SAMD21 ARM Cortex-M0+ Microcontroller & Harmony v3

SAM2002ADV v1.00

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

CAE Taiwan Team
Microchip Technology Inc.

February 2023

SAM series course history and brief

- **SAM1001** **(2017~2018)**
  - Peripheral Library (PLIB) on APP045v2.0 (SAMD21 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 (SAMD21 Arm Cortex-M0+)
  - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- **SAM2001ADV** **(2021~2022)**
  - Peripheral Library (PLIB) Advanced modules on APP045v4.x (SAMD21 Arm Cortex-M0+)
  - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- **SAM2002** **(2019~2021)**
  - System Service and Driver on APP045v4.x (SAMD21 Arm Cortex-M0+)
  - MPLAB Harmony Framework v3.x + XC32 Compiler + MPLAB X IDE
- **SAM2002ADV** **(2022~)**
  - QTouch Library and USB Device Driver on APP045v4.x (SAMD21 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

CAE Taiwan Team
Microchip Technology Inc.

February 20, 2023

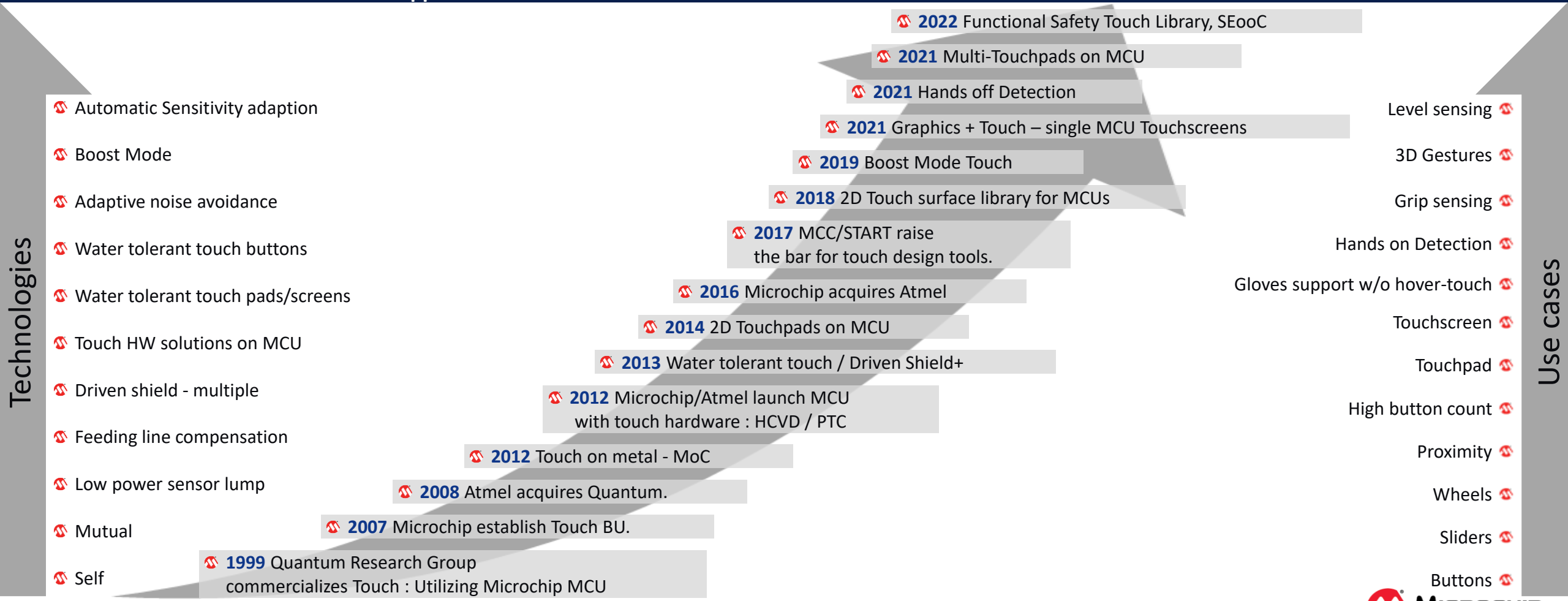
Introduction to QTouch® Peripheral Touch Controller (PTC)

Introduction to QTouch® Peripheral Touch Controller (PTC)

Long-Term innovative Player in Touch

Microchip serves **all Markets and all use cases** for touch.

Consumer | **Appliances** | **Medical** | **Industrial** | **Automotive** | **Wearables** | **IoT** | **Avionic**

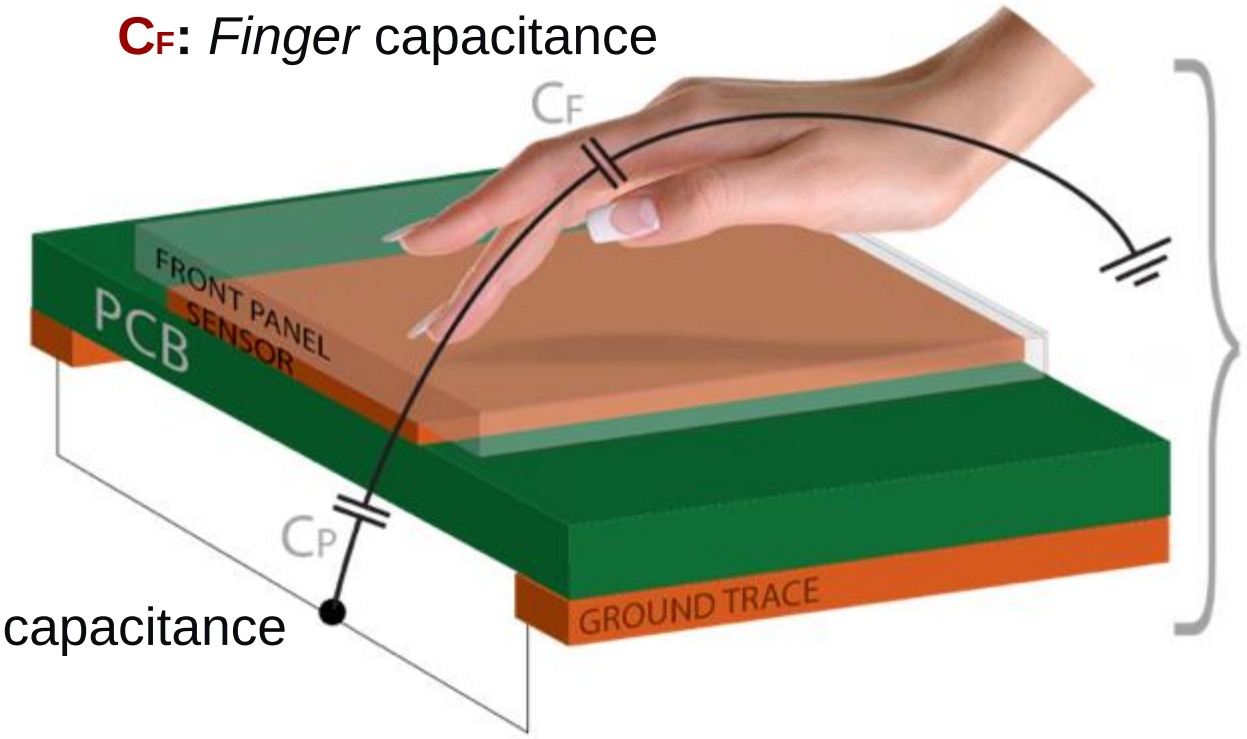


MICROCHIP

Introduction to QTouch® Peripheral Touch Controller (PTC)

