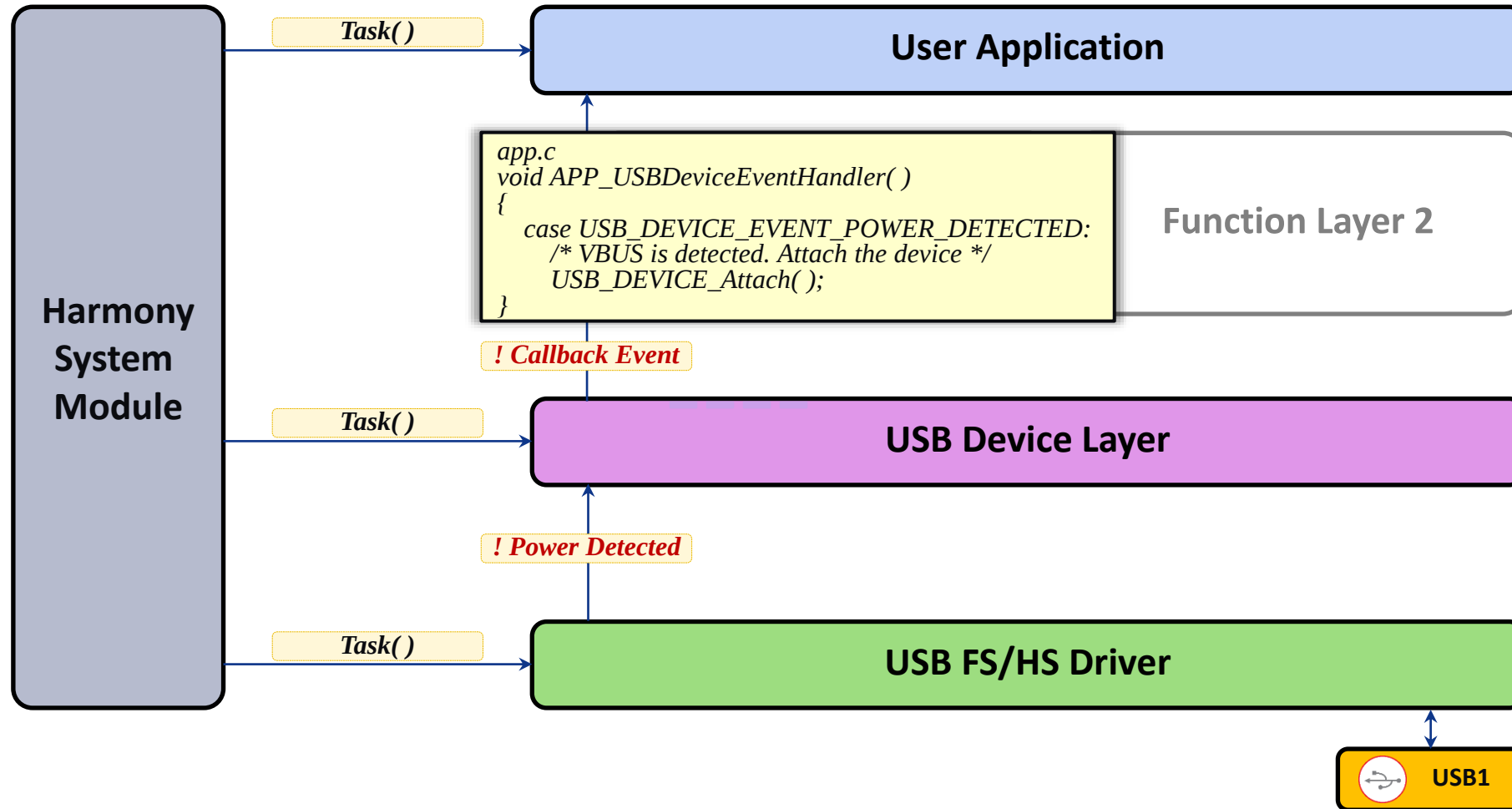
USB Stack Abstraction Model - Initialize & Open



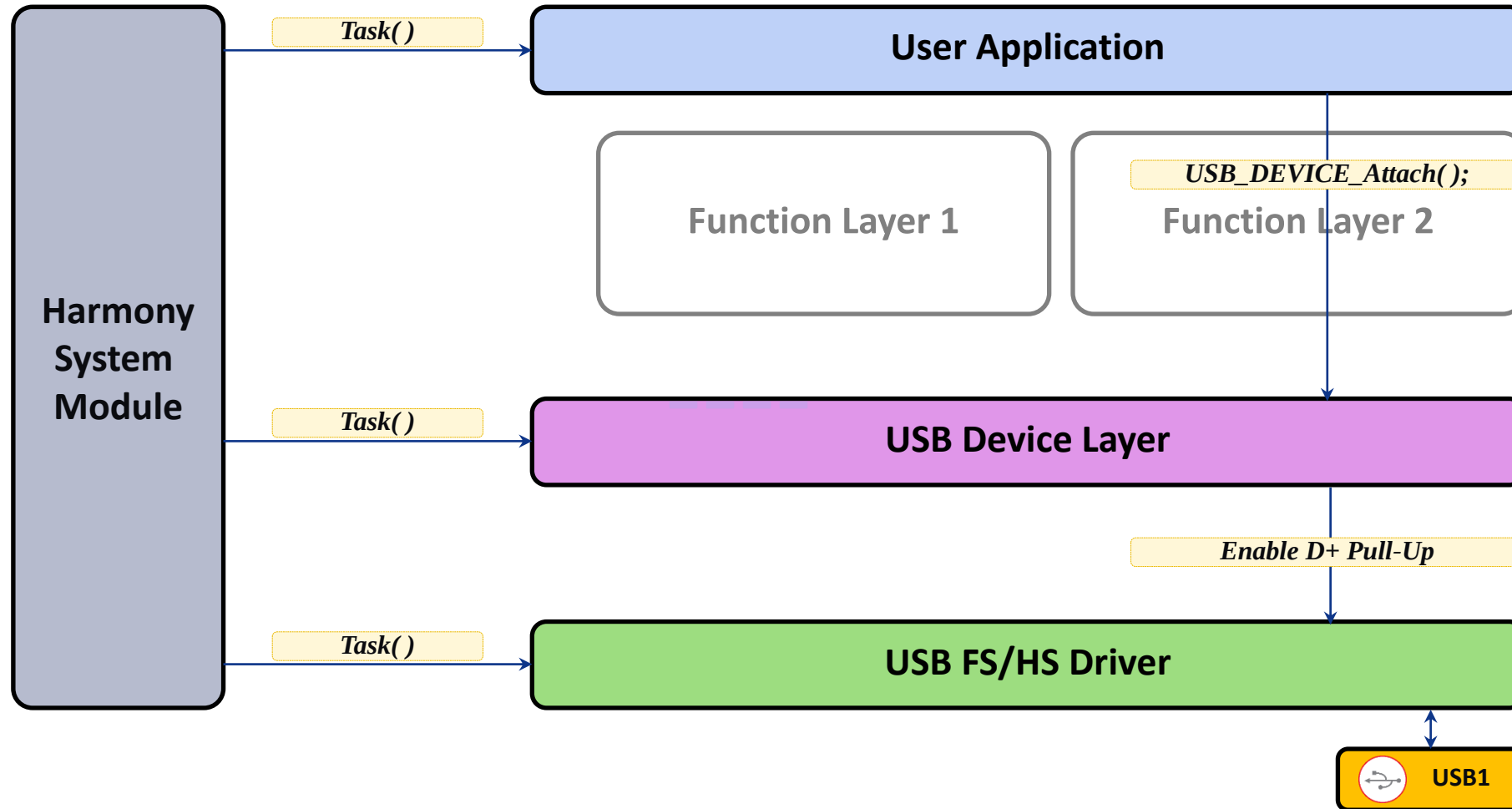
USB Stack Abstraction Model - Register Device Event Handler



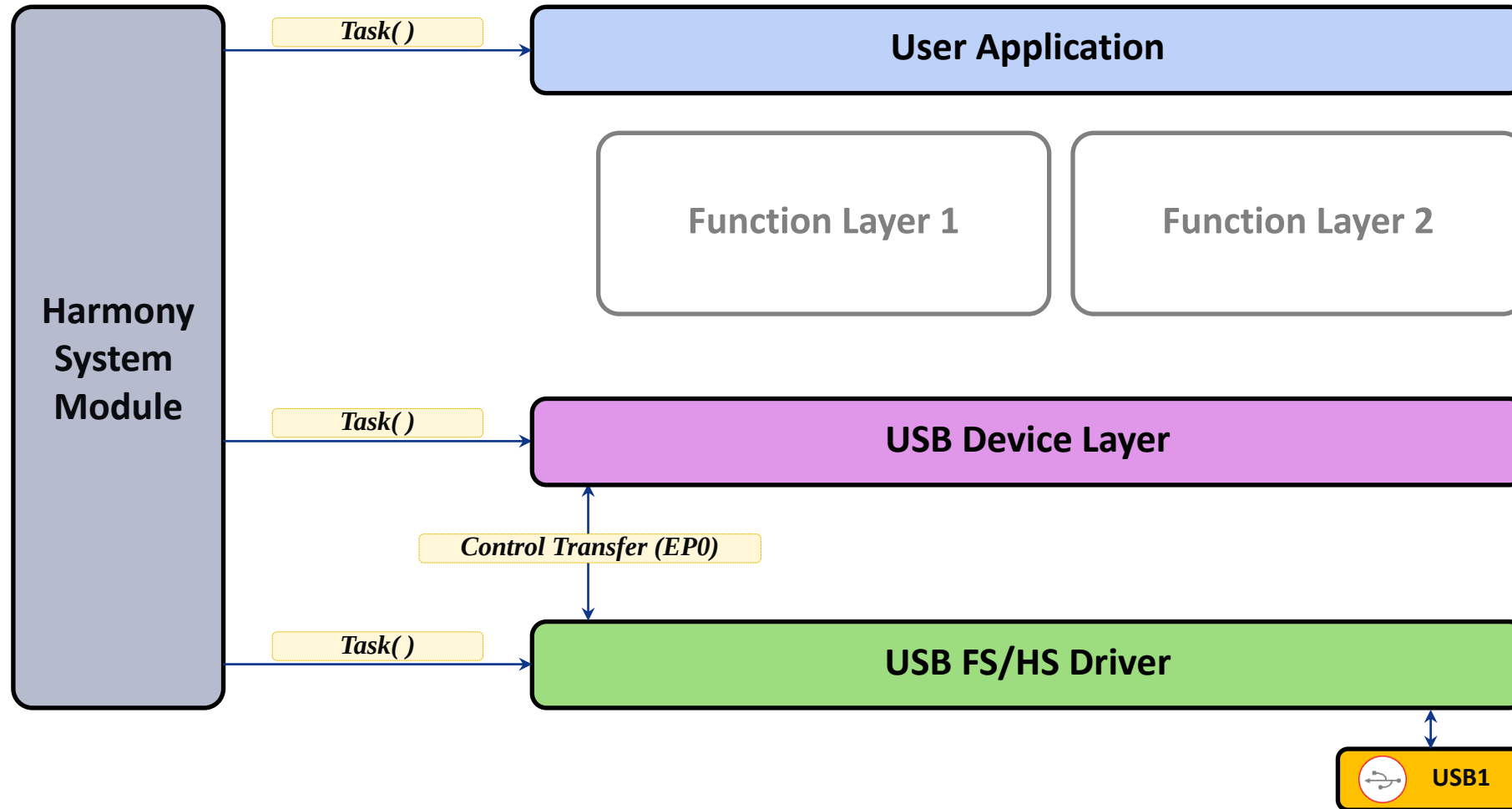
USB Stack Abstraction Model - Powered



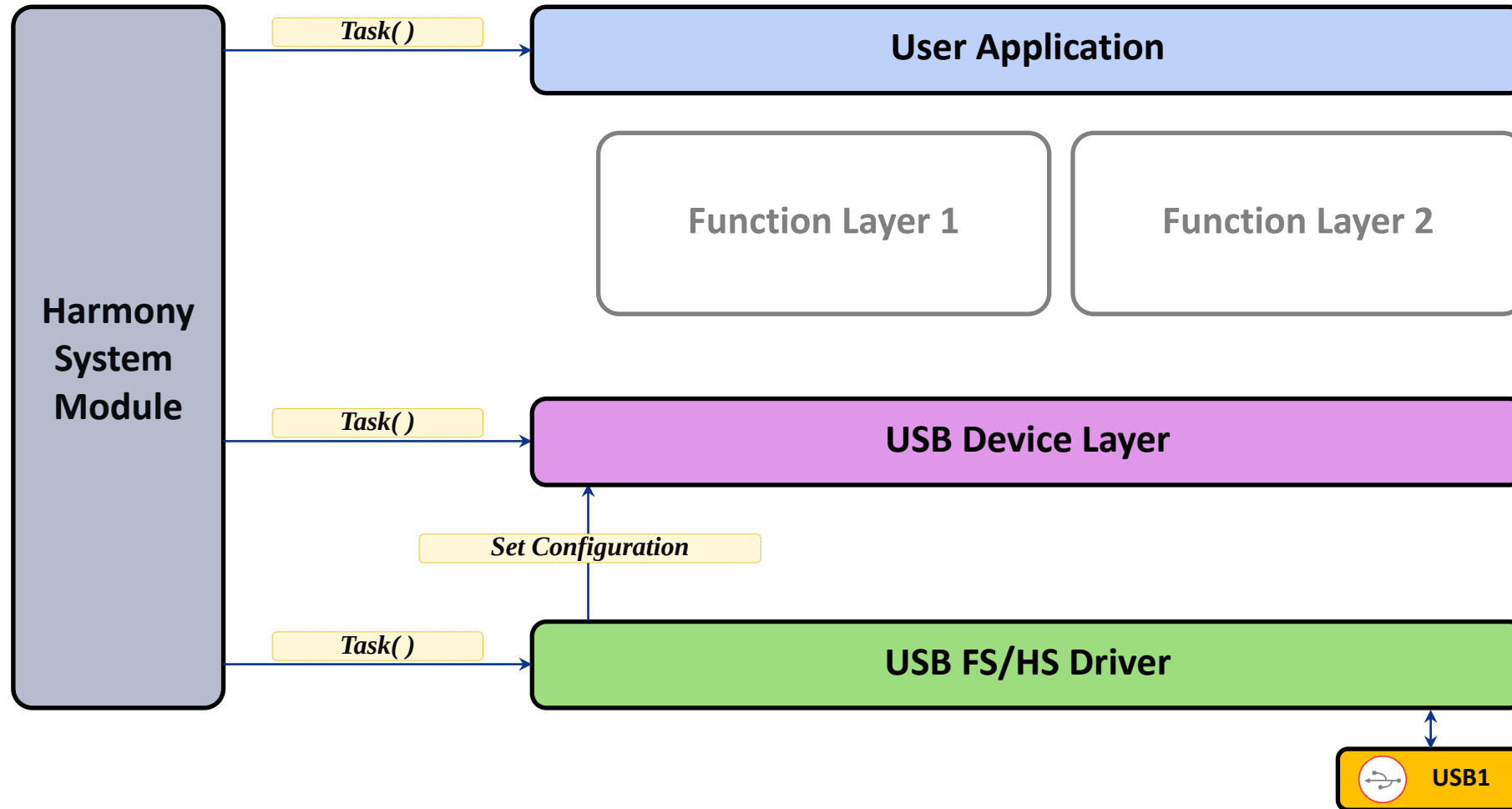
USB Stack Abstraction Model - Attach



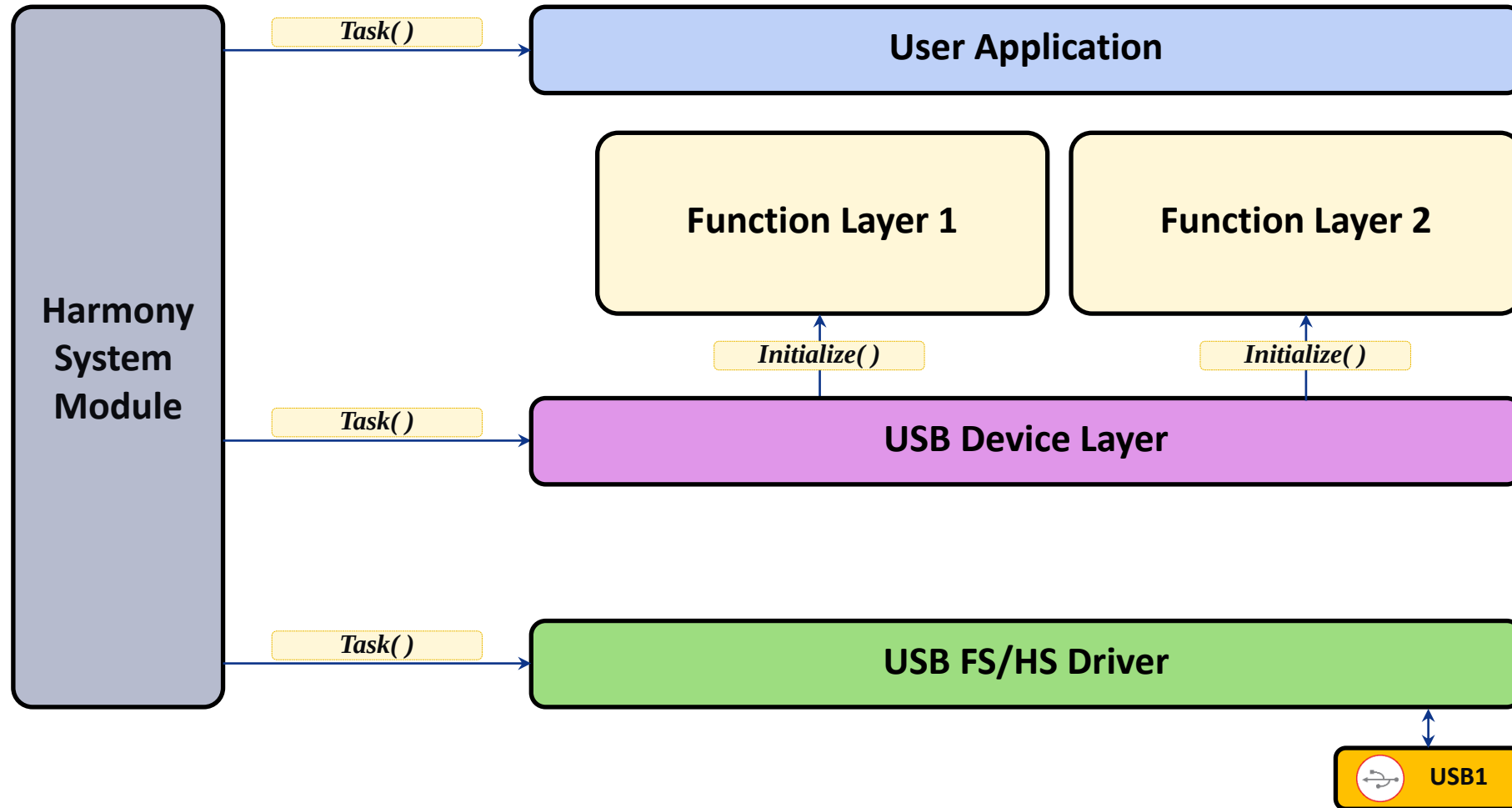
USB Stack Abstraction Model – Control Transfer



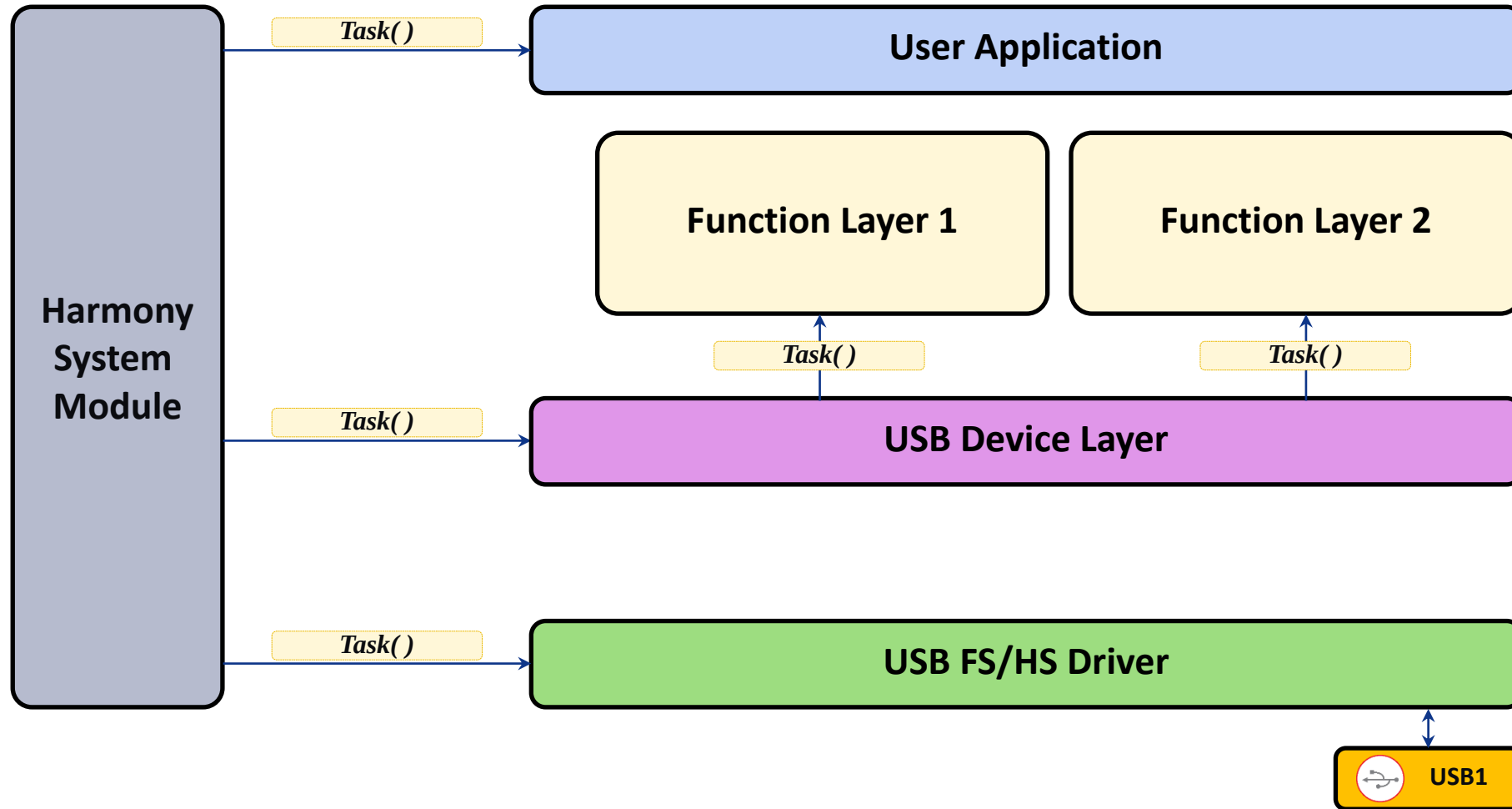
USB Stack Abstraction Model - Set Configuration



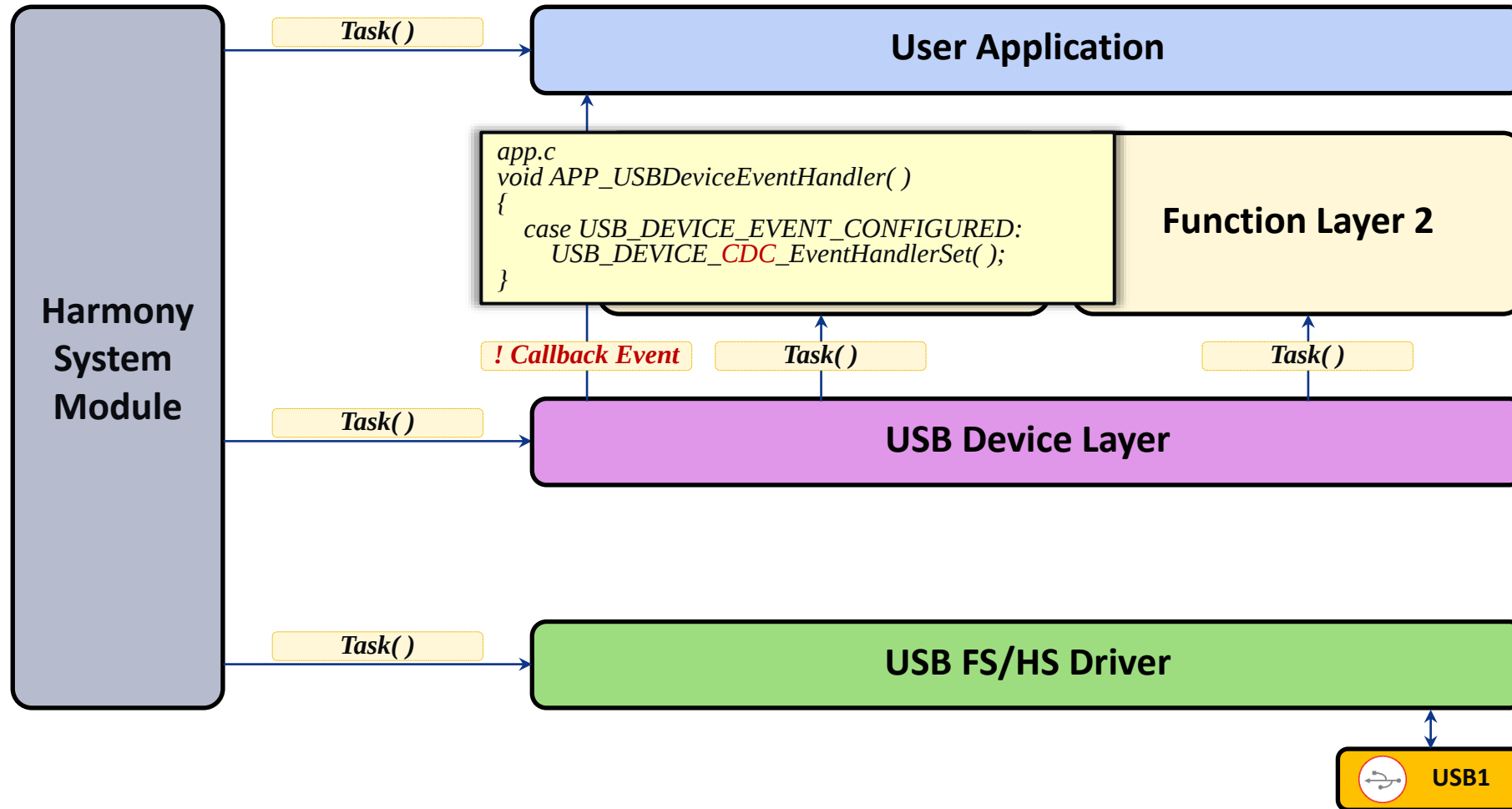
USB Stack Abstraction Model - Function Initialize & Open



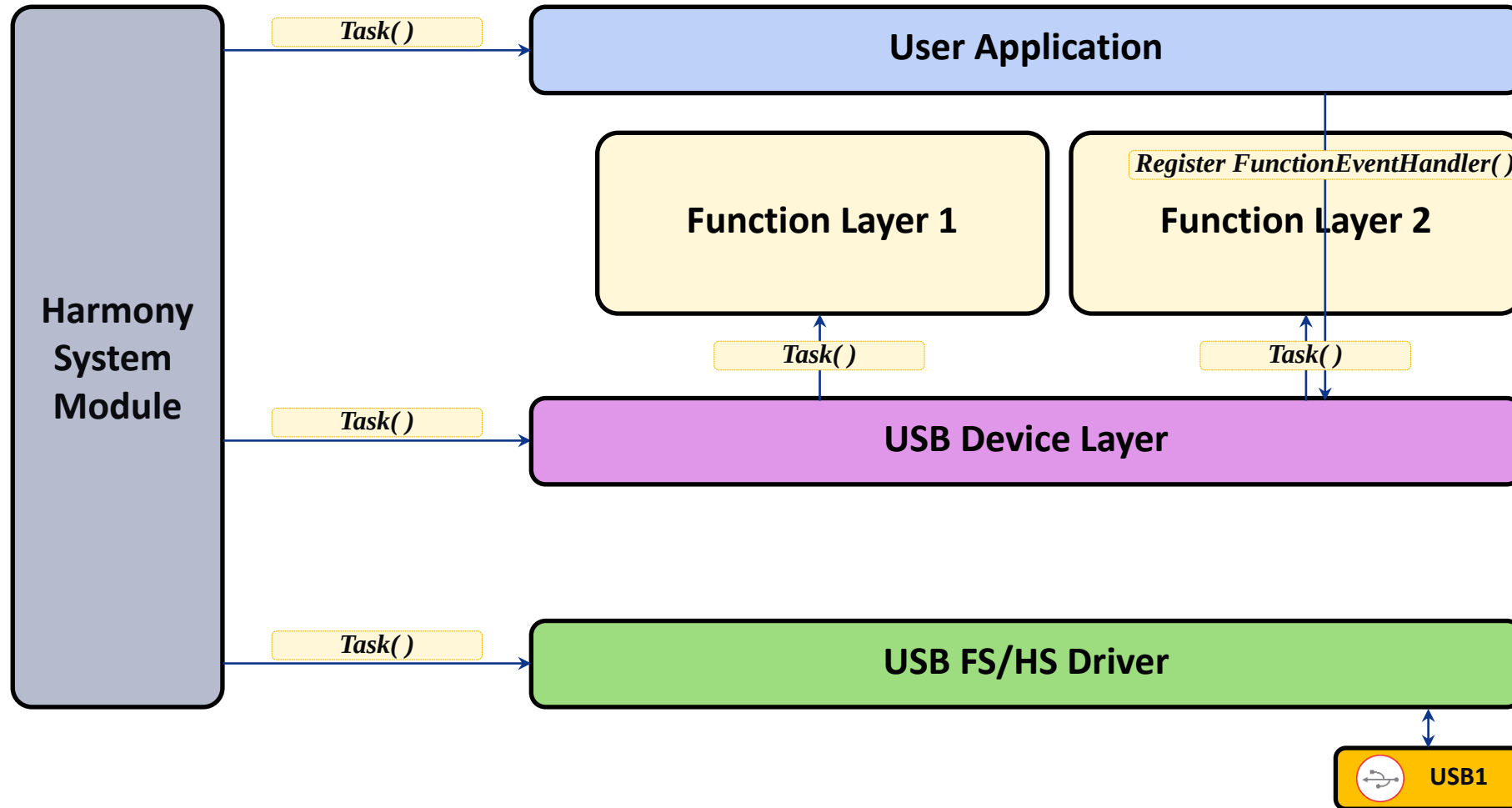
USB Stack Abstraction Model - Function Initialize & Open



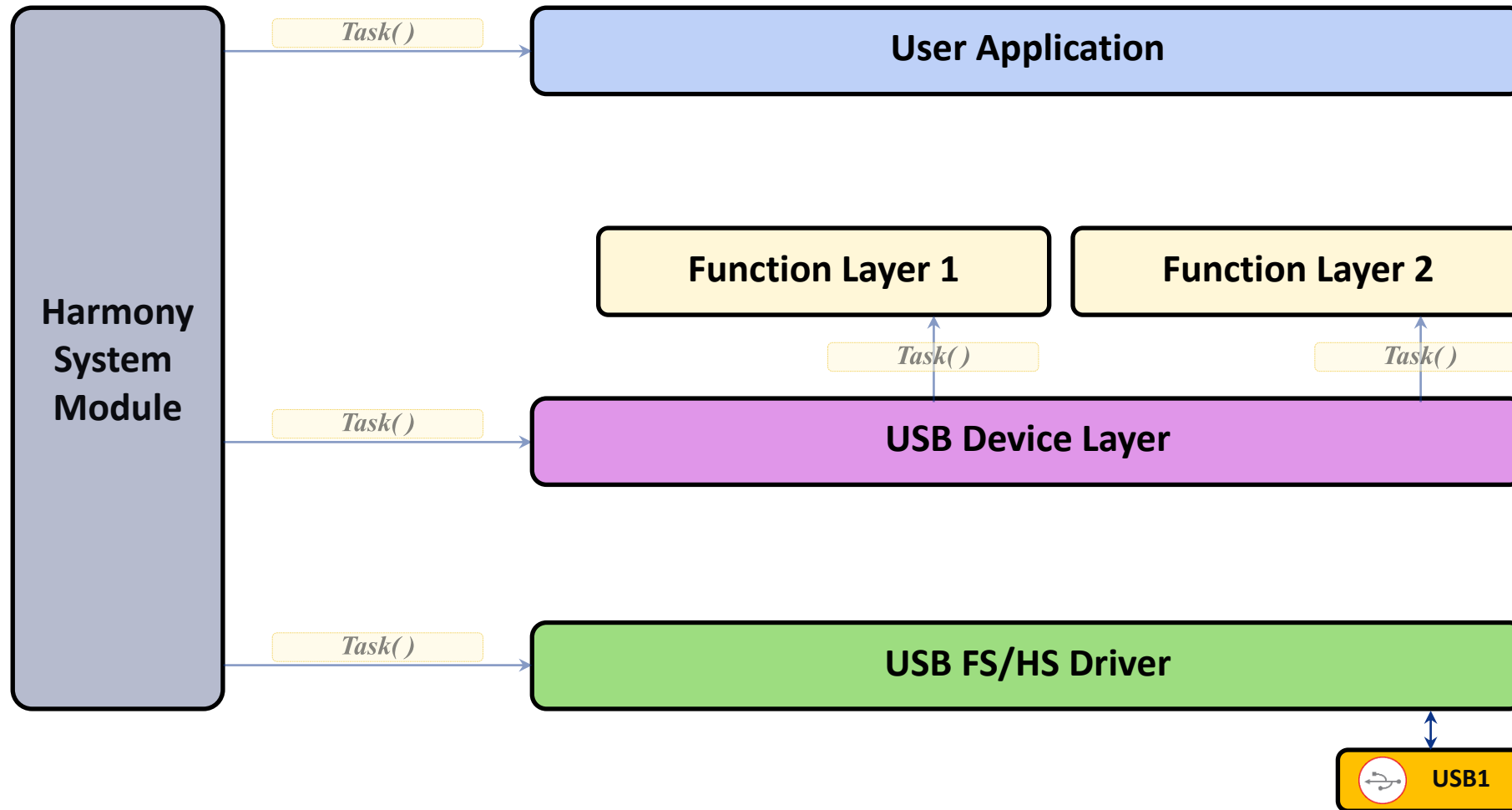
USB Stack Abstraction Model - Register Function Event Handler



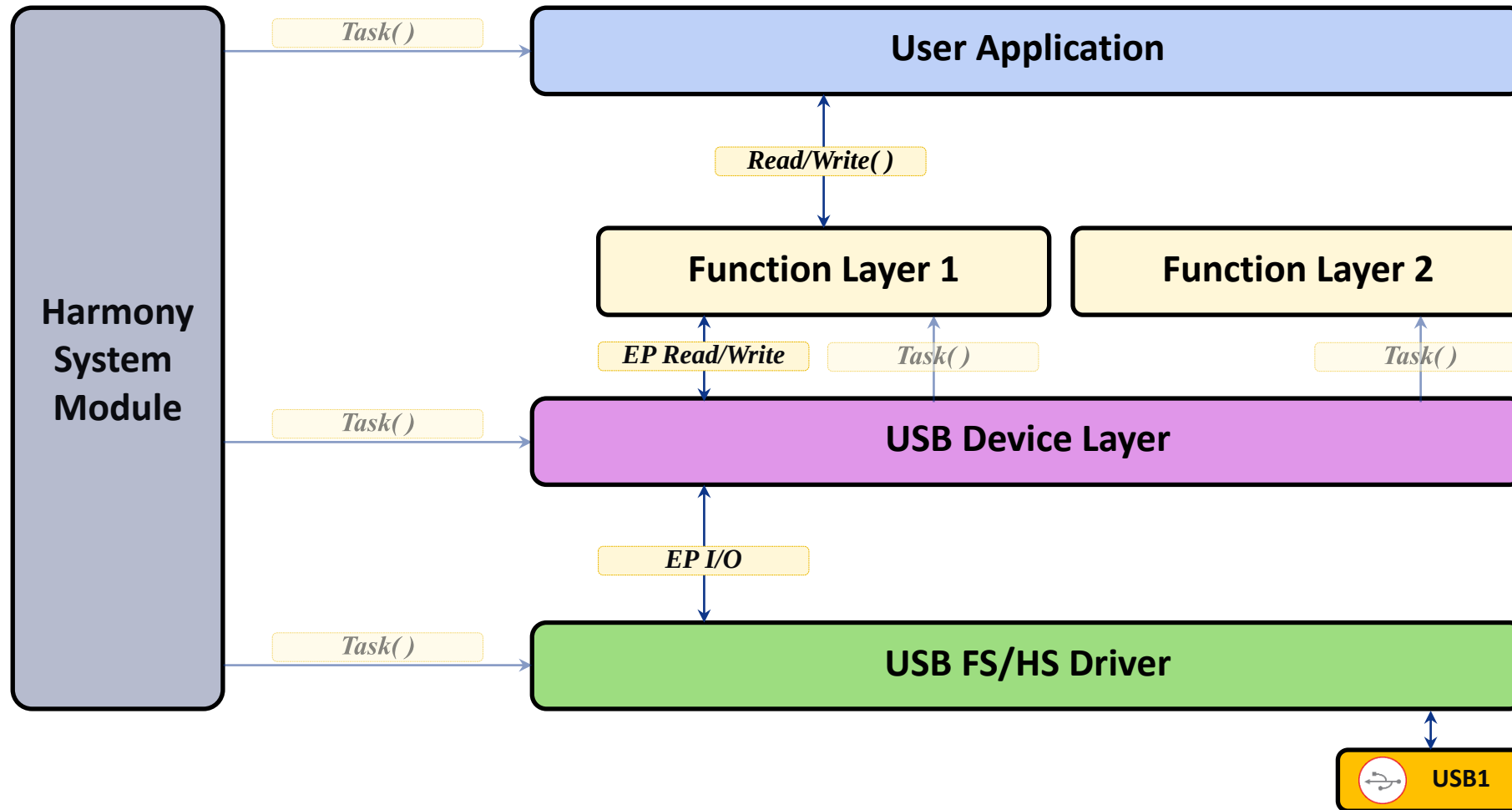
USB Stack Abstraction Model - Register Function Event Handler



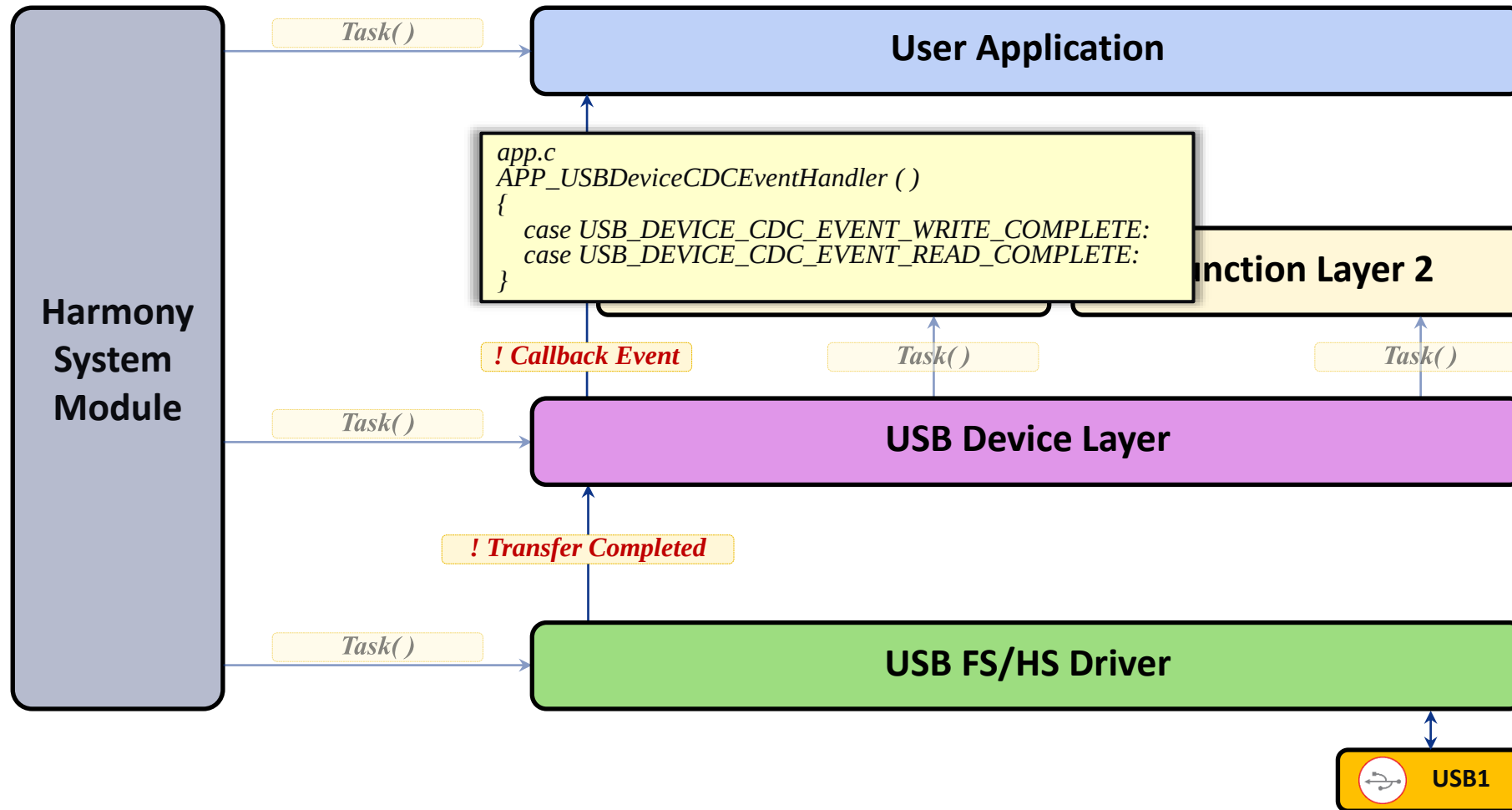
USB Stack Abstraction Model - Function Service Ready