Capacitive Touch Sensor

C_F: *Finger* capacitance



C_S: total *Sensor* capacitance

C_P: *Parasitic* capacitance

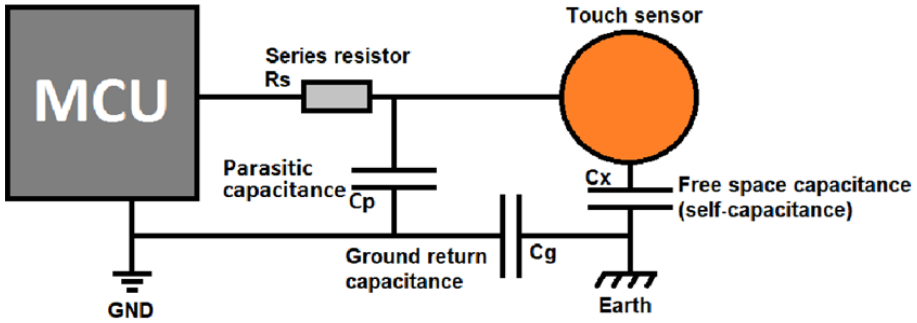
Sensor Capacitance (C_S) = C_P + C_F

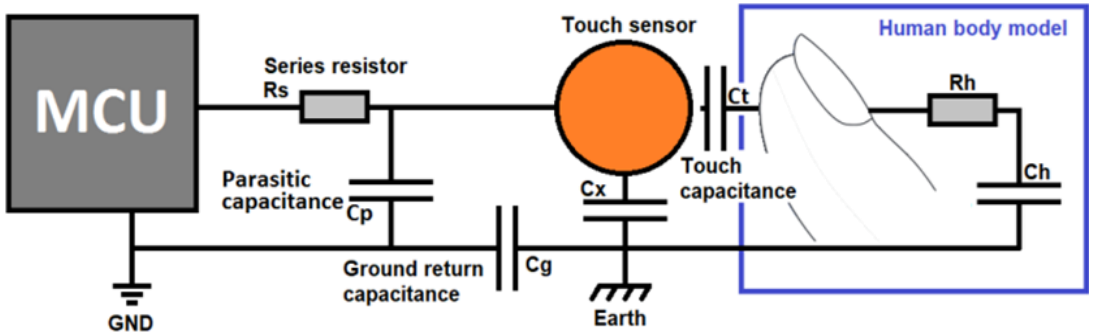
MICROCHIP

Introduction to QTouch® Peripheral Touch Controller (PTC)

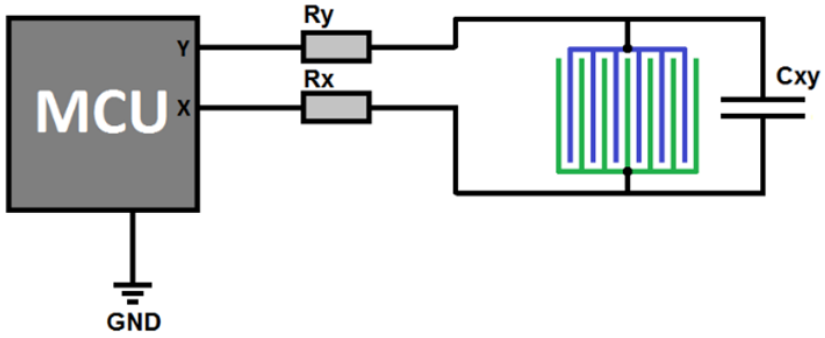
Capacitance Touch Measurement

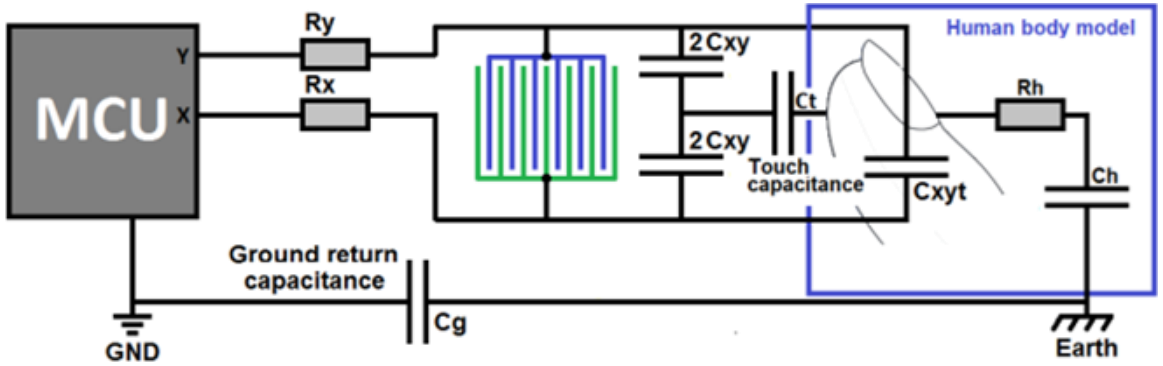
Self Capacitance Touch





Mutual Capacitance Touch



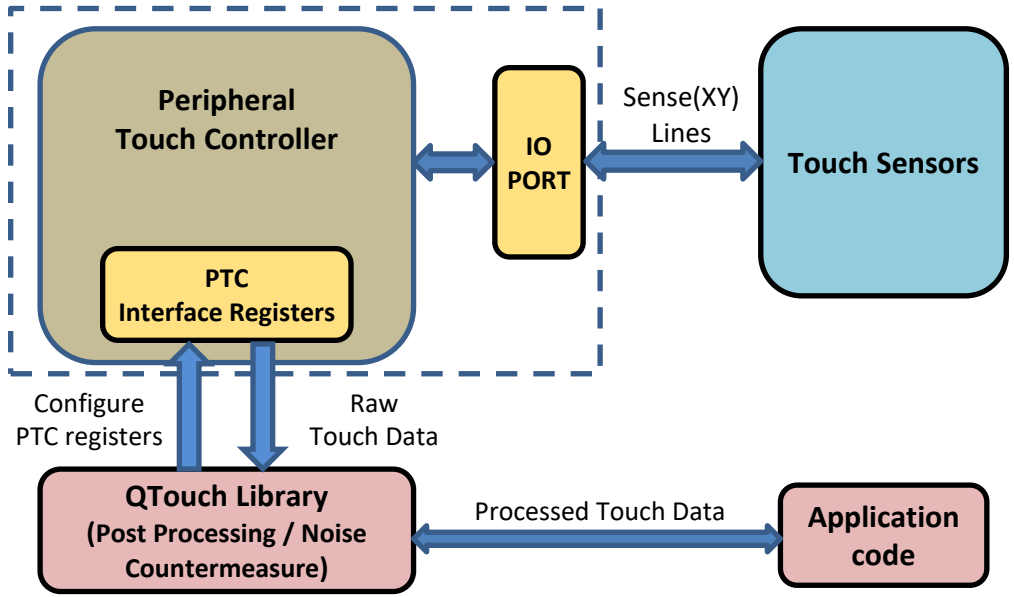


MICROCHIP

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.

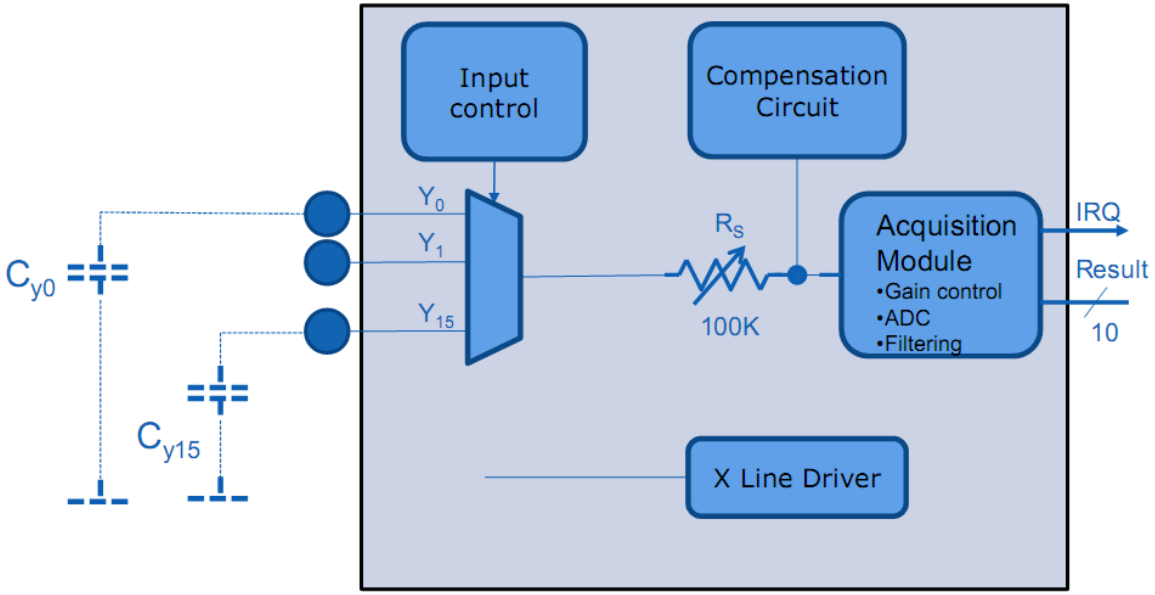


MICROCHIP

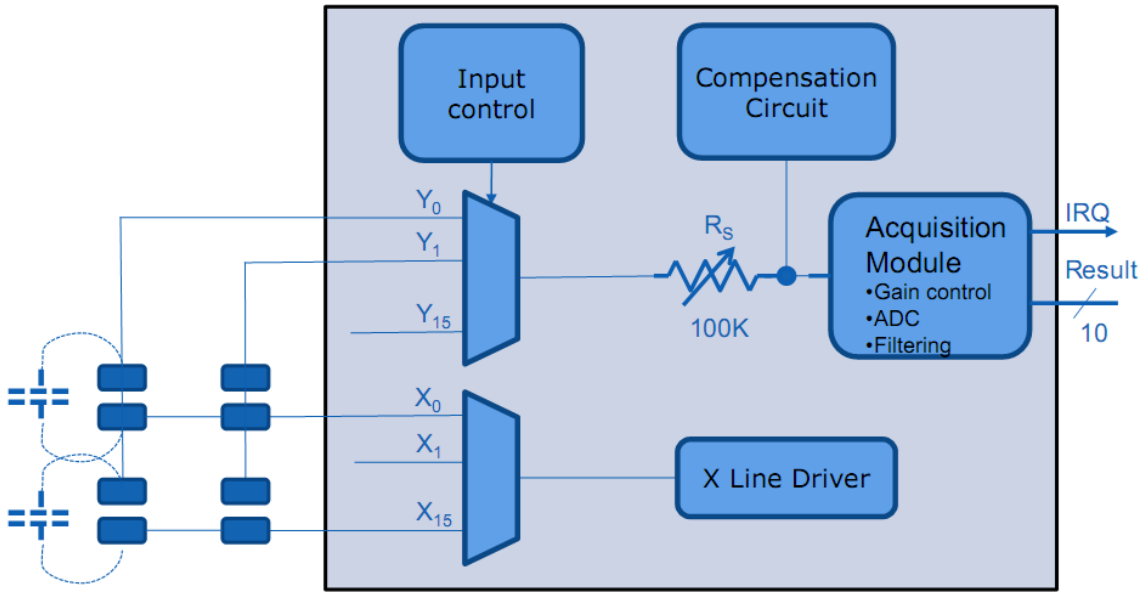
Introduction to QTouch® Peripheral Touch Controller (PTC)