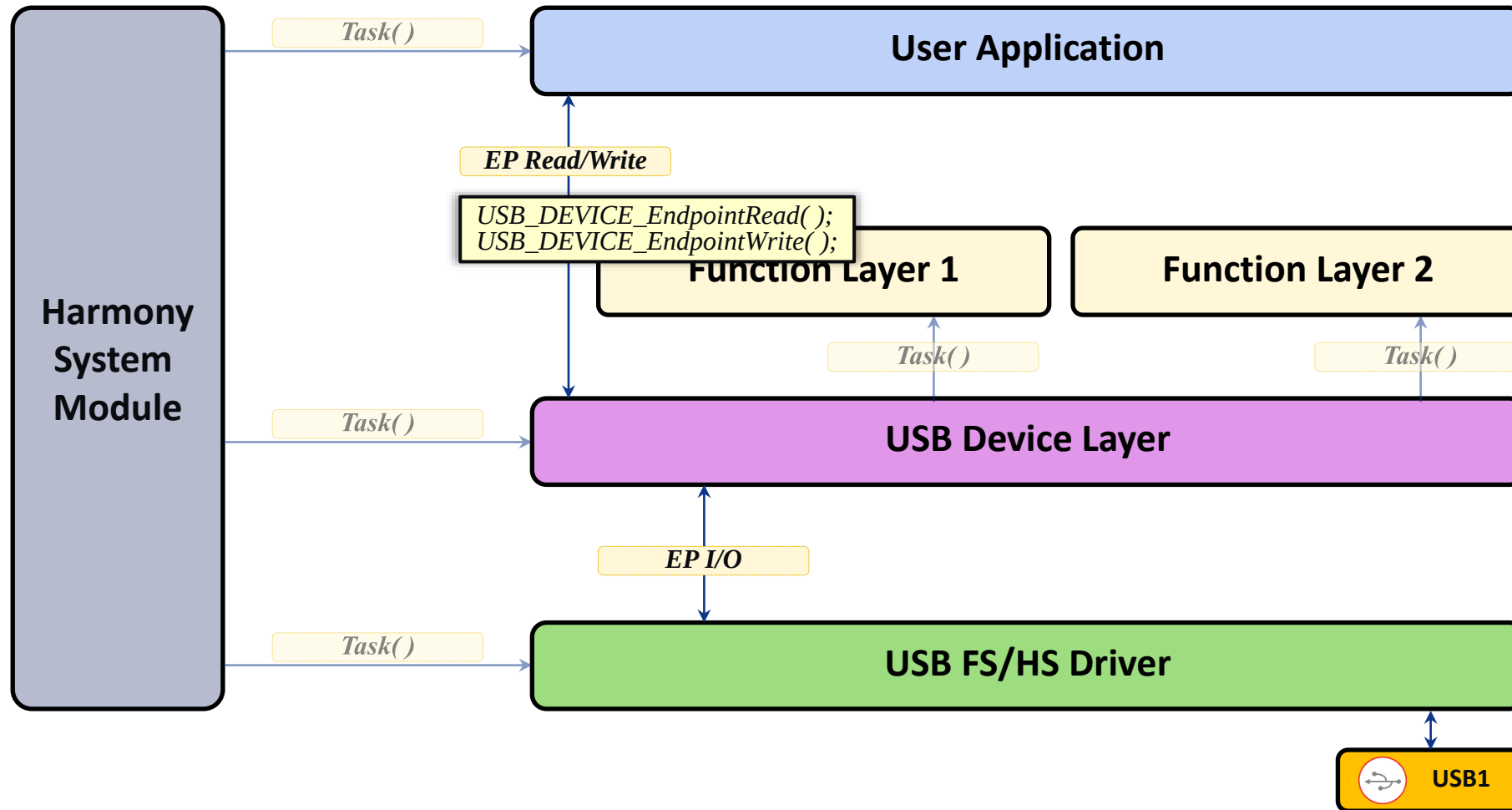
USB Stack Abstraction Model - Functionally Read/Write



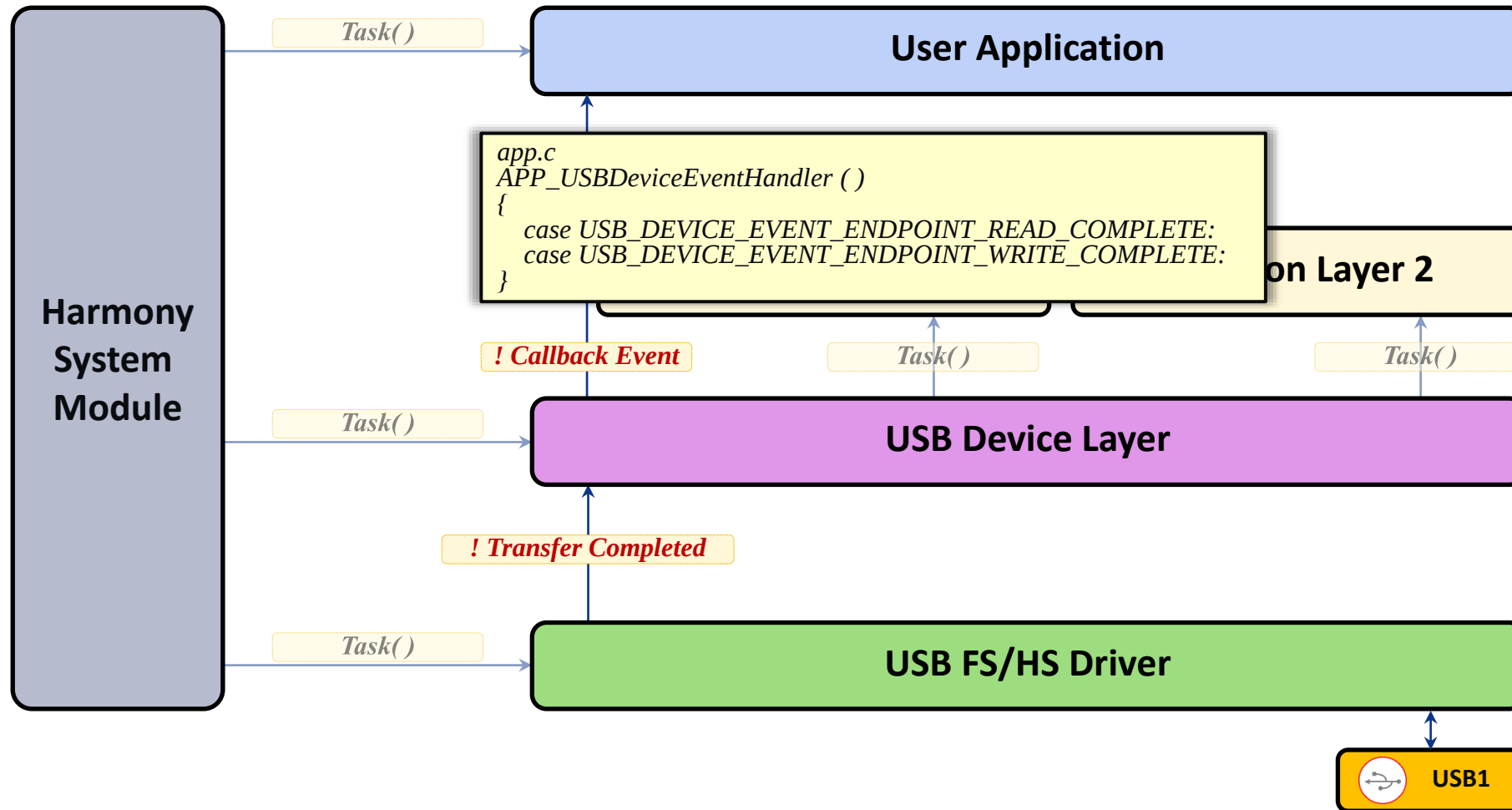
USB Stack Abstraction Model - Functionally Read/Write



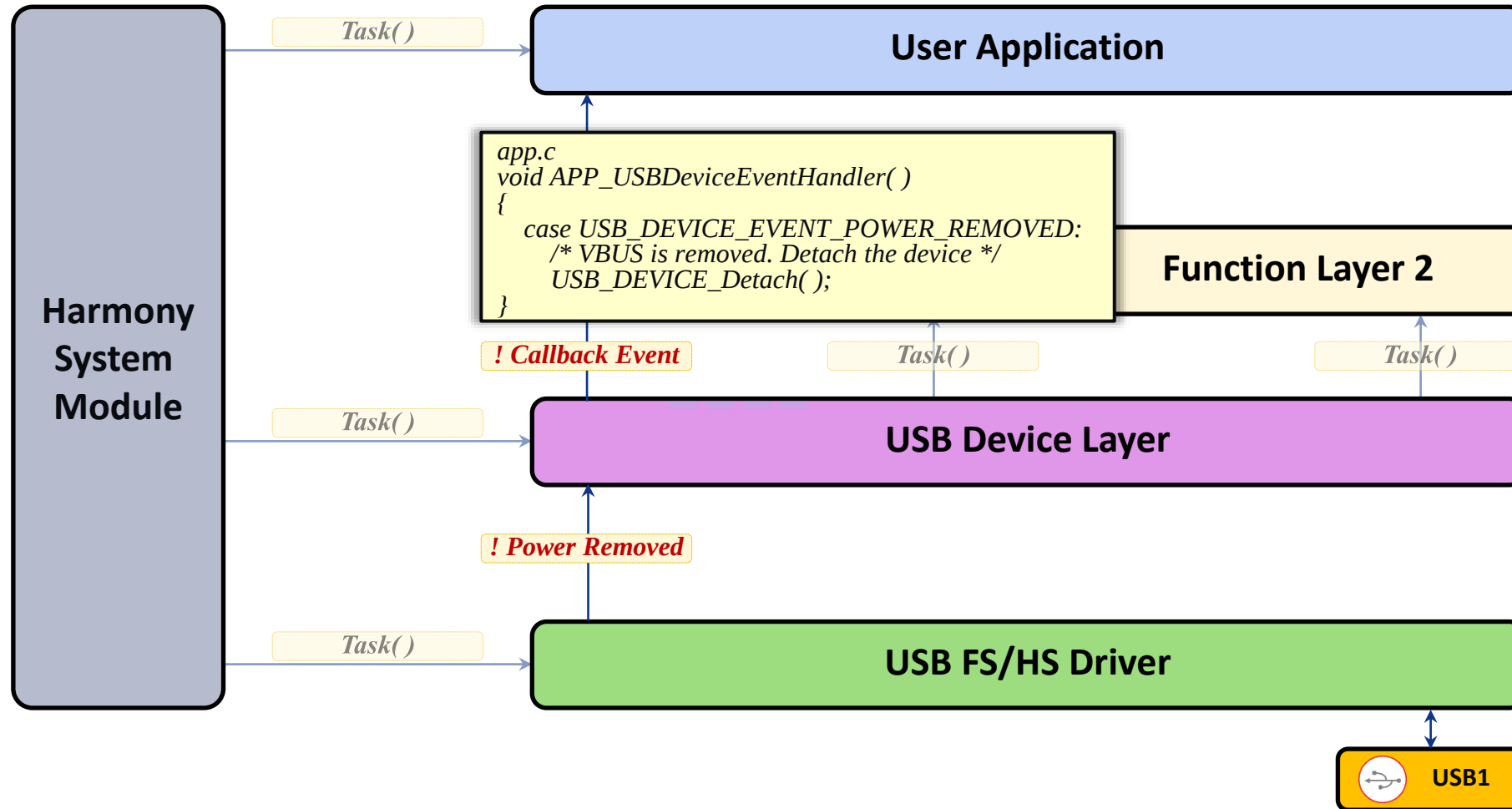
USB Stack Abstraction Model - Device Layer Read/Write



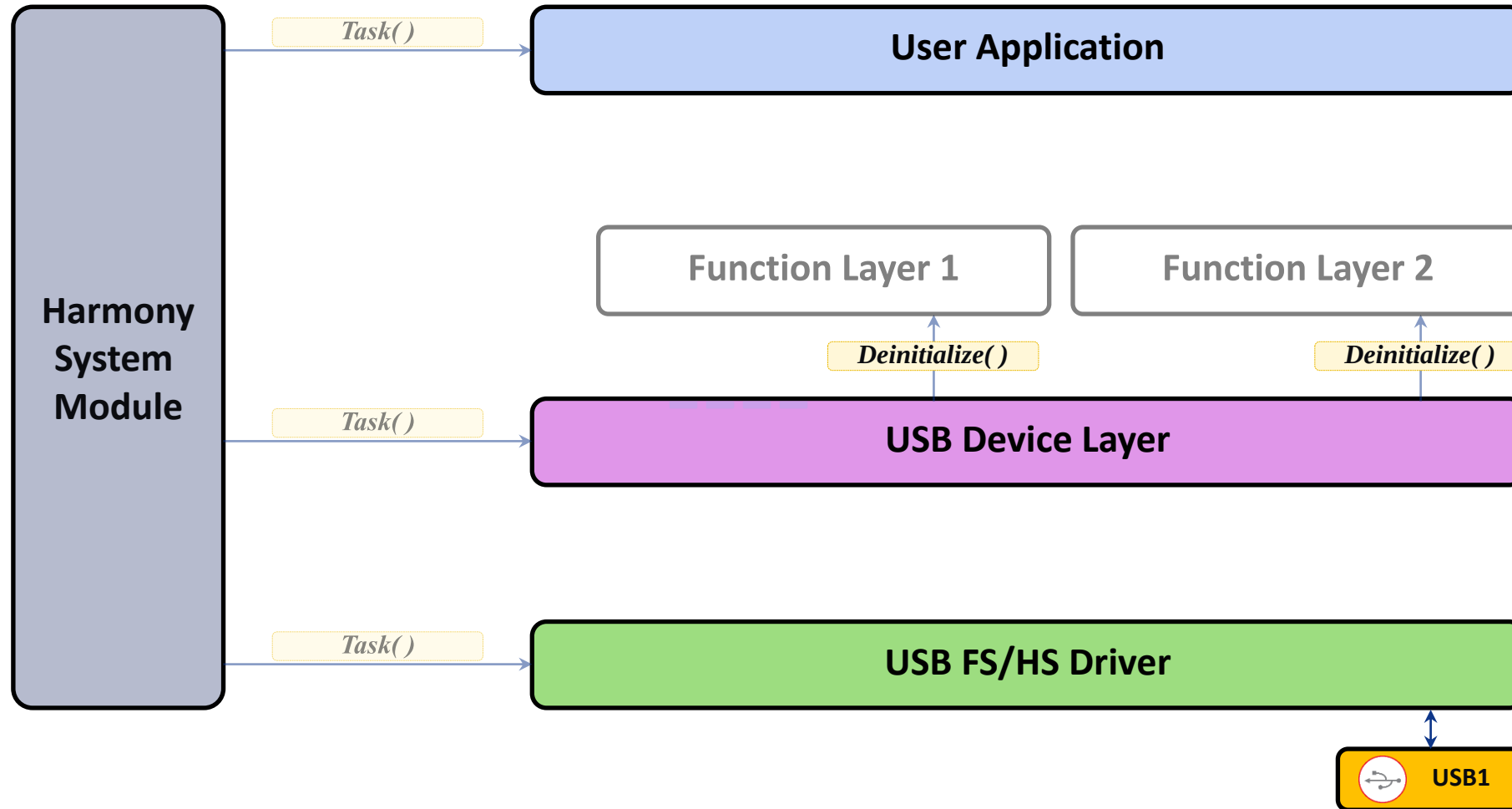
USB Stack Abstraction Model - Device Layer Read/Write



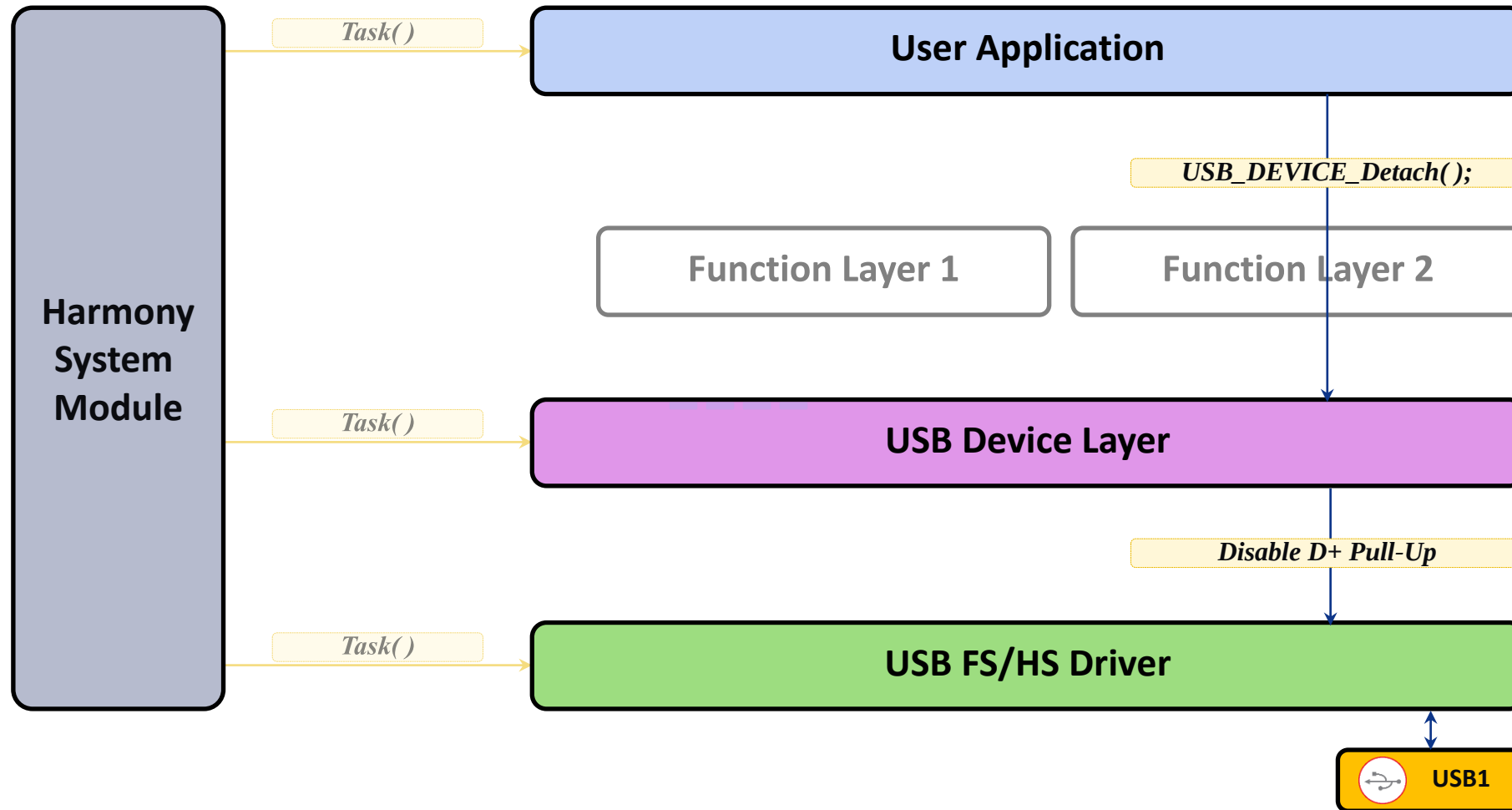
USB Stack Abstraction Model - Power Removed



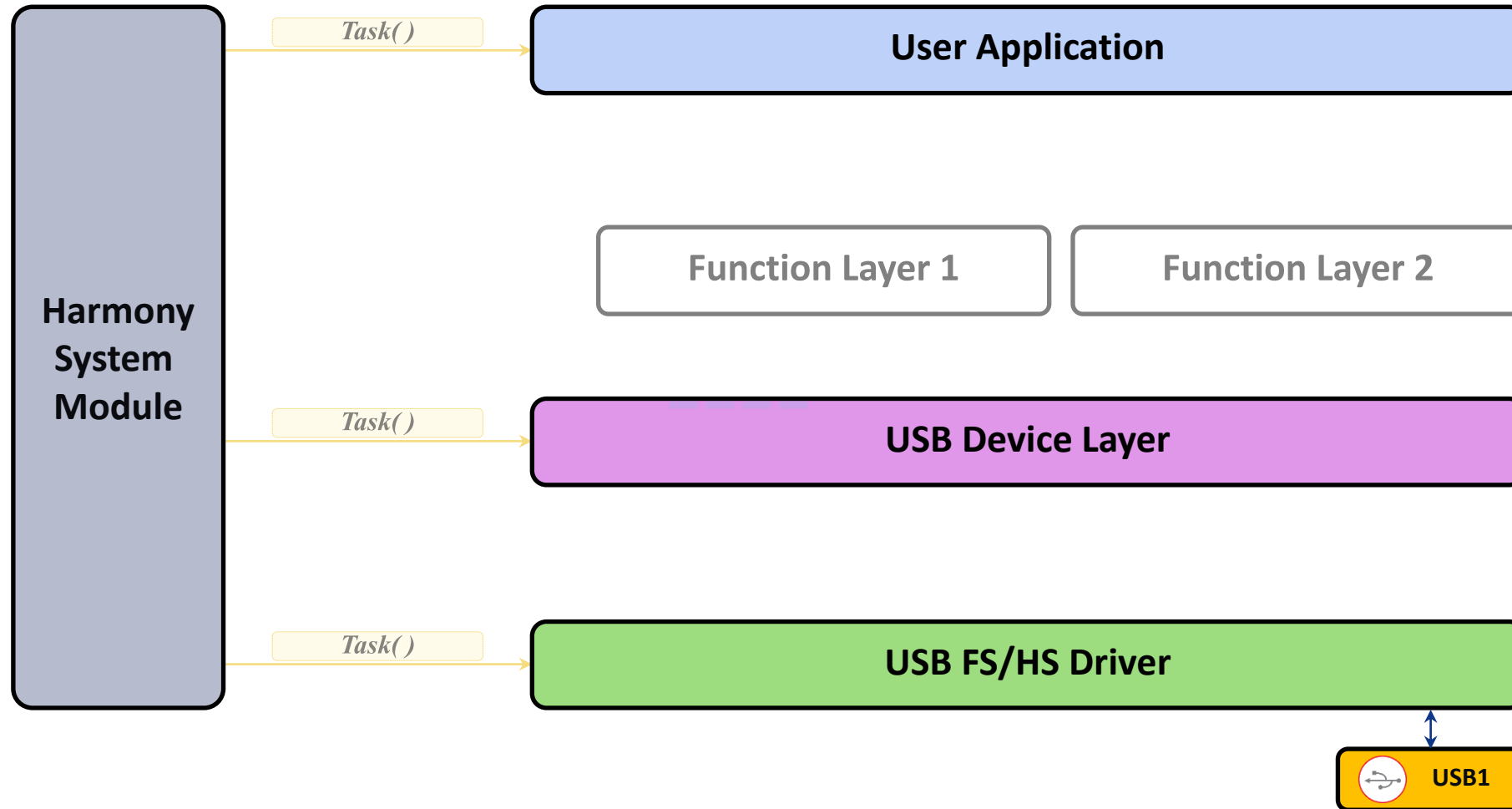
USB Stack Abstraction Model - Function Deinitialize



USB Stack Abstraction Model - Detach



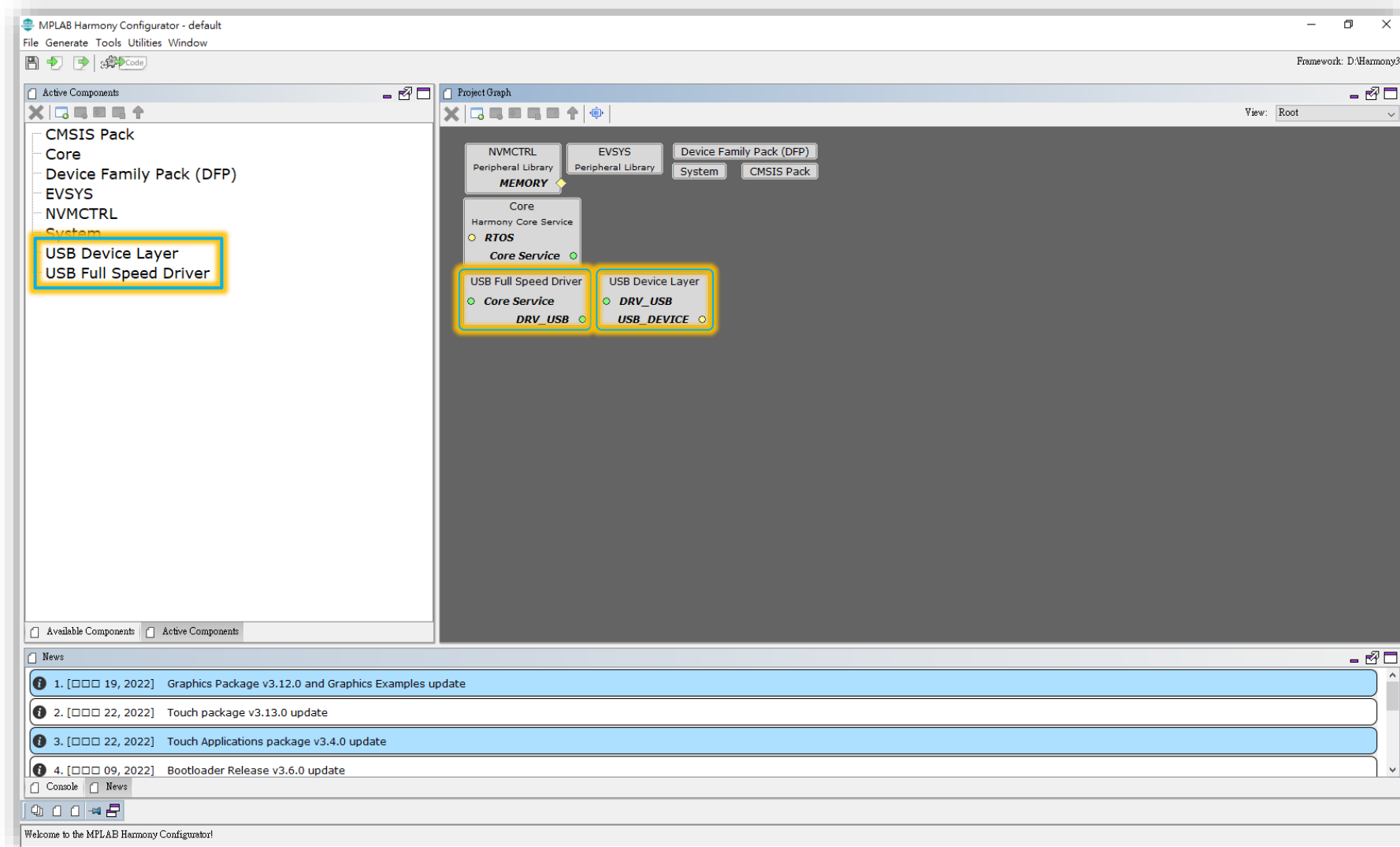
USB Stack Abstraction Model - Disconnection



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

MPLAB Harmony Configurator - default

File Generate Tools Utilities Window

Framework: D:\Harmony3\

Active Components

- CMSIS Pack
- Core
- Device Family Pack (DFP)
- EVSYS
- NVMCTRL
- System
- USB Device Layer
- USB Full Speed Driver

Pin Table

Package: TQFP48

Module	Function	PA09	PA10	PA11	VDIO0	VDIO1	FB10	FB11	PA12	PA13	PA14	PA15	PA16	PA17	PA18	PA19	PA20	PA21	PA22	PA23	USB_DM	USB_DP	VDIO2	VDIO3	FB22	FB23	PA27	RESET	PA28	VDIO4	VDIO5	PA30
TC4_W01	TC4_W01																															
TC5	TC5_W00																															
TC5	TC5_W01																															
TCC0	TCC0_W00																															
TCC0	TCC0_W01																															
TCC0	TCC0_W02																															
TCC0	TCC0_W03																															
TCC0	TCC0_W04																															
TCC0	TCC0_W05																															
TCC0	TCC0_W06																															
TCC0	TCC0_W07																															
TCC1	TCC1_W00																															
TCC1	TCC1_W01																															
TCC1	TCC1_W02																															
TCC1	TCC1_W03																															
TCC2	TCC2_W00																															
TCC2	TCC2_W01																															
USB	USB_DM																															
USB	USB_DP																															
USB	USB_SOF_1KHZ																															

Available Components Active Components

Project Graph Pin Settings Pin Table Pin Diagram

News

1. [000 19, 2022] Graphics Package v3.12.0 and Graphics Examples update
2. [000 22, 2022] Touch package v3.13.0 update
3. [000 22, 2022] Touch Applications package v3.4.0 update
4. [000 09, 2022] Bootloader Release v3.6.0 update

Console News

Welcome to the MPLAB Harmony Configurator!

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_FDPLL	☐	GCLK0	--
SYSCTRL_FDPLL32K	☐	GCLK0	--
TC3, TCC2	☐	GCLK0	--
TC4, TC5	☐	GCLK0	--
TCC0, TCC1	☐	GCLK0	--
USB	☒	GCLK0	48,000,000 Hz
WDT	☐	GCLK0	--

Close

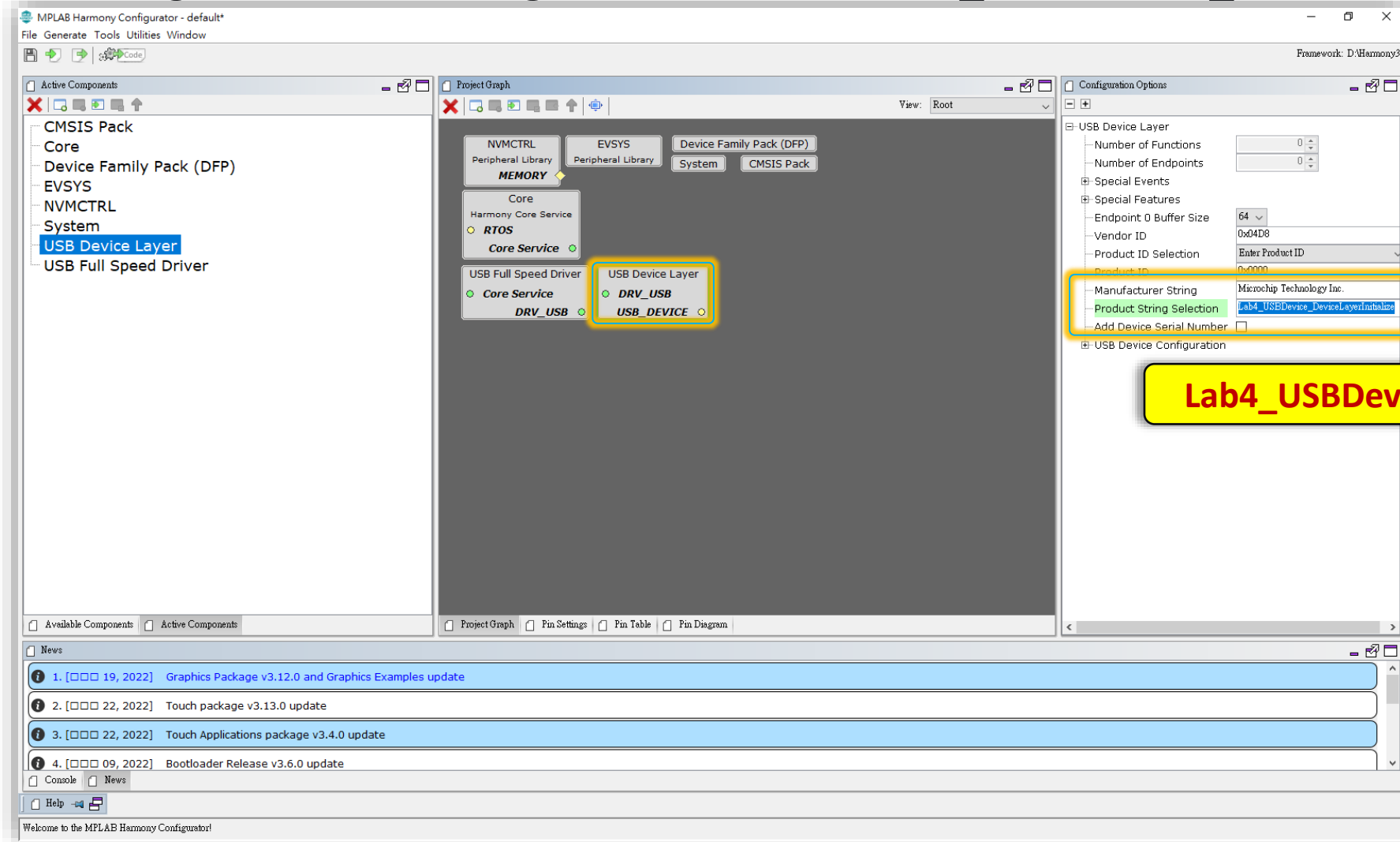
Lab4 USB Device Device Layer Initialize

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



⚡ Lab4 USB Device Device Layer Initialize

- Change “Product String Selection” to “Lab4_USBDevice_DeviceLayerInitialize”



Lab4 USB Device Device Layer Initialize

- Implement below function at app.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 4.09
    bool deviceIsConfigured;
} APP_DATA;
```

⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.

```
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 4.02
            appData.usbDevHandle = USB_DEVICE_Open(USB_DEVICE_INDEX_0, 0);
            appInitialized &= (appData.usbDevHandle != USB_DEVICE_HANDLE_INVALID);

            if (appInitialized)
            {
                // TODO 4.03
                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.

```
// TODO 4.04
// TODO 4.05
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:

            // This means the device detected a valid VBUS voltage
            // and is attached to the USB if the device is bus powered.
            // 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 4.10
            USB_DEVICE_Detach(appData.usbDevHandle);
            appData.deviceIsConfigured = false;
            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.

```
// TODO 4.04
// TODO 4.05
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)
            {
                // Device is enumerated. Register here the USB Function Driver Event Handler function.
                // TODO 4.08
                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;
}
```

⚙️ Lab4 USB Device Device Layer Initialize

- Implement below function at app.c.

```
// TODO 4.04
// TODO 4.05
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_CONTROL_TRANSFER_DATA_RECEIVED:
            // TODO 4.06
            //USB_DEVICE_ControlStatus(usbDeviceHandle, USB_DEVICE_CONTROL_STATUS_OK);
            USB_DEVICE_ControlStatus(appData.usbDevHandle, USB_DEVICE_CONTROL_STATUS_OK);
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_DATA_SENT:
            break;

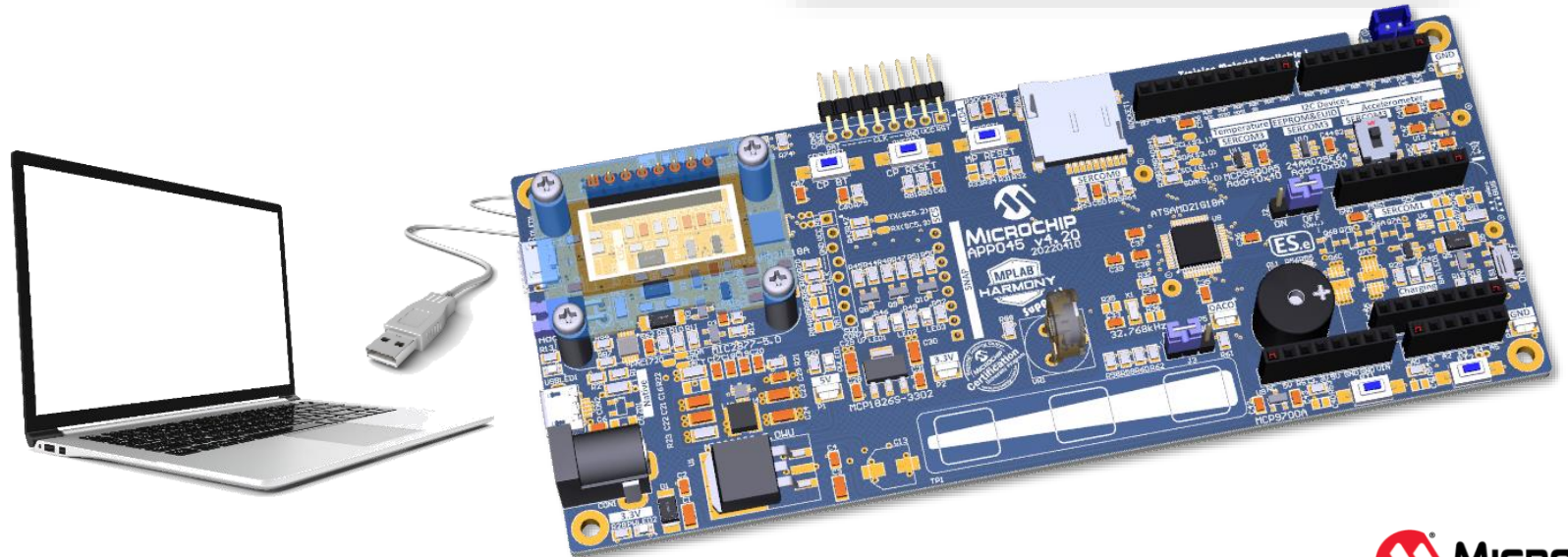
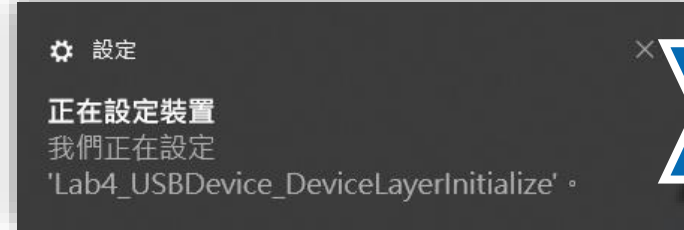
        case USB_DEVICE_EVENT_CONTROL_TRANSFER_ABORTED:
            break;

        default:
            break;
    }

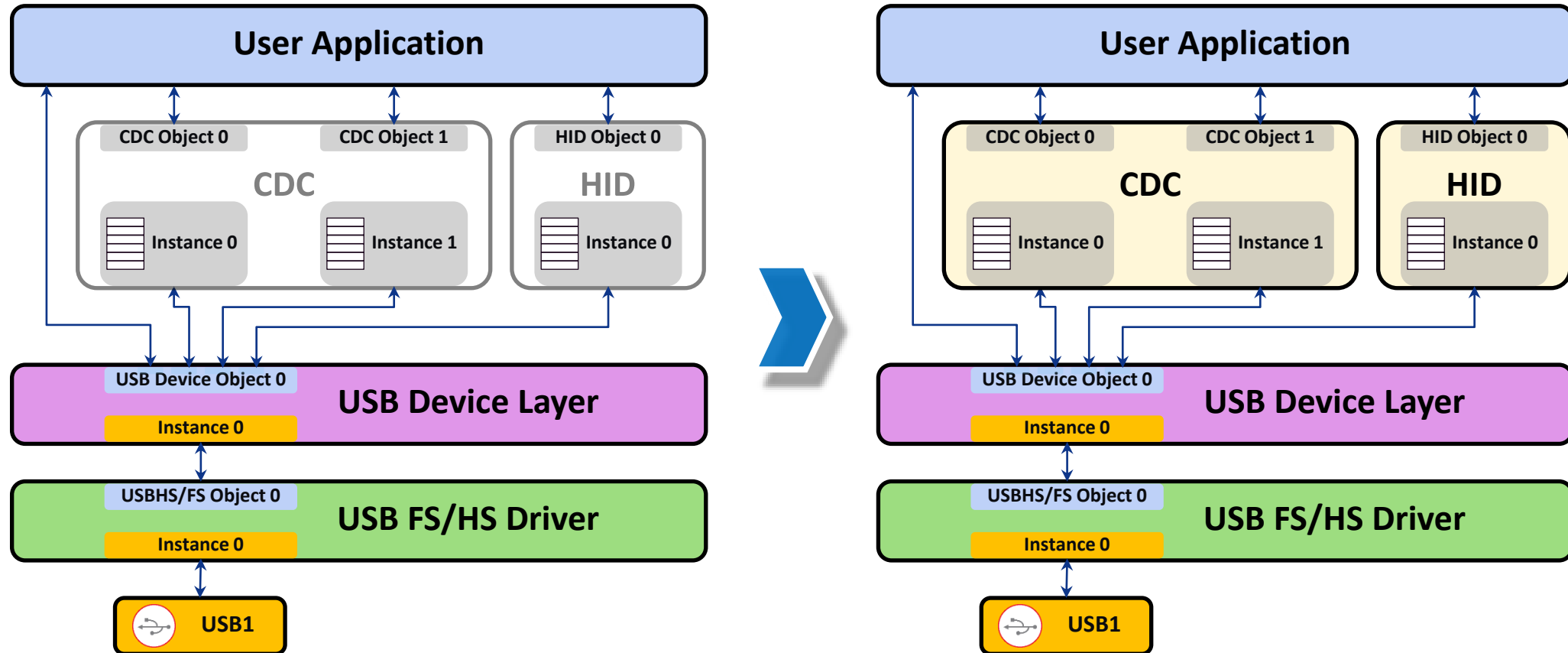
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

✖ 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)