PTC Block Diagram

Self Capacitance Touch

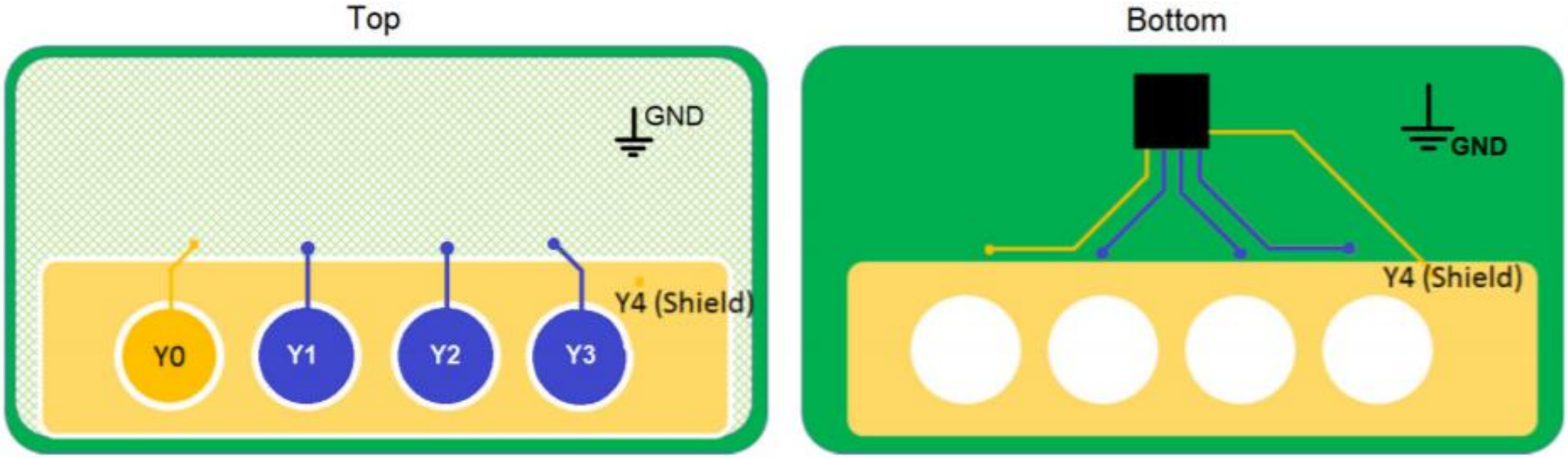


Mutual Capacitance Touch

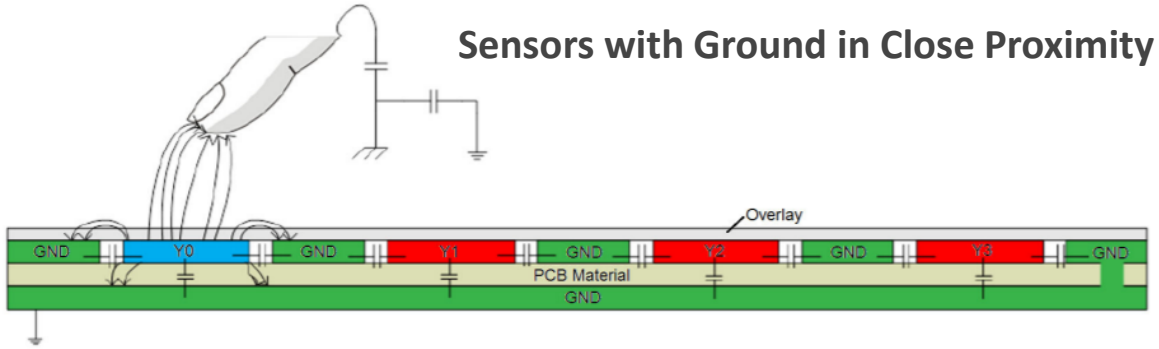


Introduction to QTouch® Peripheral Touch Controller (PTC)