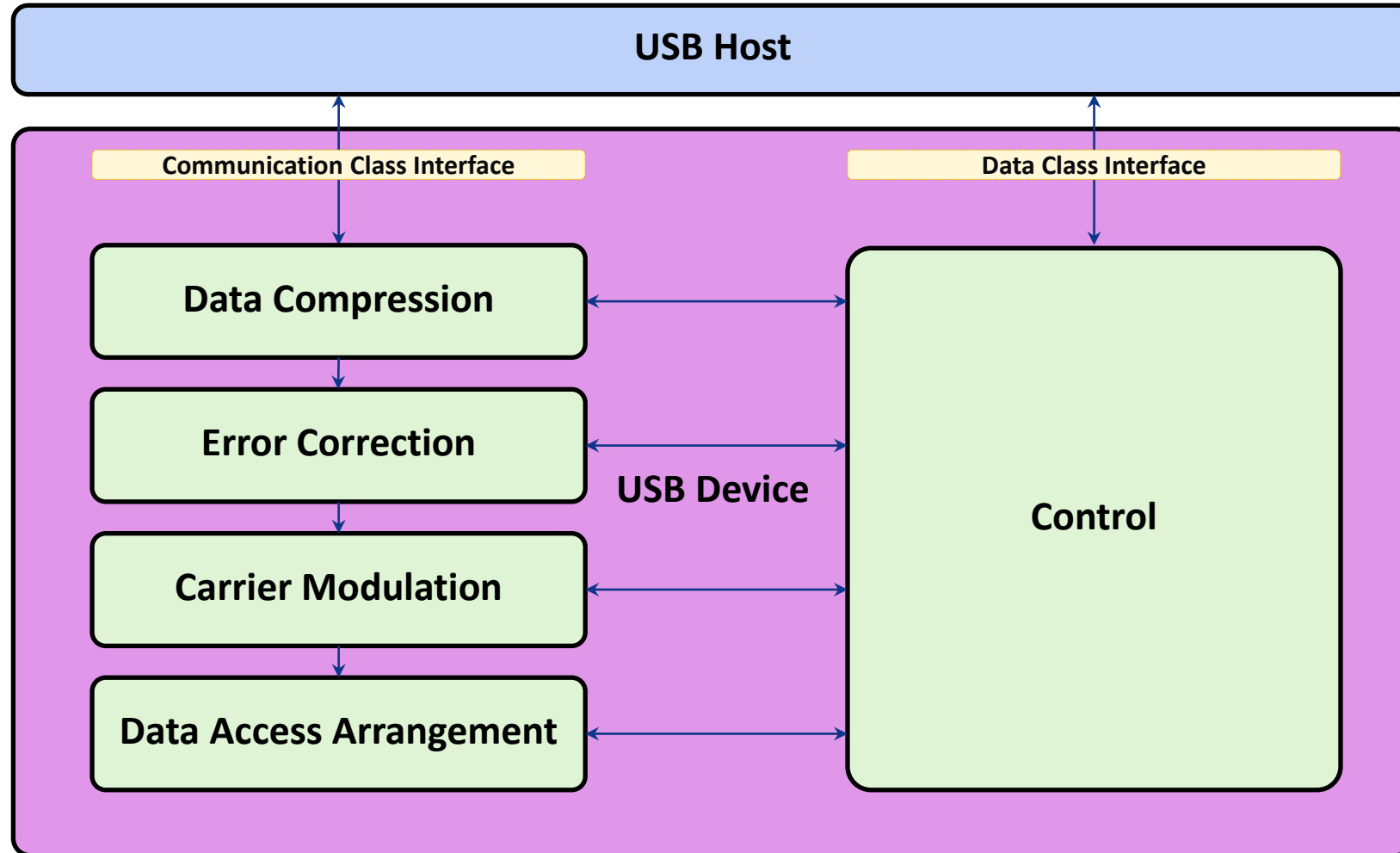
Communications Class Subclass Codes

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	[USBEE1.0]
0Dh	Network Control Model	[USBNCM1.0]
0Dh-7Fh	RESERVED (future use)	
80-FEh	RESERVED (vendor specific)	

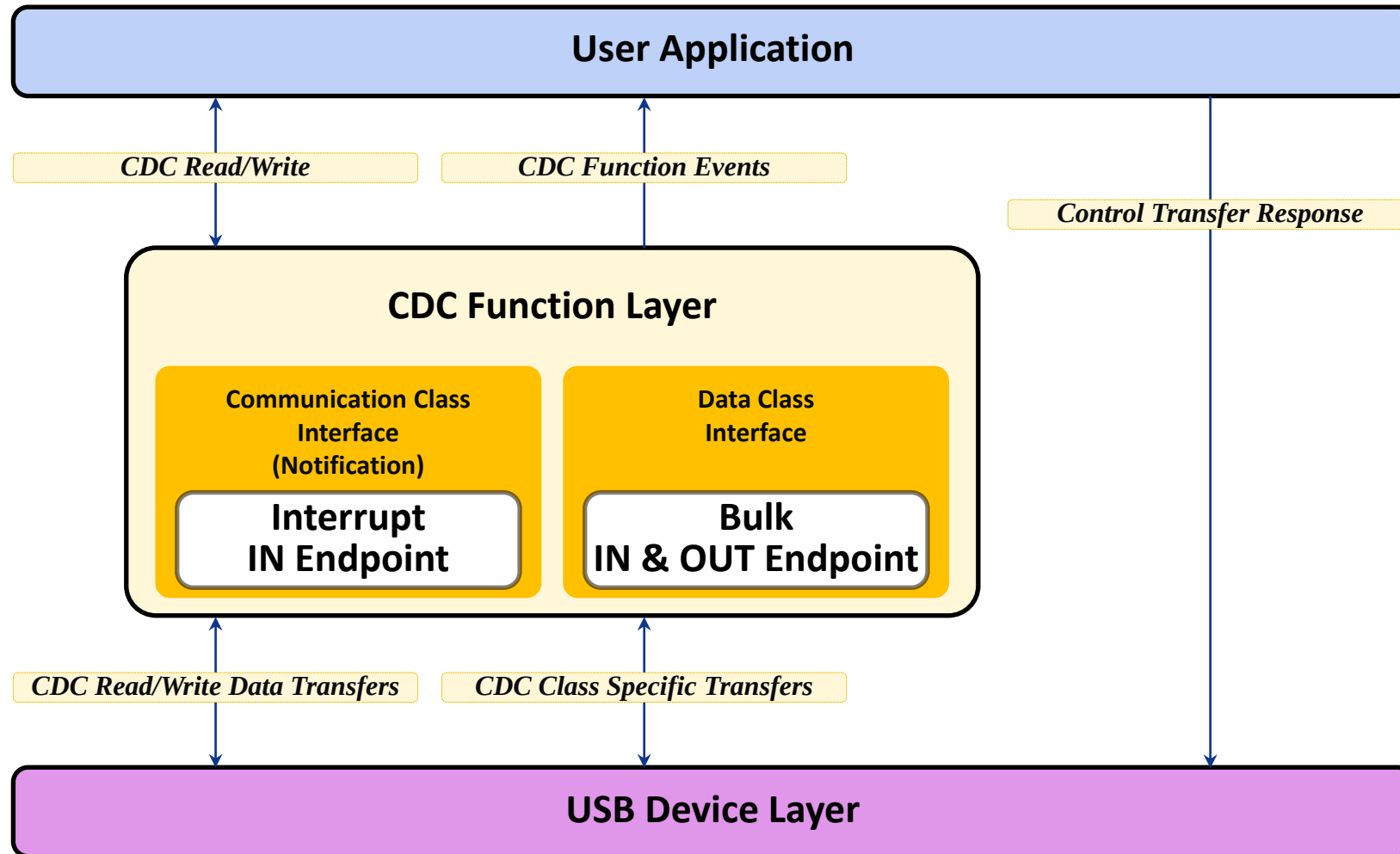
Communications Class Protocol Codes

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
    ...
}
```

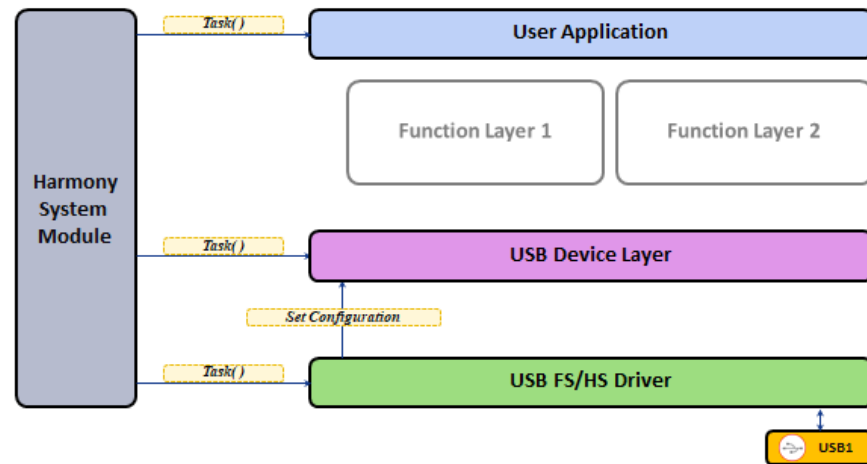
```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // Descriptor Type is Int
    0, // Interface Number
    0x00, // Alternate Setting Number
    0x01, // Number 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
    ...
}
```

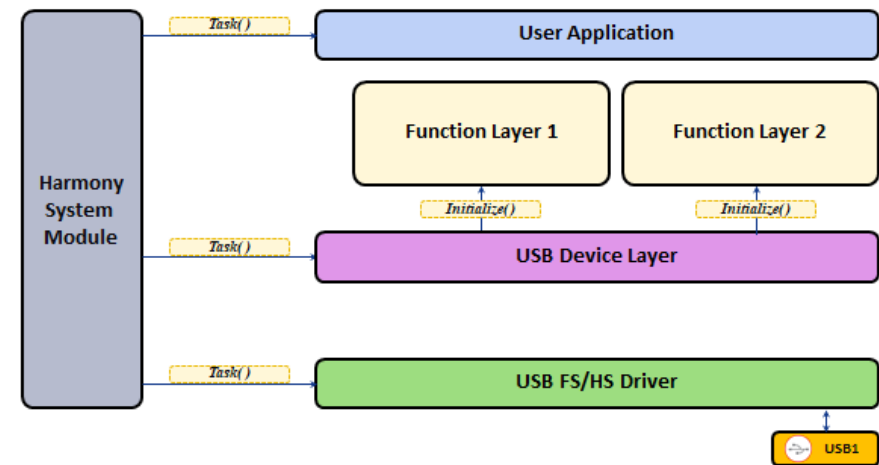
Review USB Stack Abstraction Model

- 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

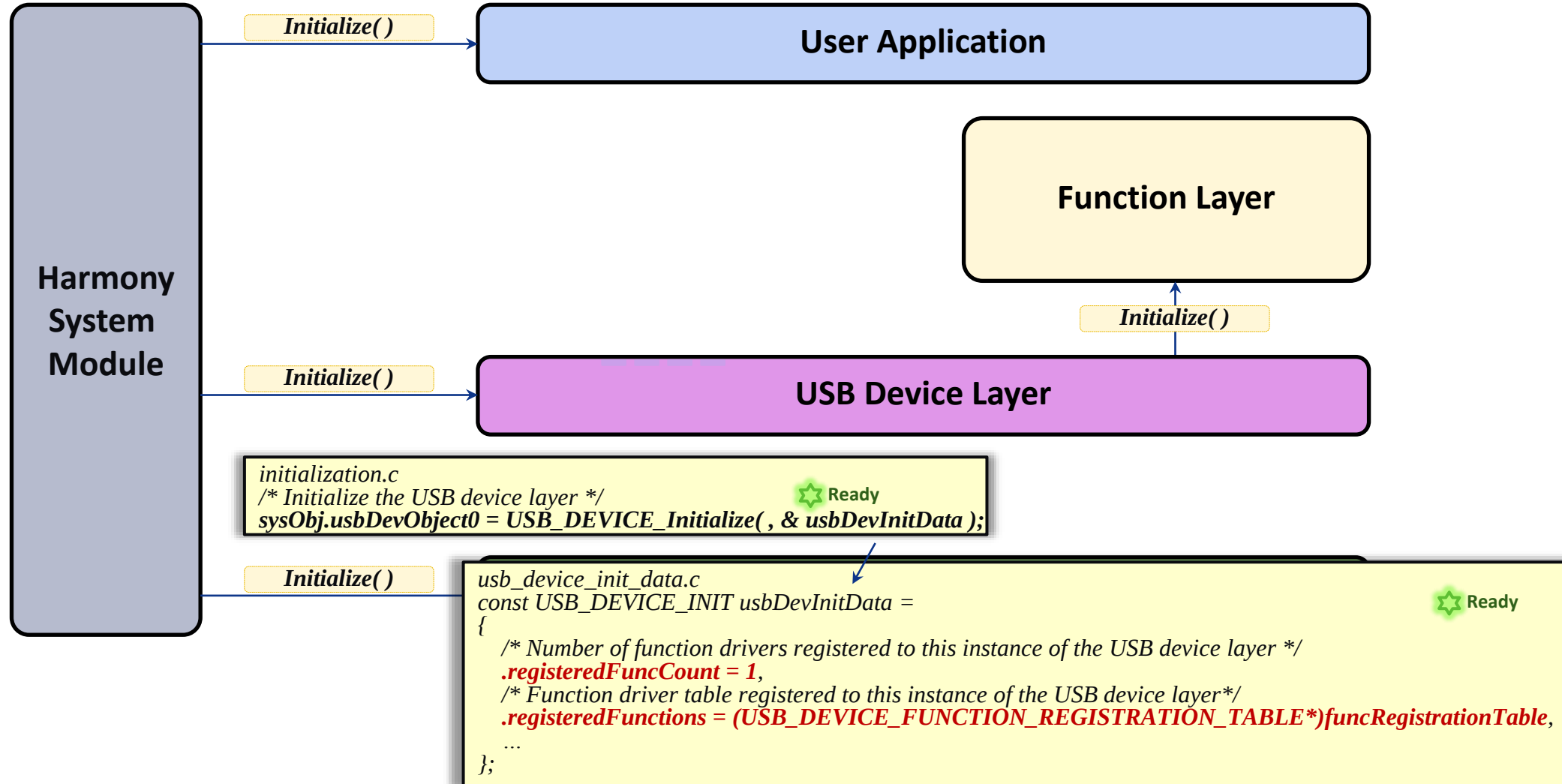


USB Stack Abstraction Model - Function Initialize & Open



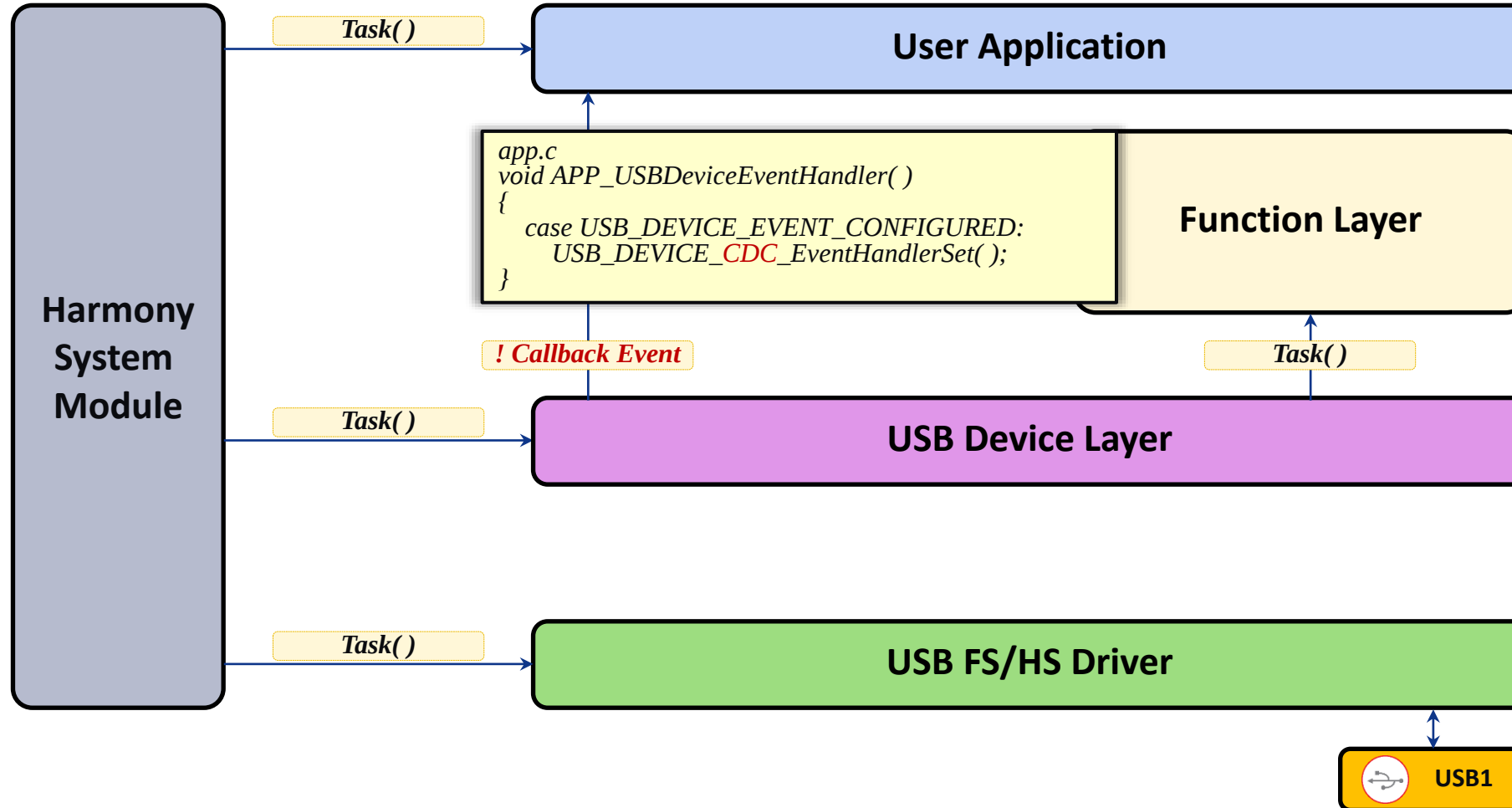
Review USB Stack Abstraction Model

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

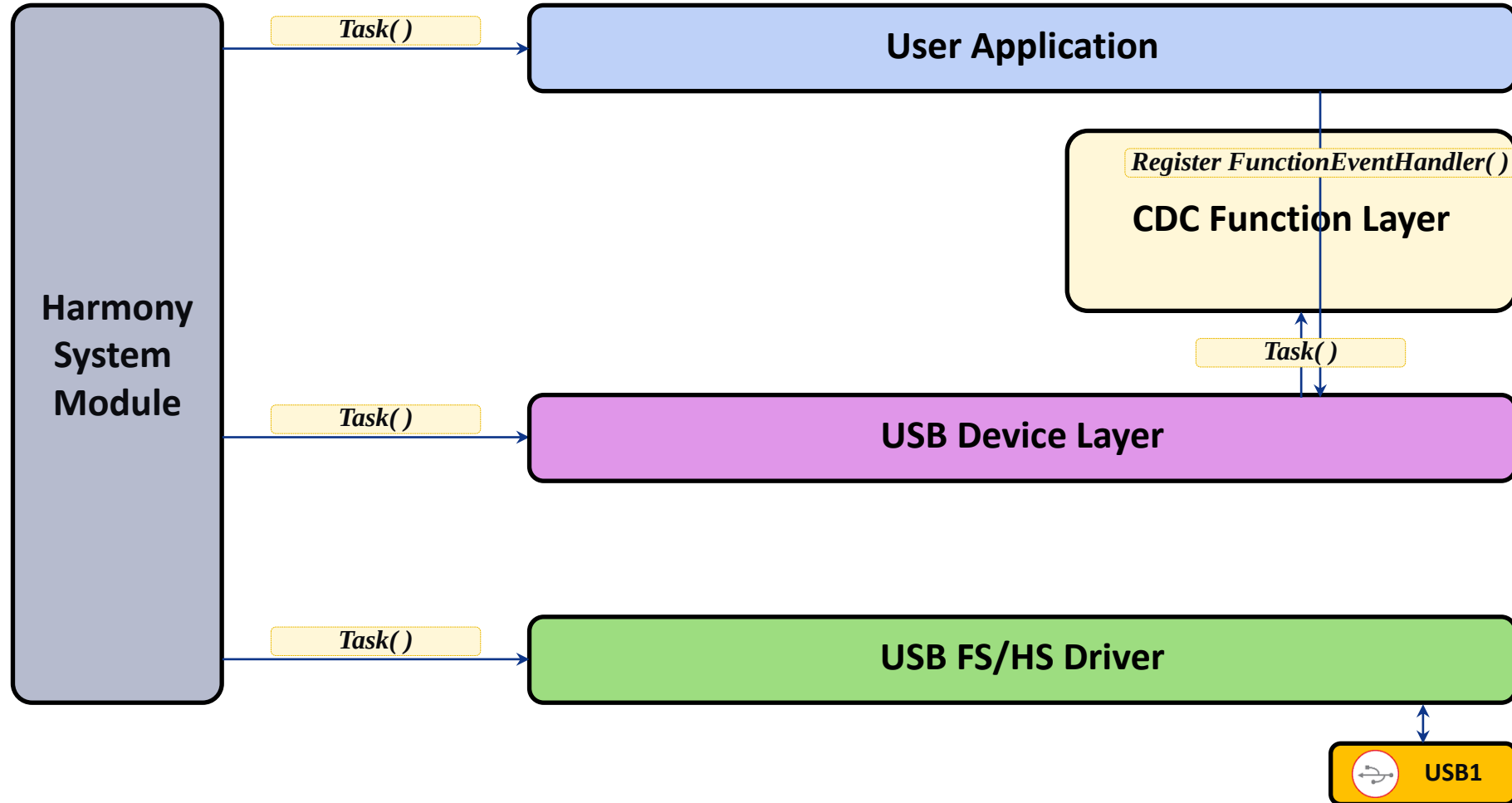


USB Stack Abstraction Model - Register Function Event Handler

- 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_TRANSFER_FLAGS_DATA_COMPLETE  
);
```