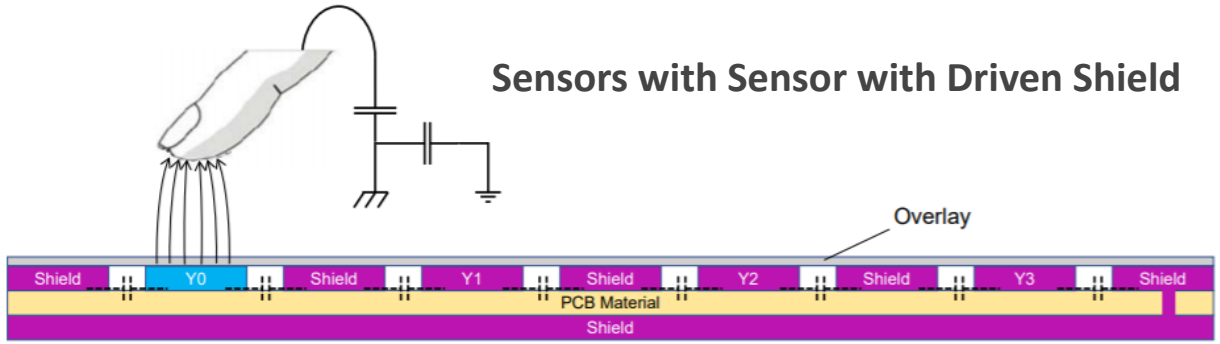
Driven Shield



Sensors with Ground in Close Proximity



Sensors with Sensor with Driven Shield



https://microchipdeveloper.com/touch:guide-for-ptc-driven-shield-design

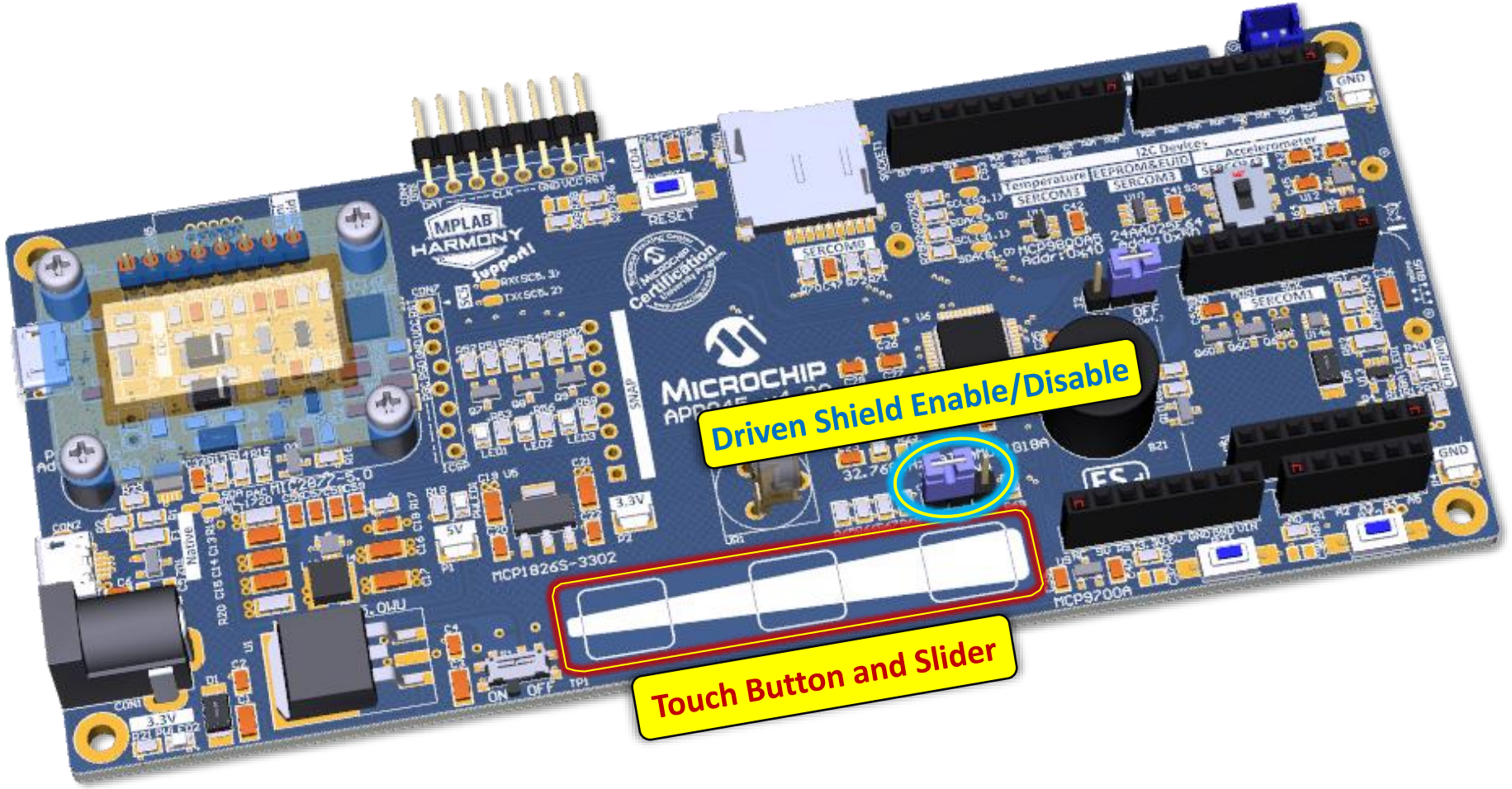
MICROCHIP

APP045 EVB QTouch® Design and Pattern

MICROCHIP

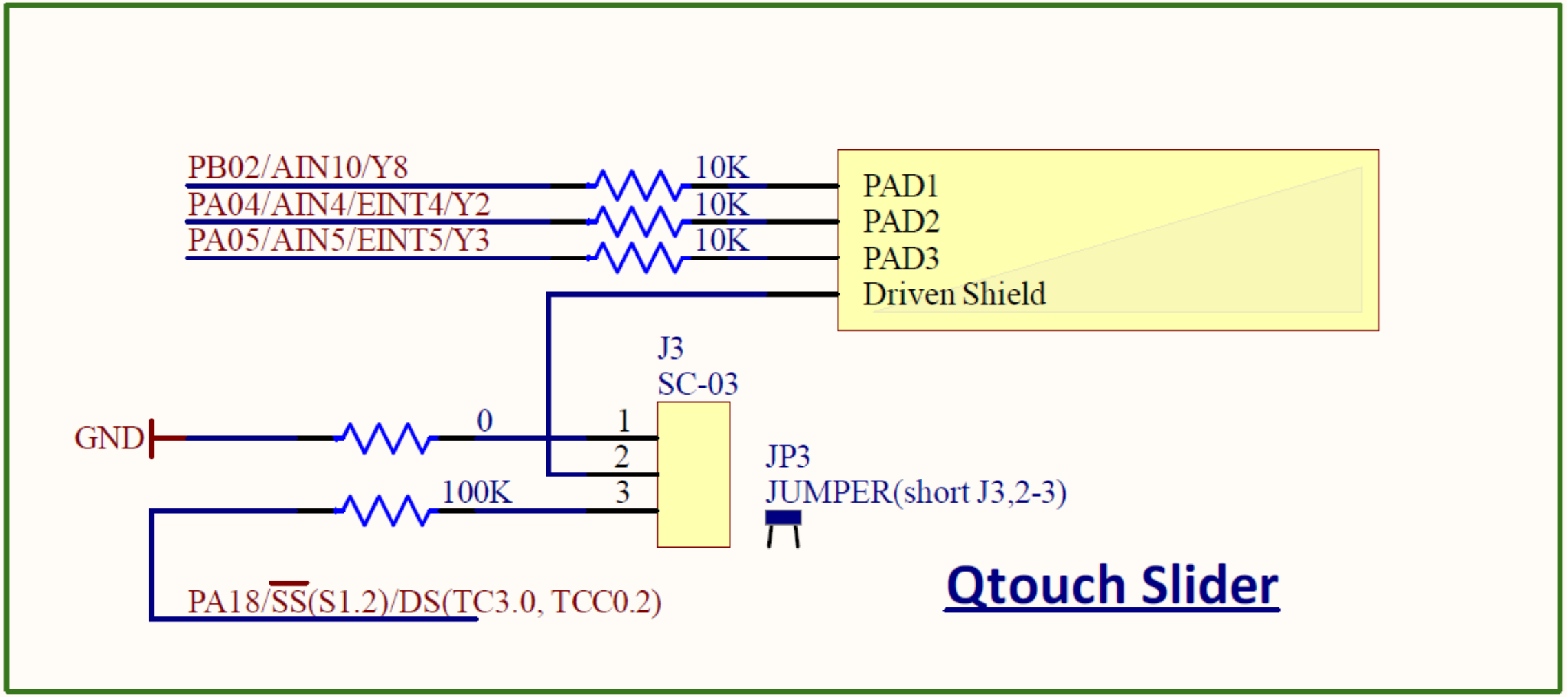
APP045 EVB QTouch® Design and Pattern

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



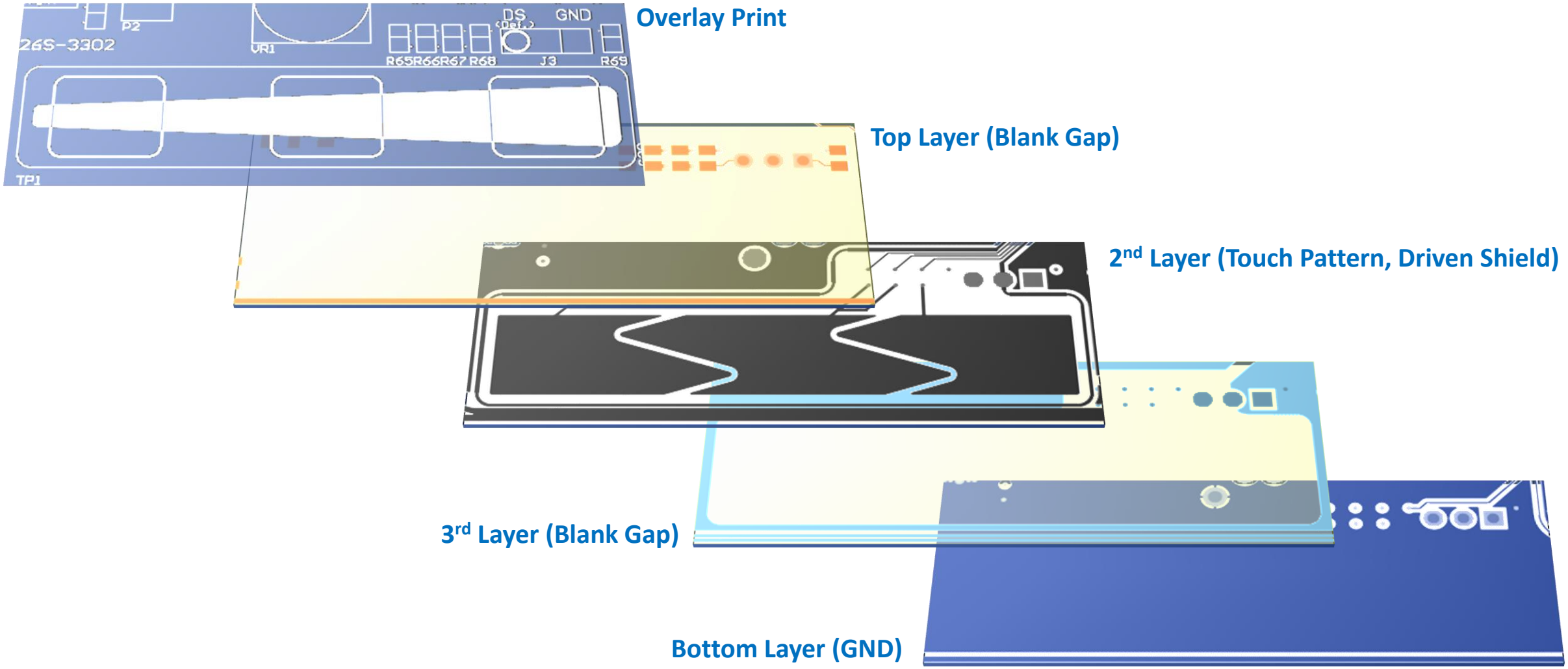
APP045 EVB QTouch® Design and Pattern

Schematic



APP045 EVB QTouch® Design and Pattern

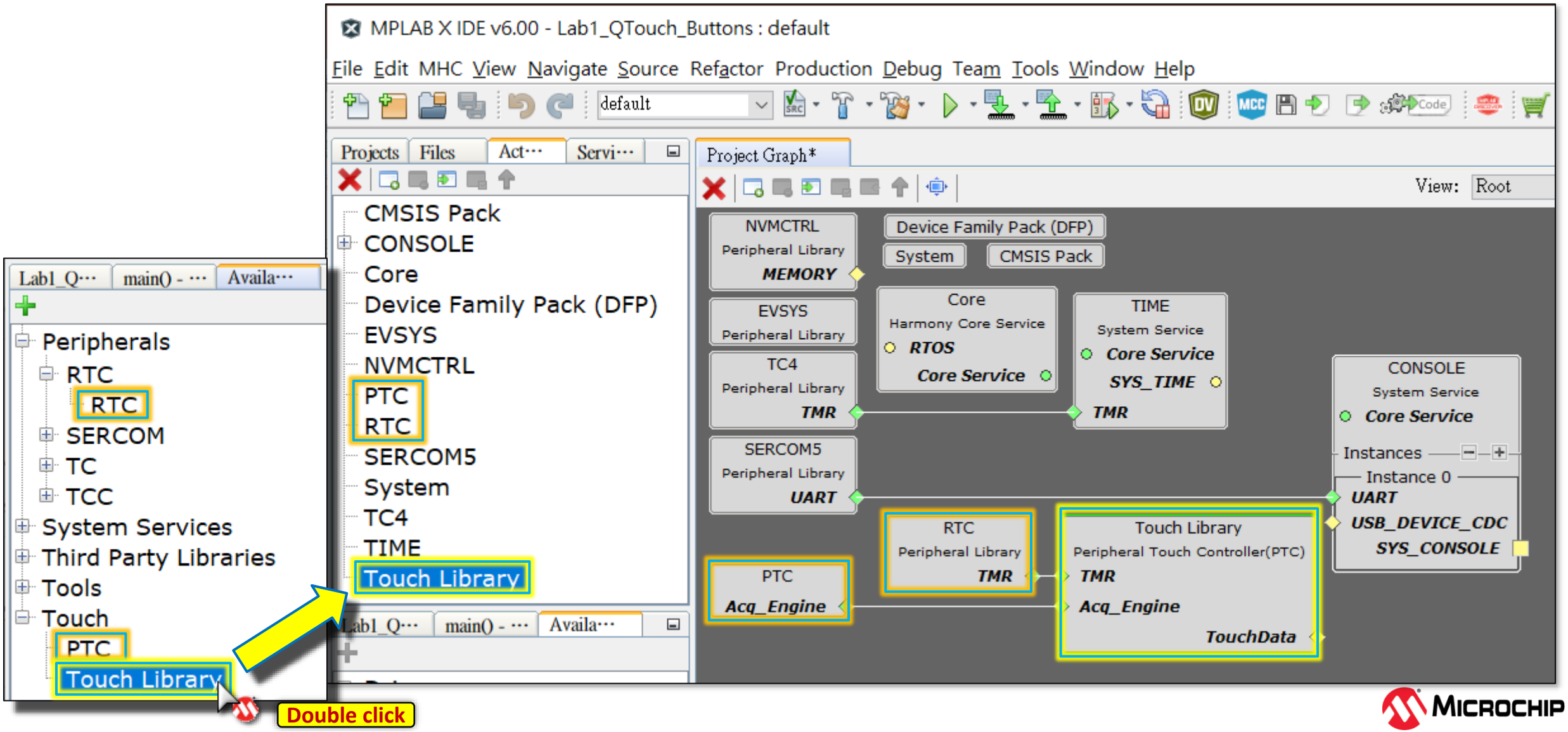
PCB Stack