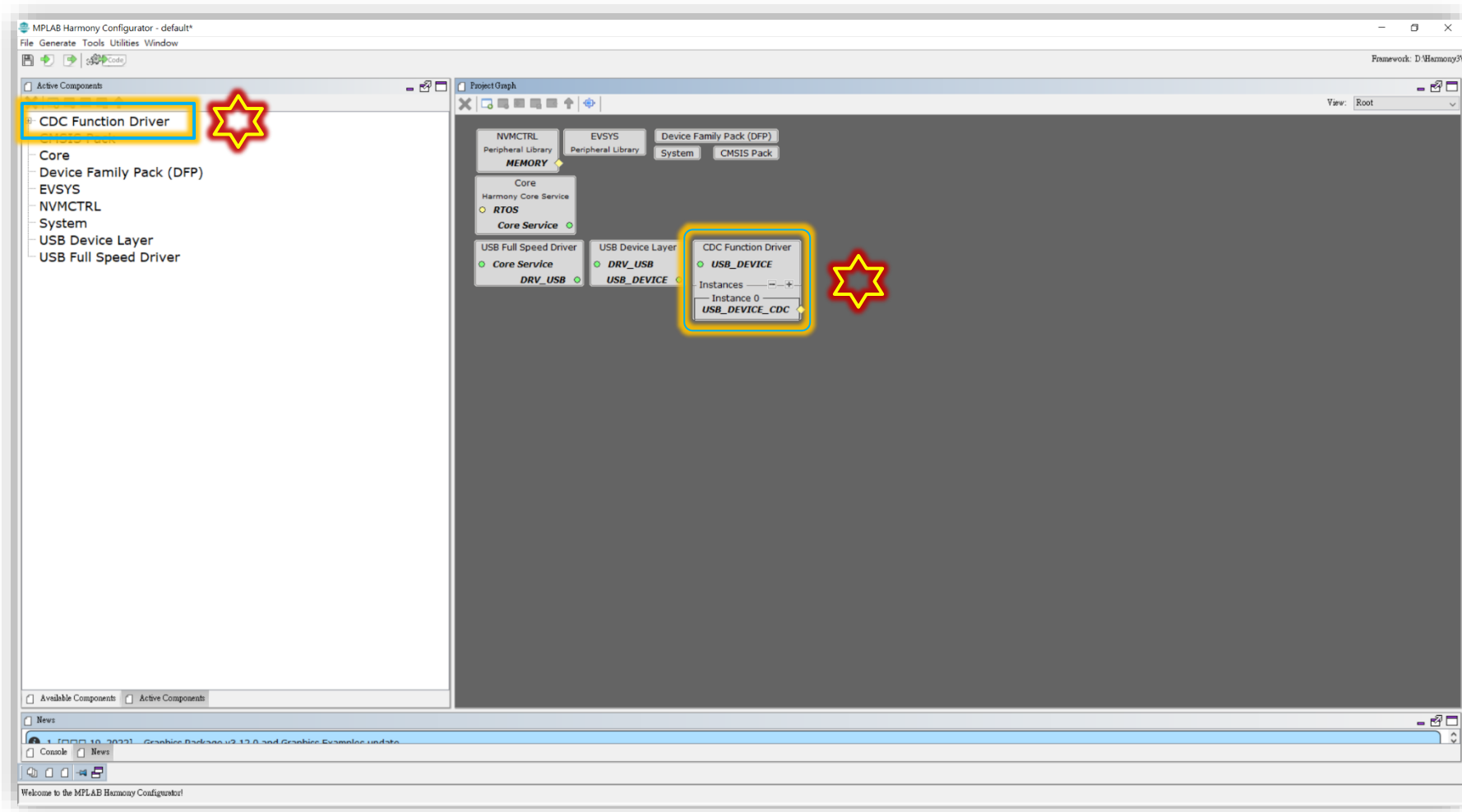
Lab5 USB Device CDC Class - Basic TX and RX

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”

The screenshot displays the MPLAB Harmony Configurator interface. On the left, the 'Available Components' list includes Audio, Bluetooth, Board Support Packages (BSPs), Bootloader, Drivers, Graphics, Harmony, Input, Libraries, Peripherals, SmartEnergy, System Services, TCP/IP, Third Party Libraries, Tools, Touch, Touch Turnkey, USB, Wireless, and X2C. The 'Project Graph' in the center shows a block diagram with components like NVMCTRL, EVSYS, Device Family Pack (DFP), Core Service, USB Full Speed Driver, USB Device Layer, and USB_DEVICE. The 'USB Device Layer' is highlighted with a yellow box. On the right, the 'Configuration Options' panel shows settings for the USB Device Layer, including Number of Functions (1), Number of Endpoints (2), Special Events, Special Features, Endpoint 0 Buffer Size (64), Vendor ID (0x04D8), Product ID Selection (Enter Product ID), Product ID (0x0000), Manufacturer String (Microchip Technology Inc.), and Product String Selection (Lab5_USBDevice_CDC_Basic_TX&RX). The 'Product String Selection' is highlighted with a yellow box and a red star. A yellow banner at the bottom right of the configurator reads 'Lab5_USBDevice_CDC_Basic_TX&RX'. The bottom of the window shows a 'News' section with various updates.

✂ Lab5 USB Device CDC Class - Basic TX RX

- Set LEDs and BTs in Pin Settings

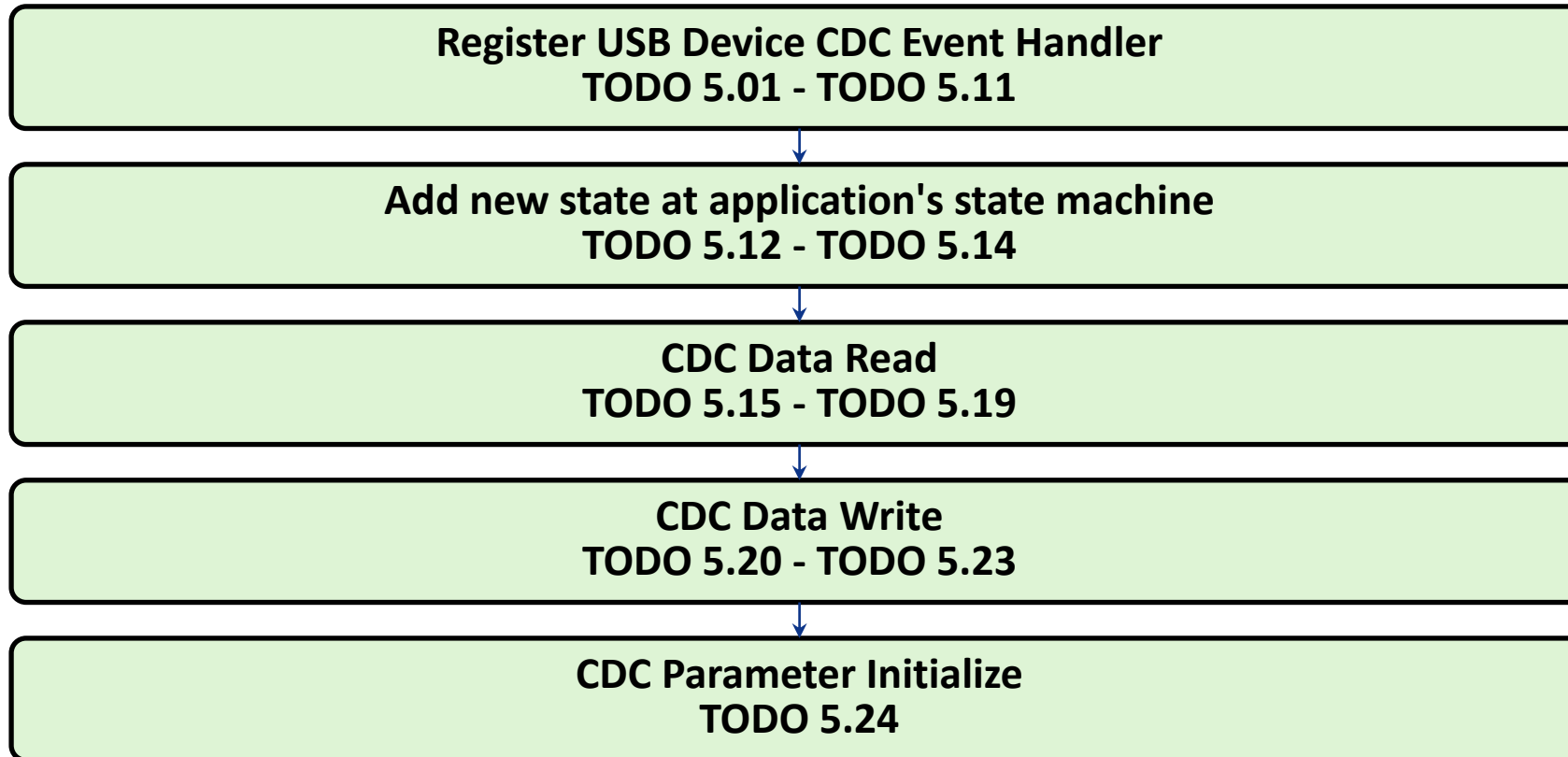
Pin Settings

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
16	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
17	VDDIO			Digital	High Impedance	Low	☐	☐	NORMAL
18	GNDIO			Digital	High Impedance	Low	☐	☐	NORMAL
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

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

```
// 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;

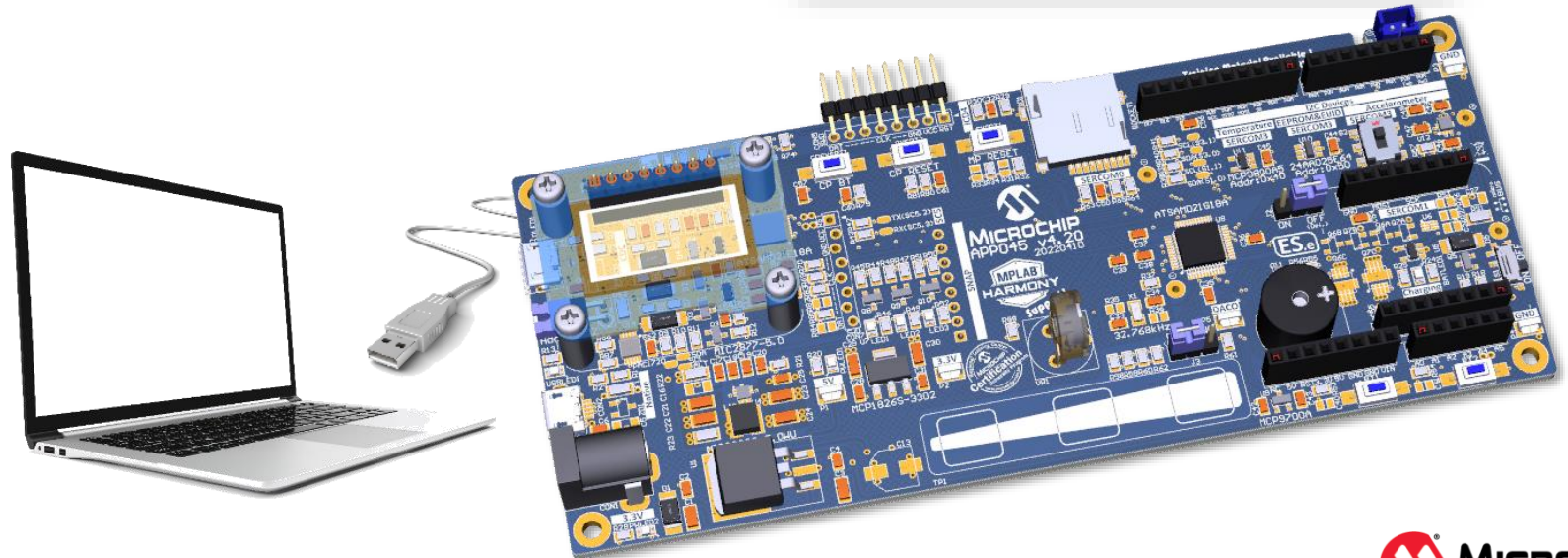
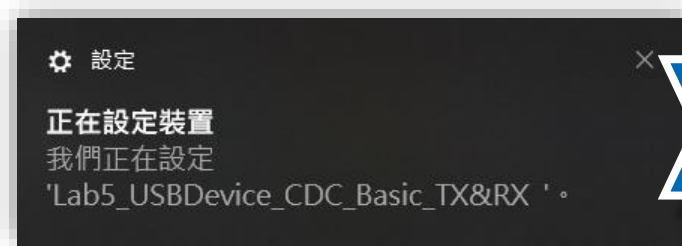
        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            break;

        ...
        return USB_DEVICE_EVENT_RESPONSE_NONE;
    }
}
```

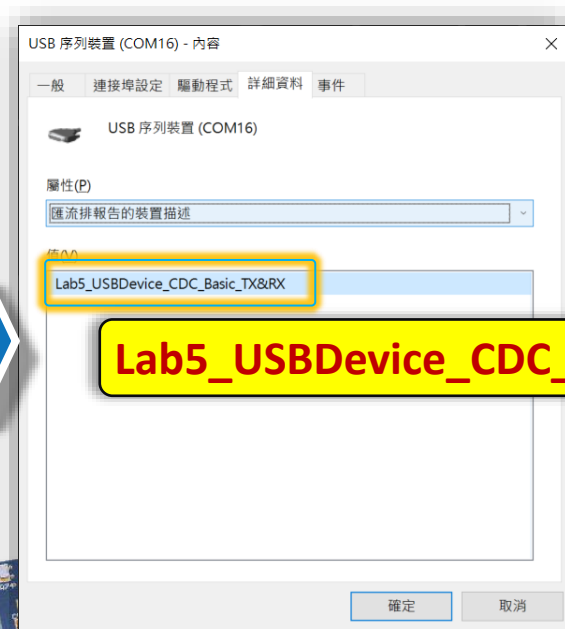
✂ Lab5 USB Device CDC Class - Basic TX RX

- 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”



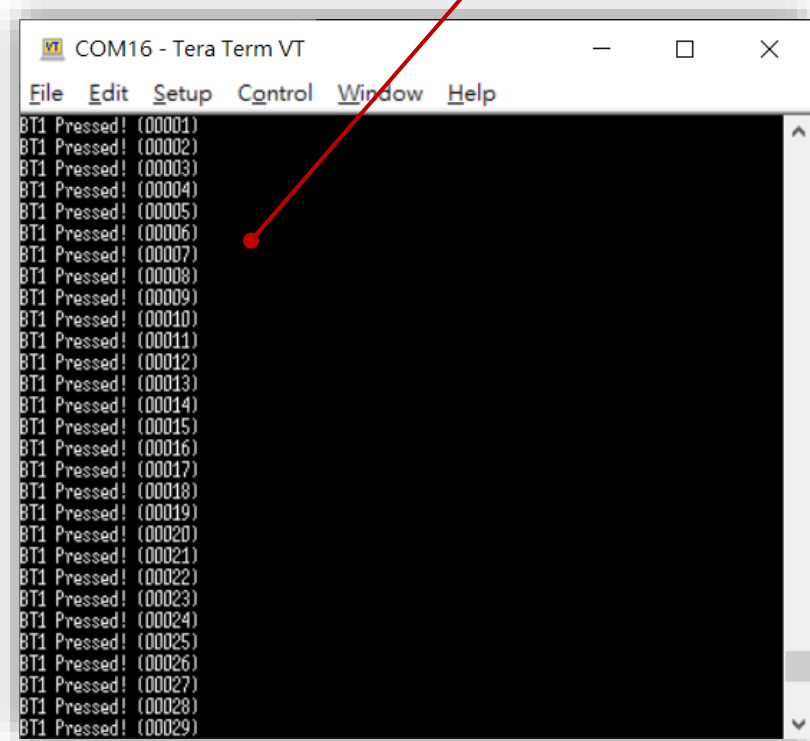
04D8,0000

Lab5_USBDevice_CDC_Basic_TX&RX

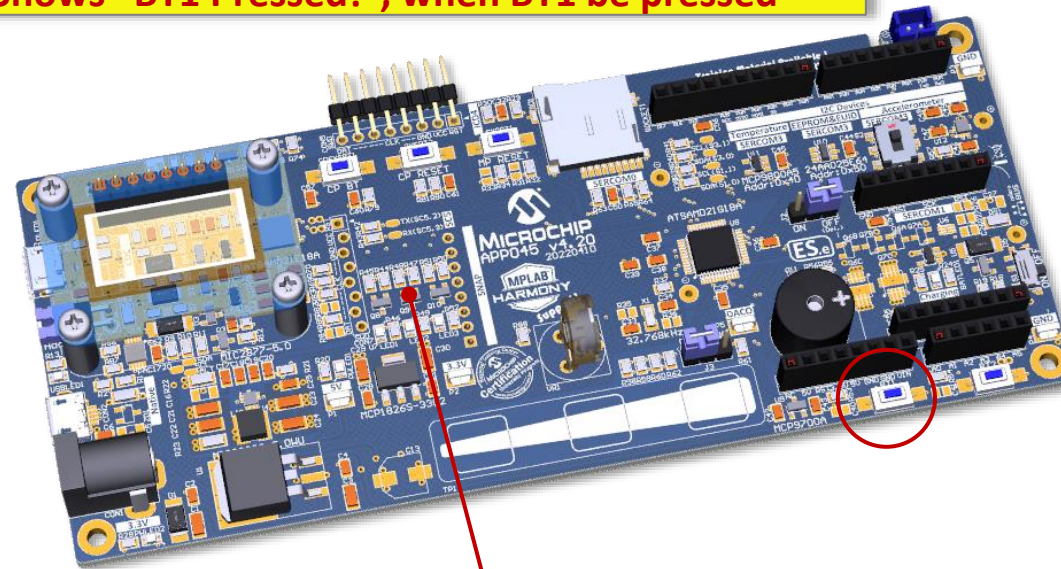


⚡ Lab5 USB Device CDC Class - Basic TX RX

- 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.



Shows "BT1 Pressed!", when BT1 be pressed



LED1~3 toggle when key-in '1','2' and '3'

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
    ...
}
```

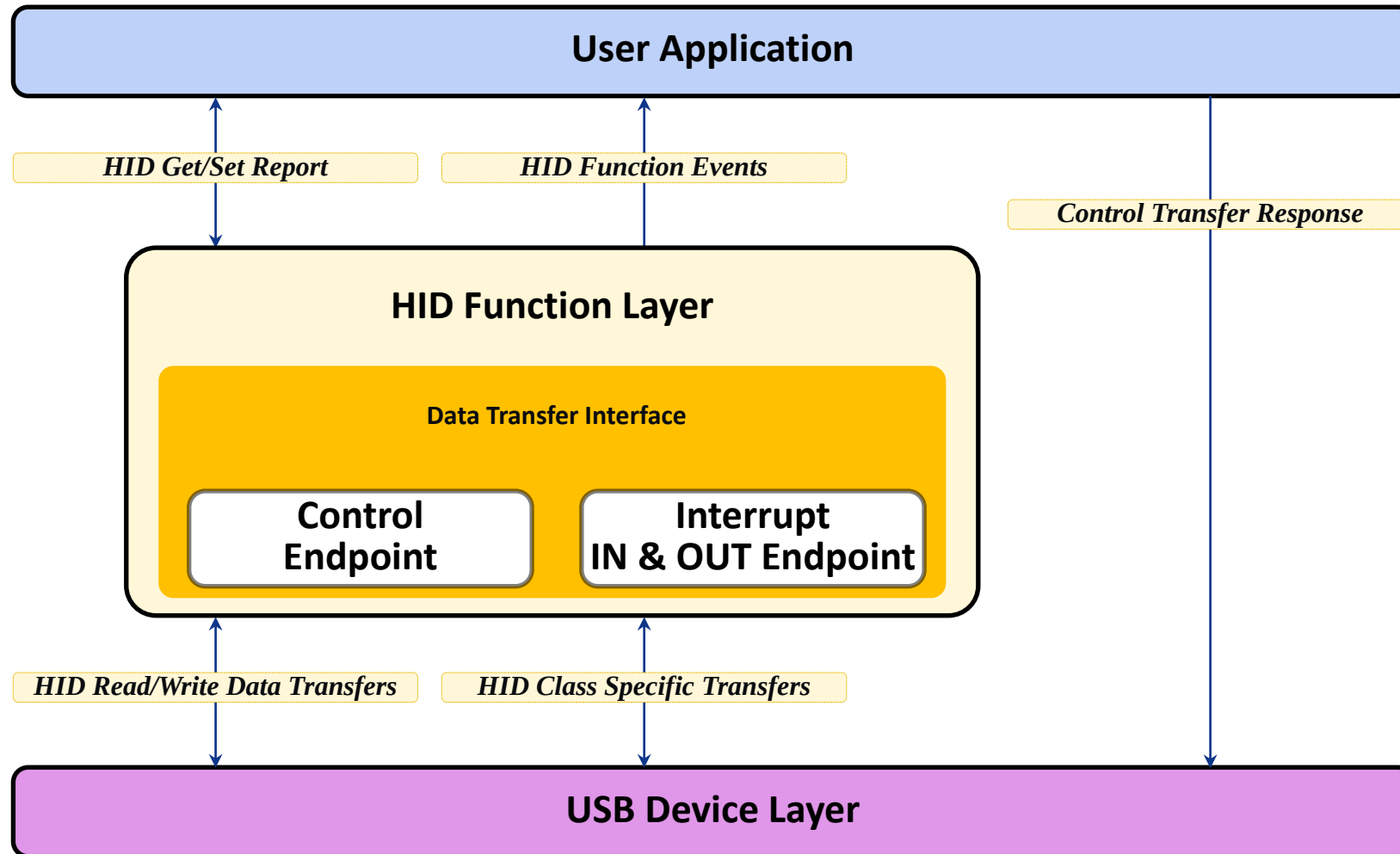
HID Usage Tables

- 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

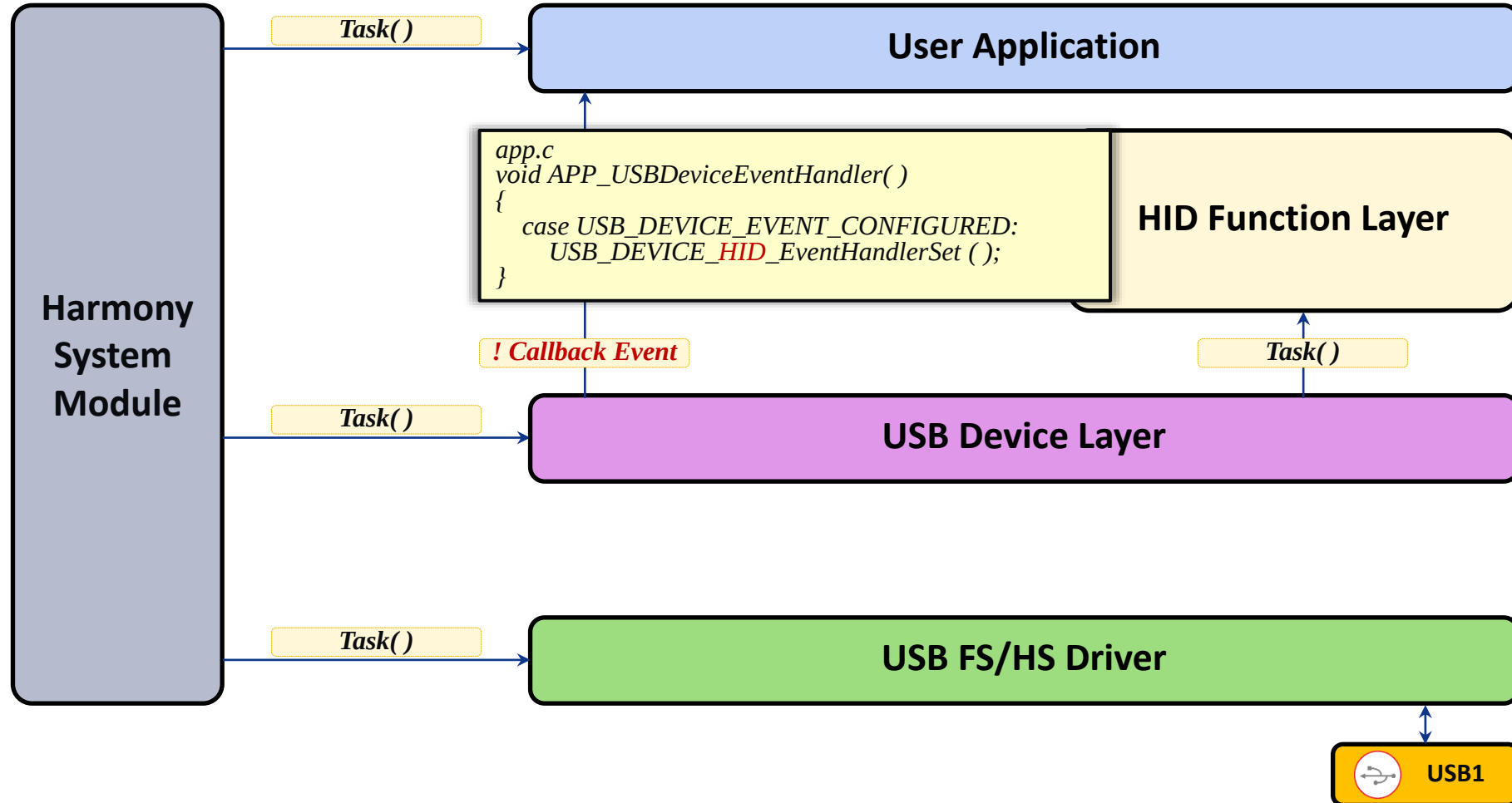
```
usb_device_init_data.c
const uint8_t hid_rpt0[] =
{
    0x06, 0x00, 0xFF, // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,      // Usage (Vendor Usage 1)
    0xA1, 0x01, // Colsslection (Application)
    0x19, 0x01, // Usage Minimum
    0x29, 0x40, // Usage Maximum //64 input usages 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

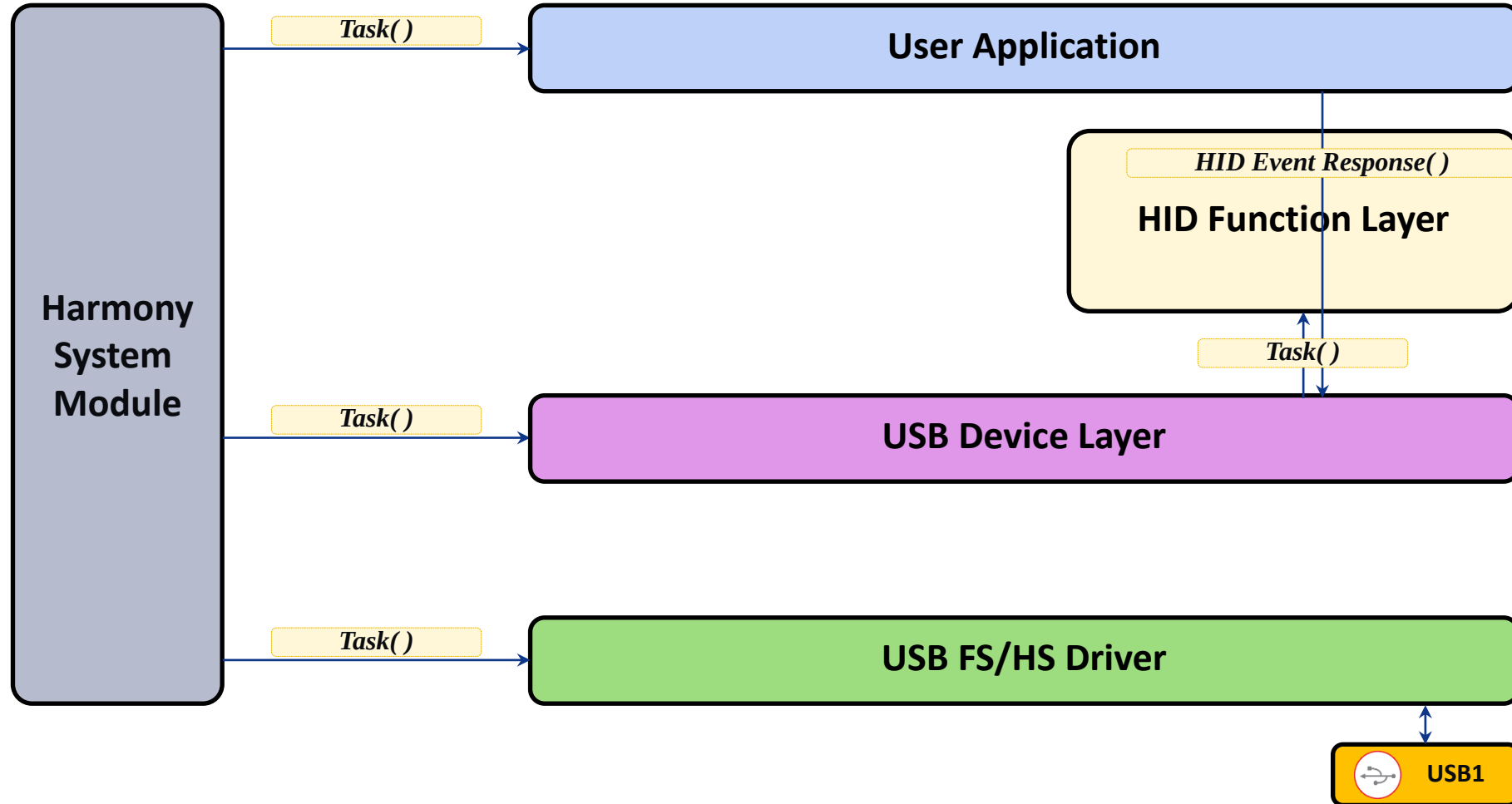


USB Stack Abstraction Model - Register Function Event Handler

- 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`

`_USB_DEVICE_HID_GlobalInitialize( )`
`_USB_DEVICE_HID_InitializeByDescriptorType( )`
`_USB_DEVICE_HID_ControlTransferHandler`
`_USB_DEVICE_HID_DeInitialize()`
`_USB_DEVICE_HID_ReportReceiveCallBack( )`
`_USB_DEVICE_HID_ReportSendCallBack( )`

`USB_DEVICE_HID_TransferCancel( )`

`USB_DEVICE_HID_EventHandlerSet( )`

`USB_DEVICE_HID_ReportReceive( )`

`USB_DEVICE_HID_ReportSend( )`

Called by the device layer during Set Configuration Processing.
Initialize the HID function driver by interface.
local function and should not be called directly by the application.

This callback forwards the event to application callback function.
This callback forwards the event to application callback function.

Cancels the HID transfer which has been submitted before.

Allows the application to register an event handler.

Receive a report from host.

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. :

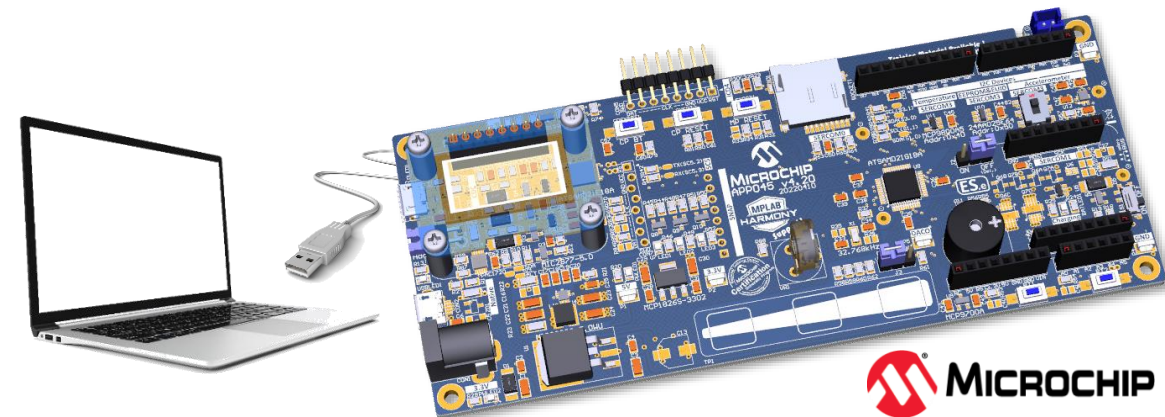
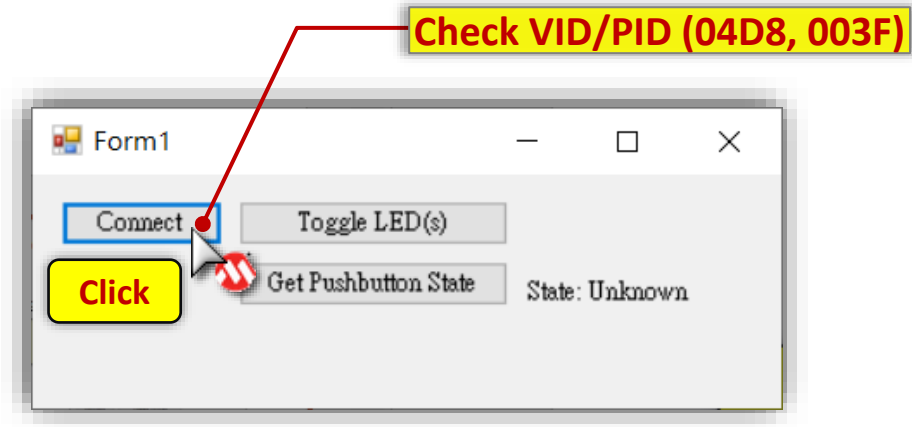
```
USB_DEVICE_HID_ReportSend  
(  
    USB_DEVICE_HID_INDEX_0,  
    &appData.writeTransferHandle,  
    appData.HIDtxDataBuffer, 64  
);
```

Lab6 USB Device HID Class - Basic TX and RX

Lab6 USB Device HID Class - Basic TX and RX

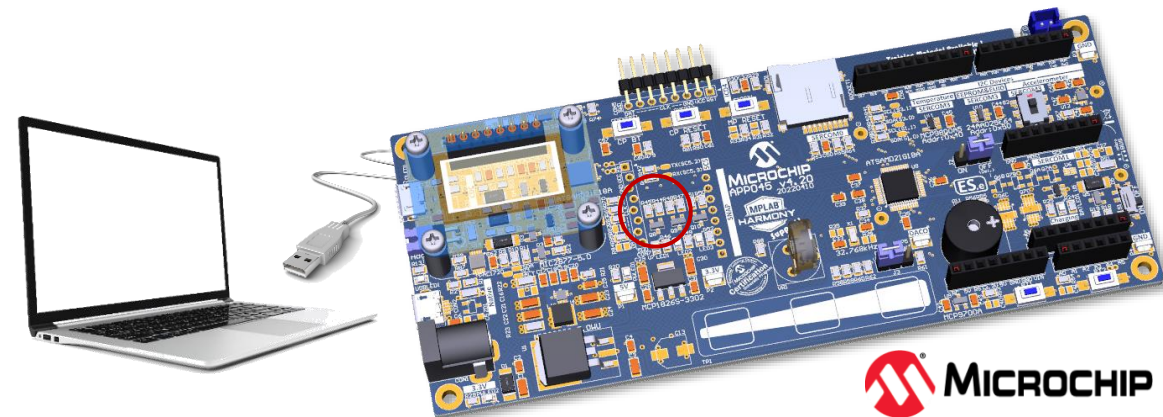
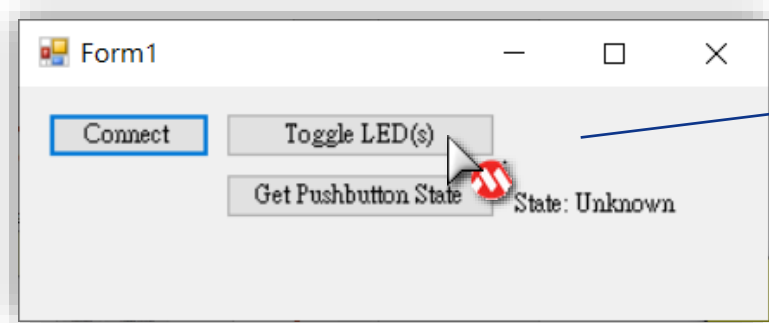
- At this lab, we try to add CDC function to our USB Device.
- The CDC function subclass is “No Subclass” (0x00) , protocol is “None” (0x00).
- Application require,
 - LED1 be toggle by ‘Toggle LED(s)’ GUI button at Generic HID Simple Demo.exe.
 - Show the BT1 status at GUI when click “Get pushbutton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

PC Application - Generic HID Simple Demo.exe

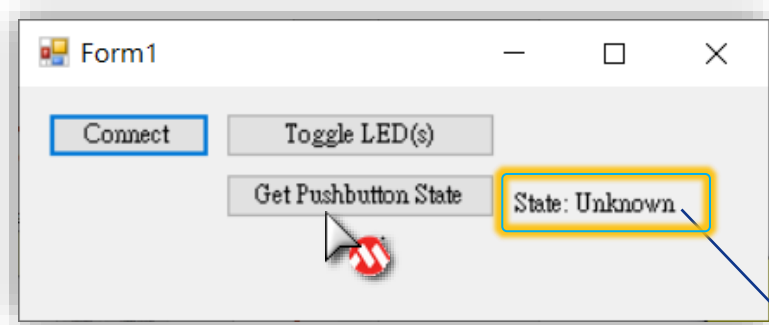


PC Application - Generic HID Simple Demo.exe

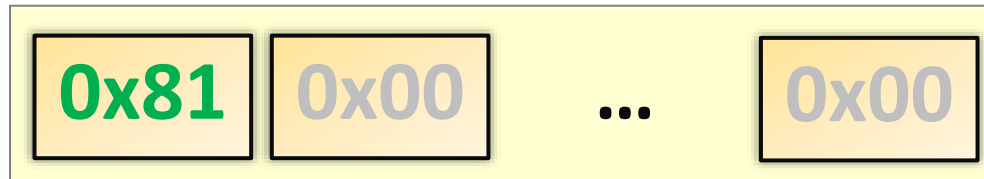
Toggle LED, (OUT Report)



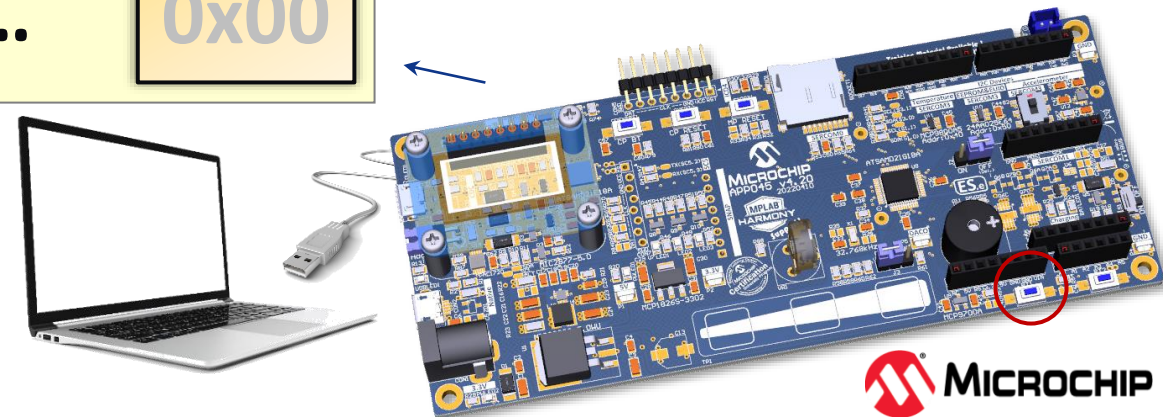
PC Application - Generic HID Simple Demo.exe



Get Push Button State (OUT Report)