Harmony Touch Configurator

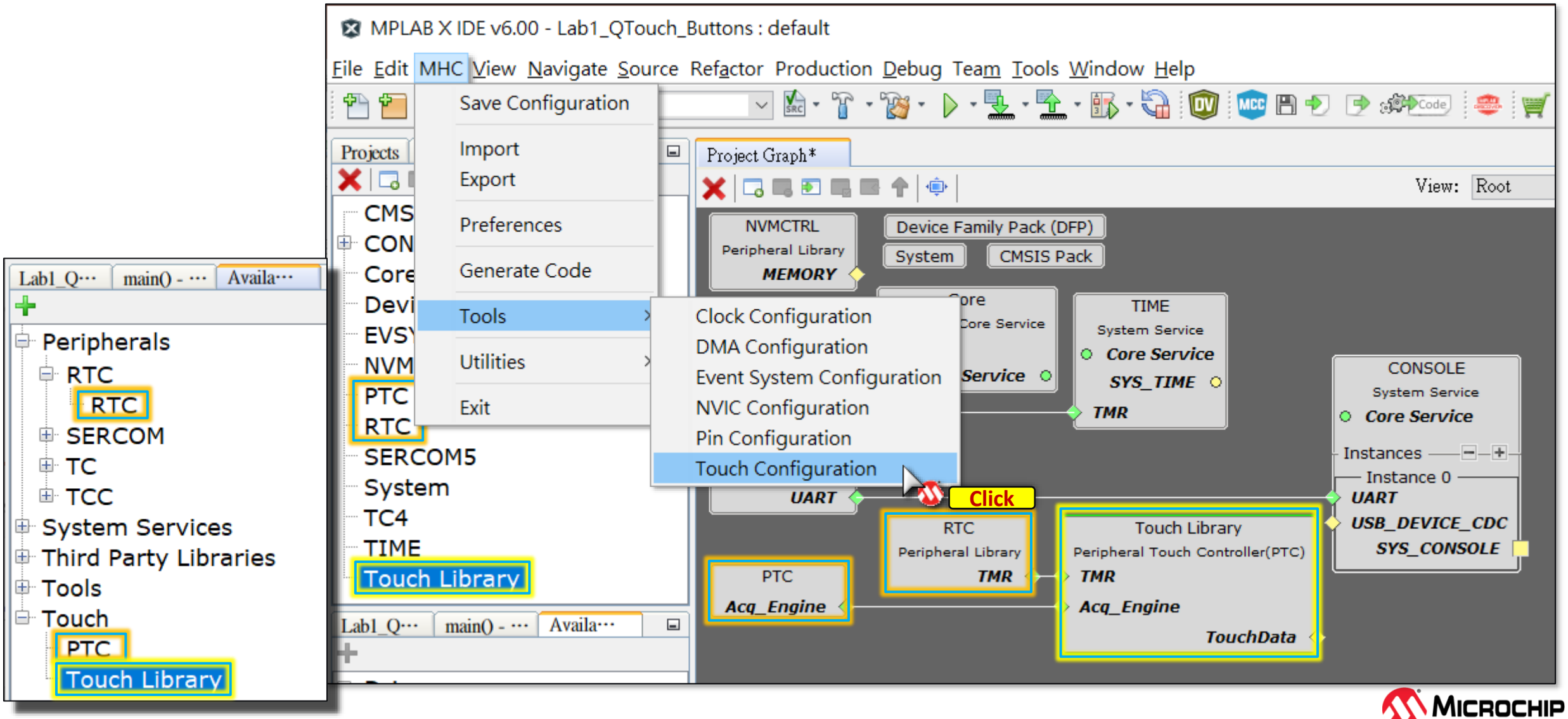
Harmony Touch Configurator

Add Touch Library in MHC



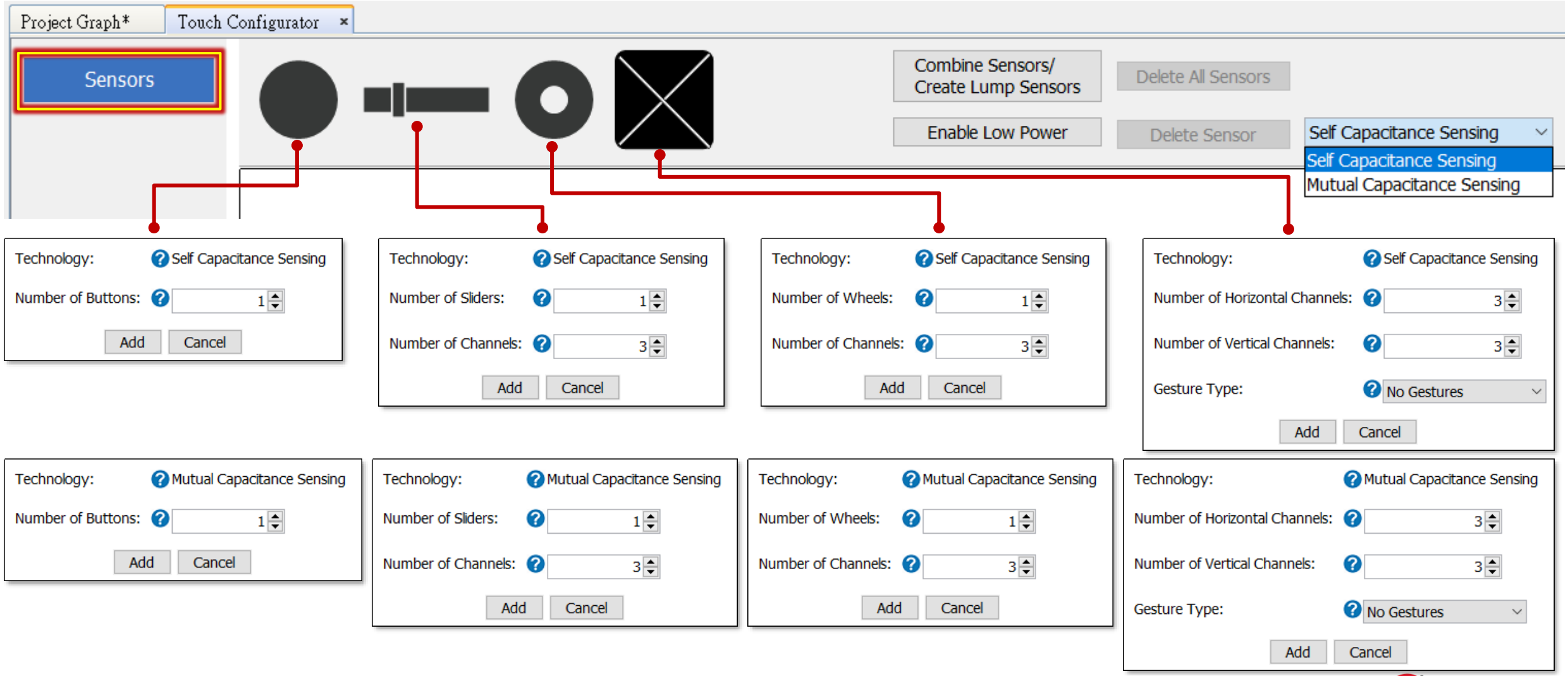
Harmony Touch Configurator

Select MHC\Tools\Touch Configuration



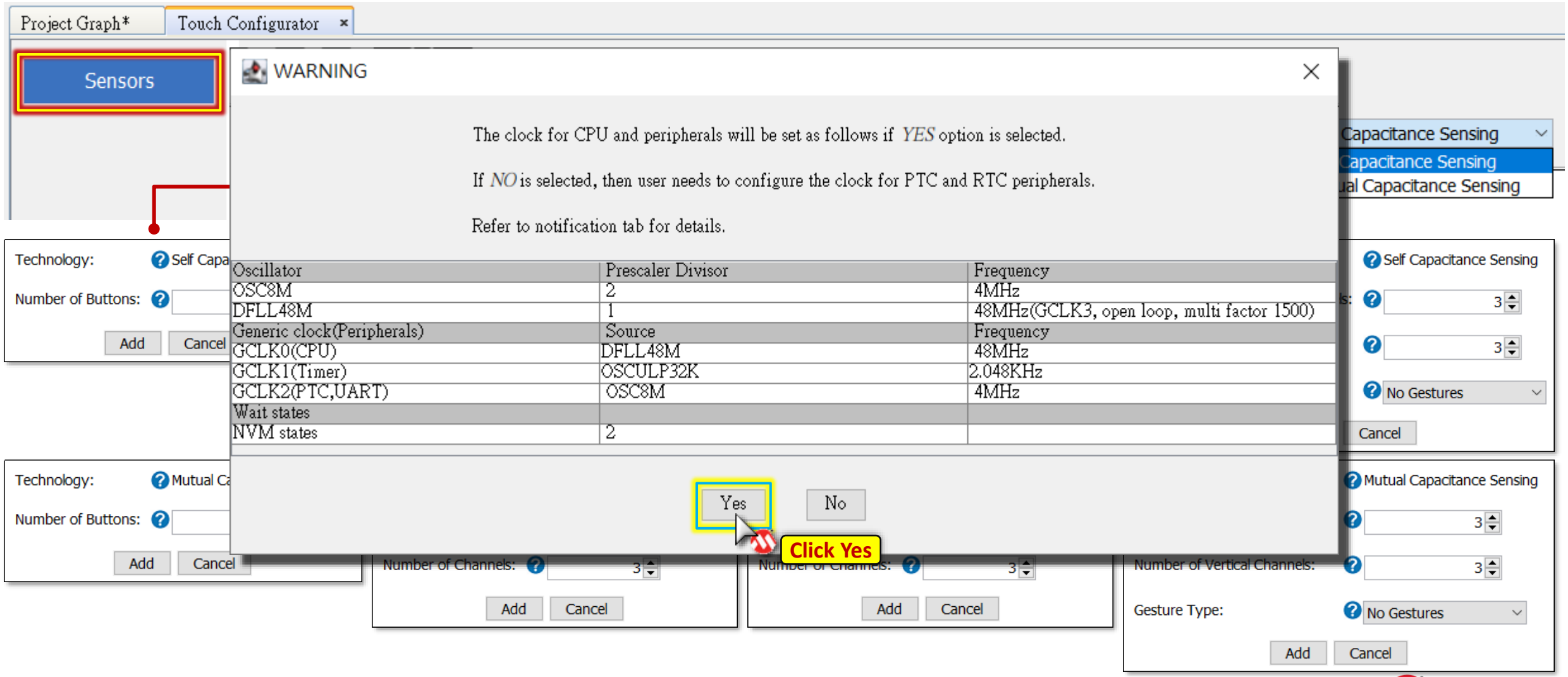
Harmony Touch Configurator

Sensors : Select Sensing Type and add Touch Sensors