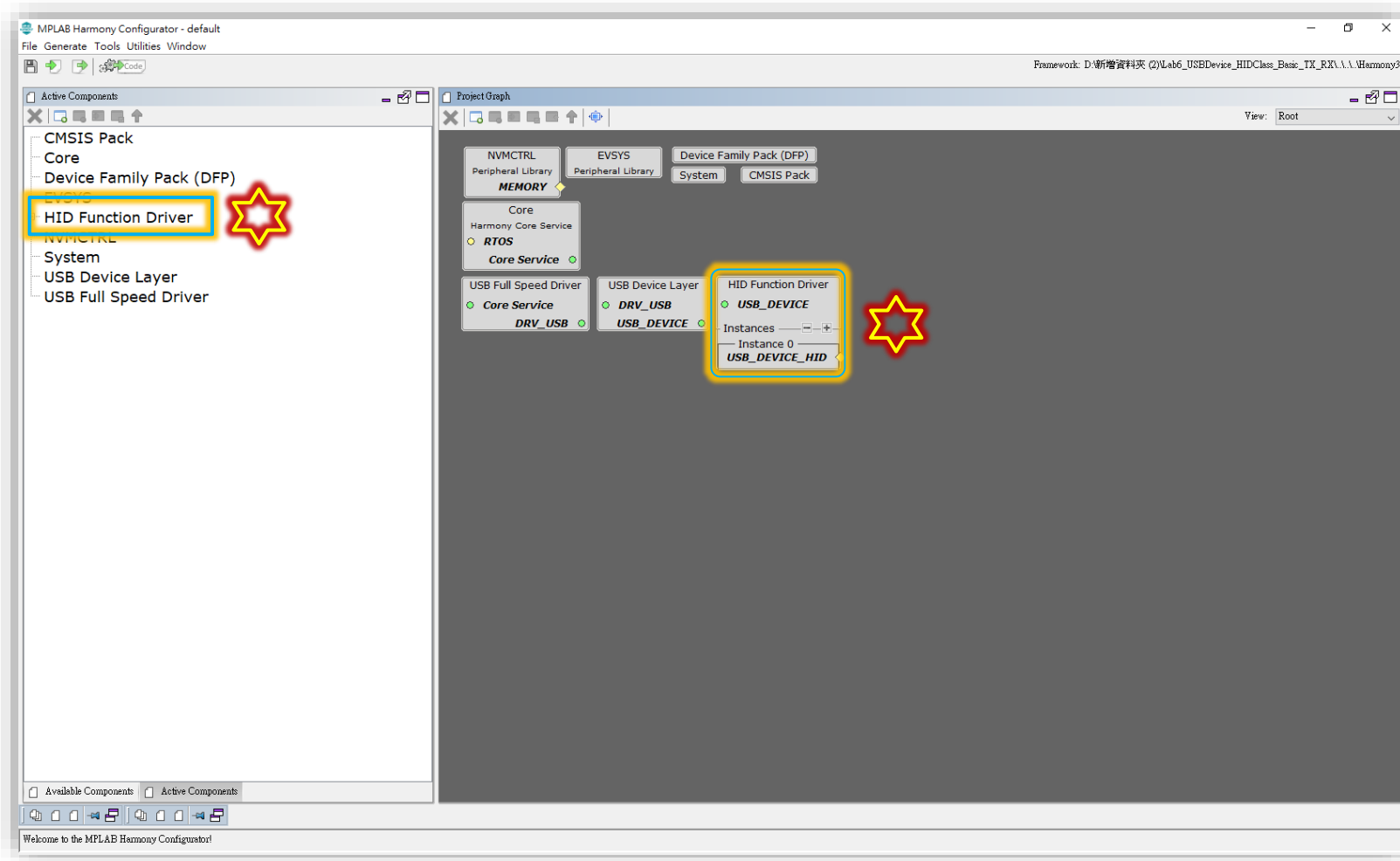


(IN Report)



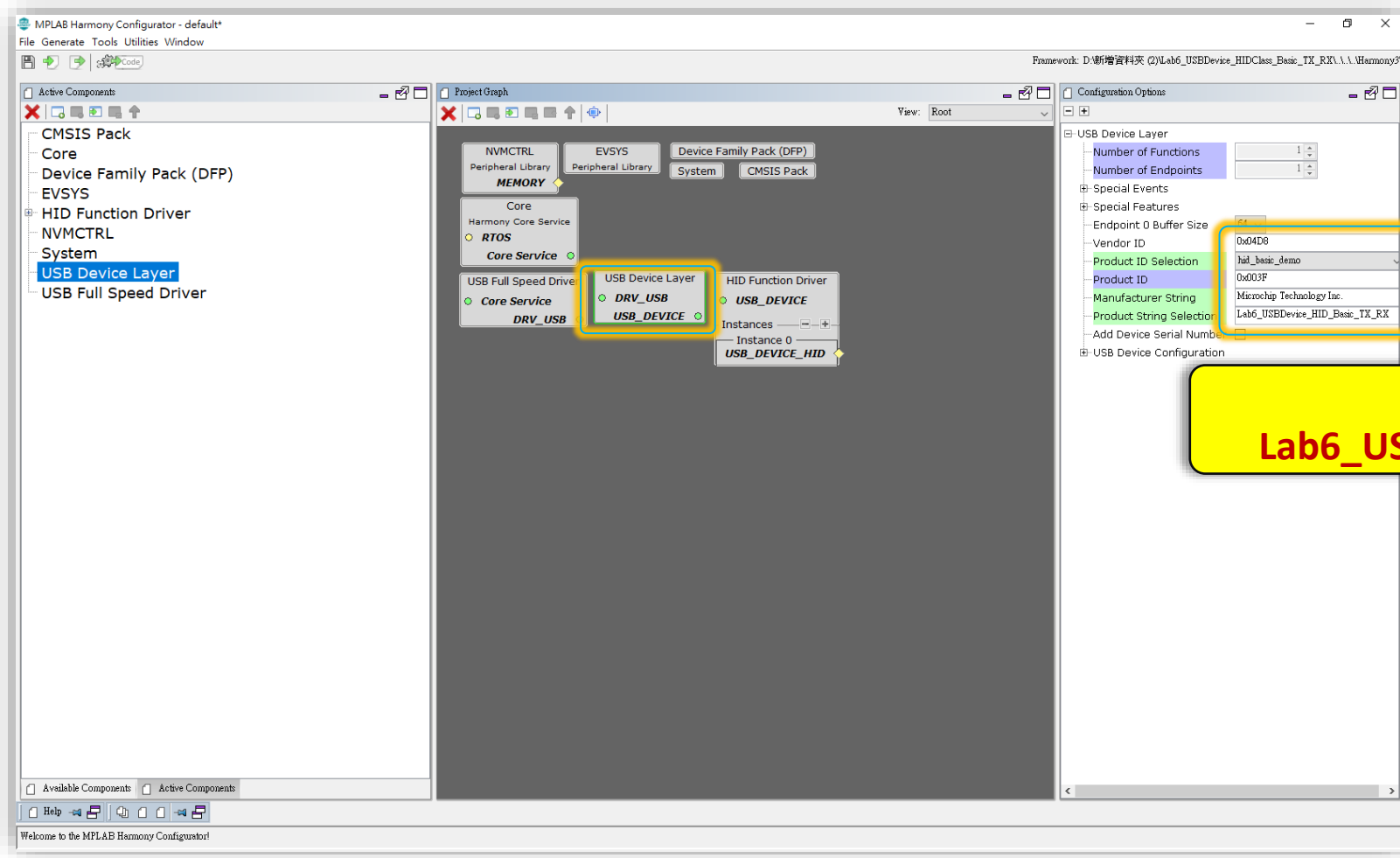
⚙️ Lab6 USB Device HID Class - Basic TX and RX

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



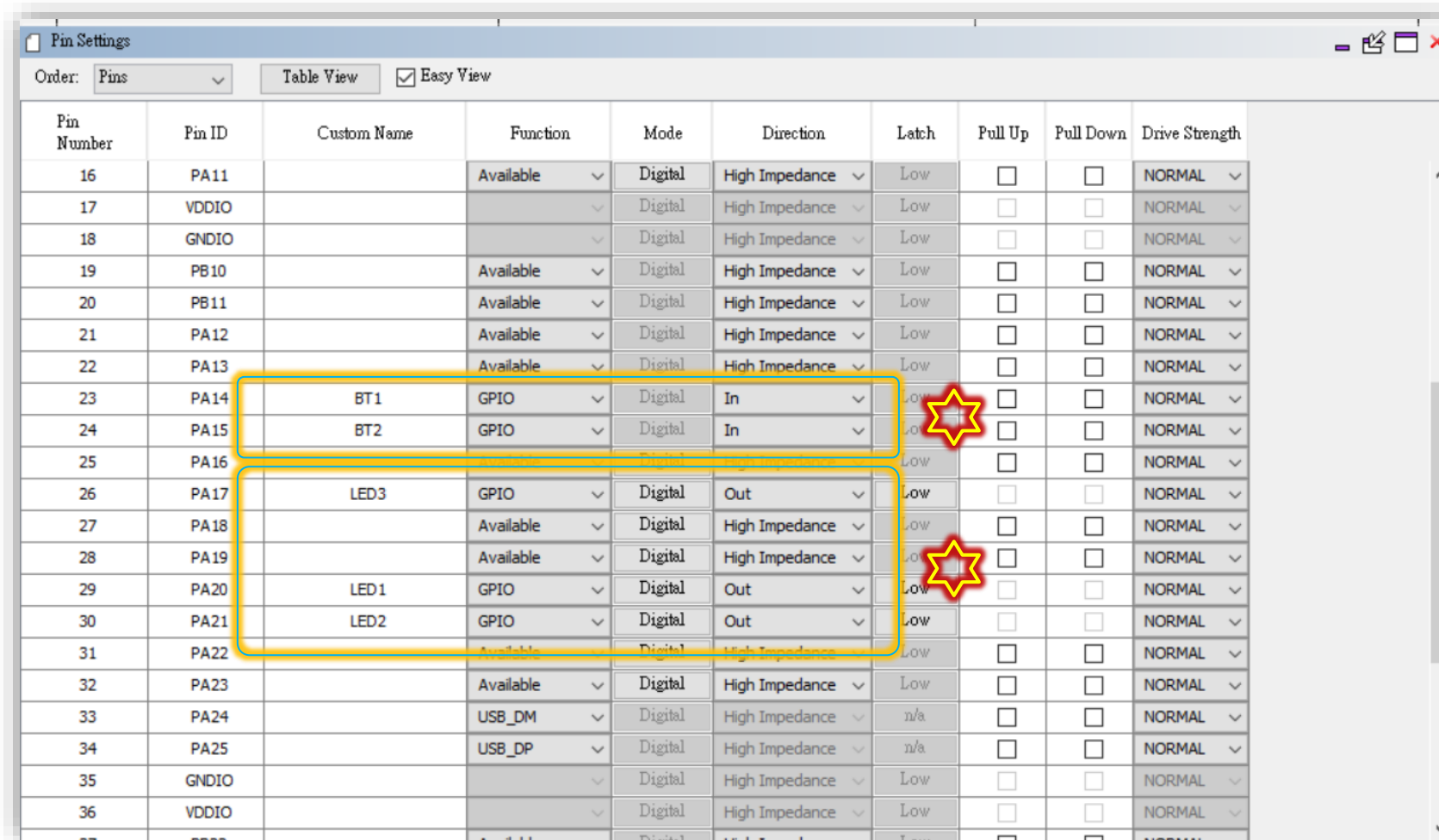
⚡ Lab6 USB Device HID Class - Basic TX and RX

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



✖ Lab6 USB Device HID Class - Basic TX and RX

- Set LEDs and BTs in Pin Settings



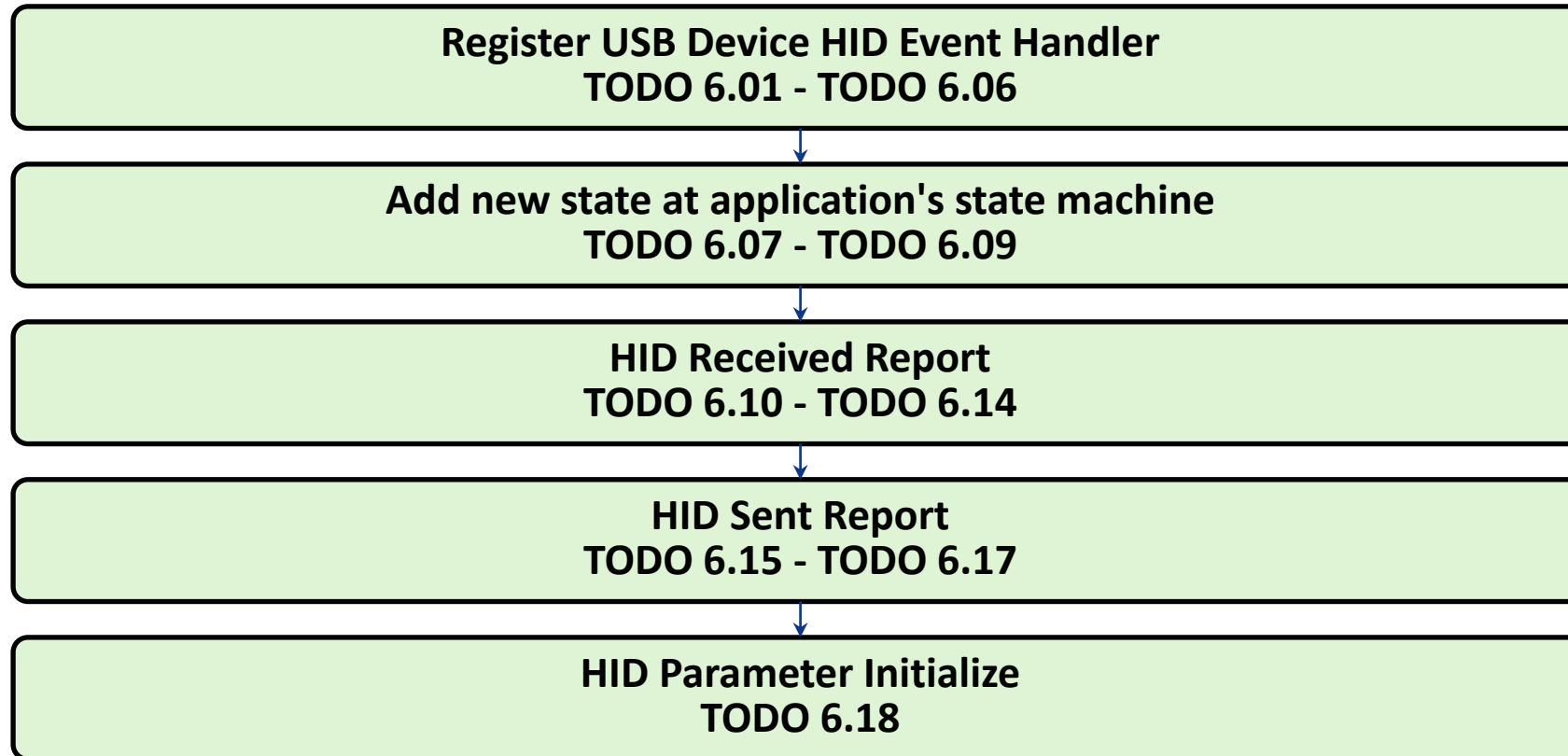
Pin Settings

Order: Pins Table View Easy View

Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
16	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
17	VDDIO			Digital	High Impedance	Low	☐	☐	NORMAL
18	GNDIO			Digital	High Impedance	Low	☐	☐	NORMAL
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

⚙️ 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;

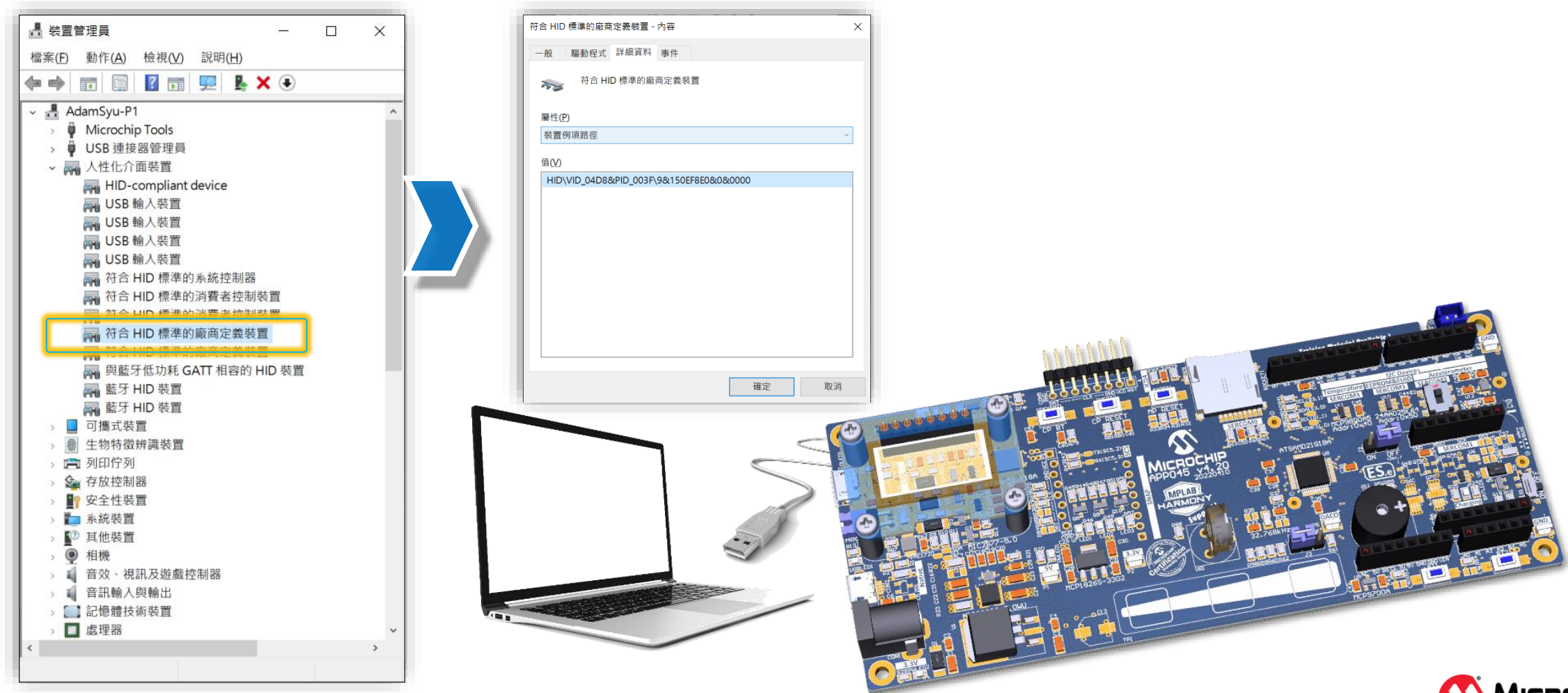
        case USB_DEVICE_EVENT_RESUMED:
            break;

        case USB_DEVICE_EVENT_CONTROL_TRANSFER_SETUP_REQUEST:
            break;

        ...
    }
    return USB_DEVICE_EVENT_RESPONSE_NONE;
}
```

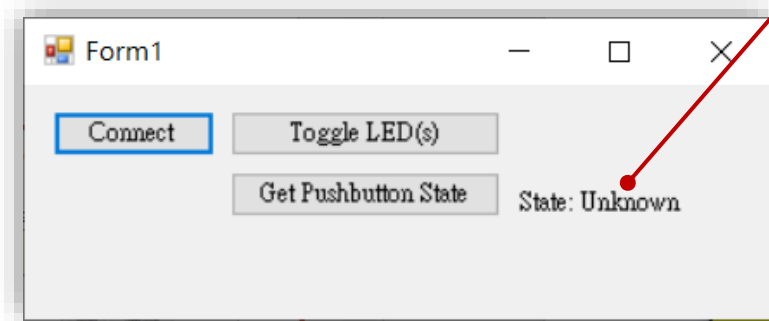
✂ Lab6 USB Device HID Class - Basic TX and RX

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

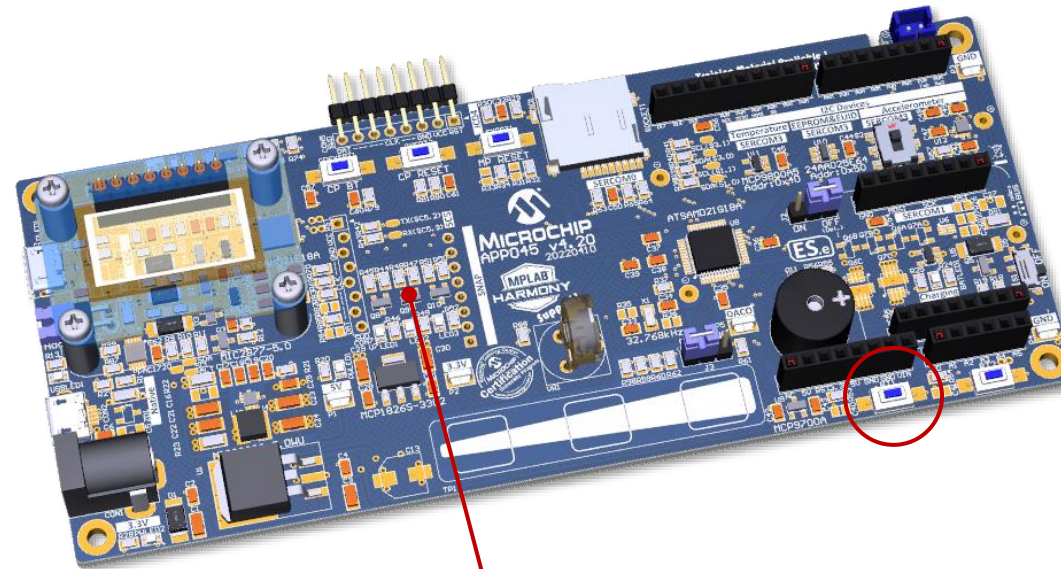


⚙️ Lab6 USB Device HID Class - Basic TX and RX

- 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:

AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, 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:

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