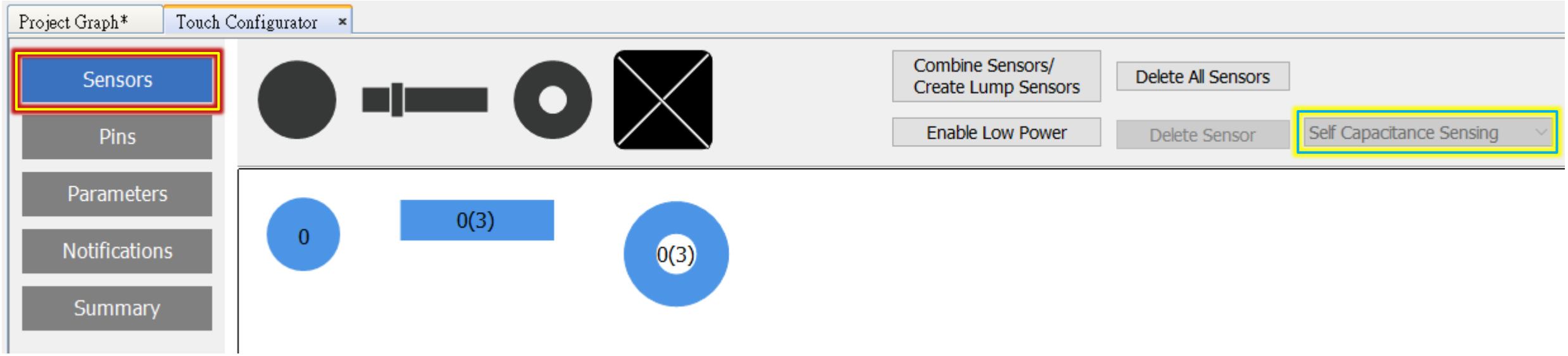
Harmony Touch Configurator

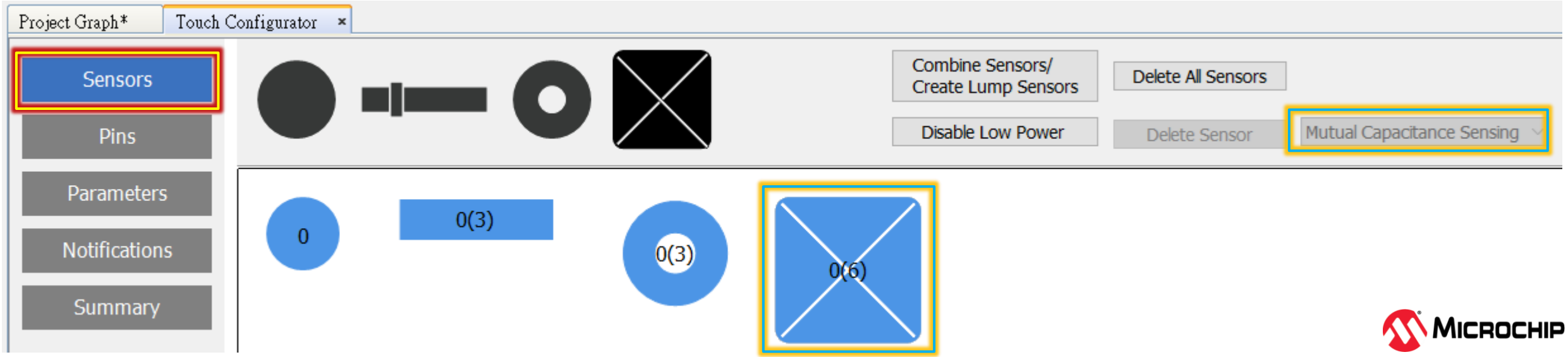
Sensors : Auto configure the clock for CPU and peripherals



Harmony Touch Configurator

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

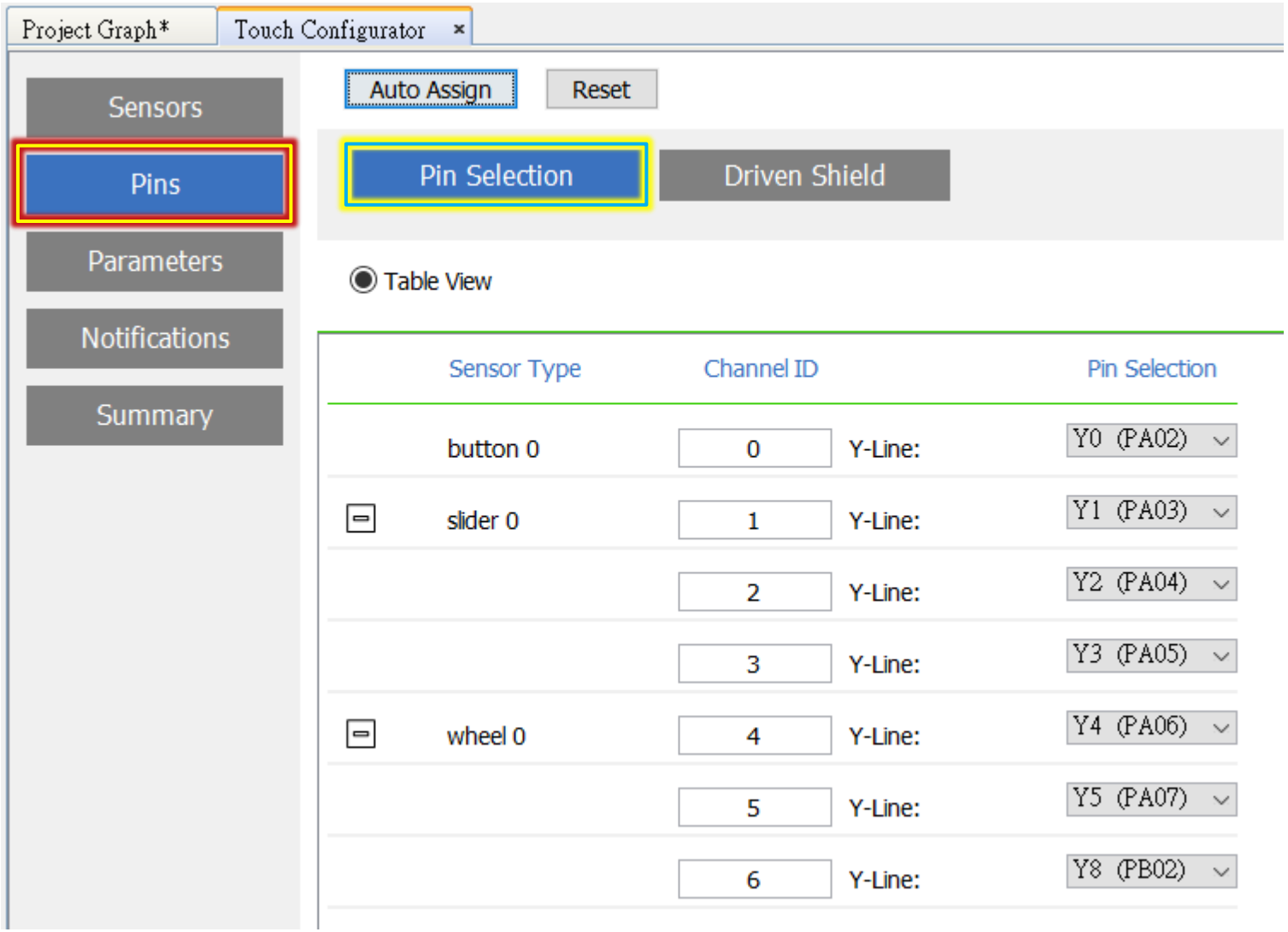


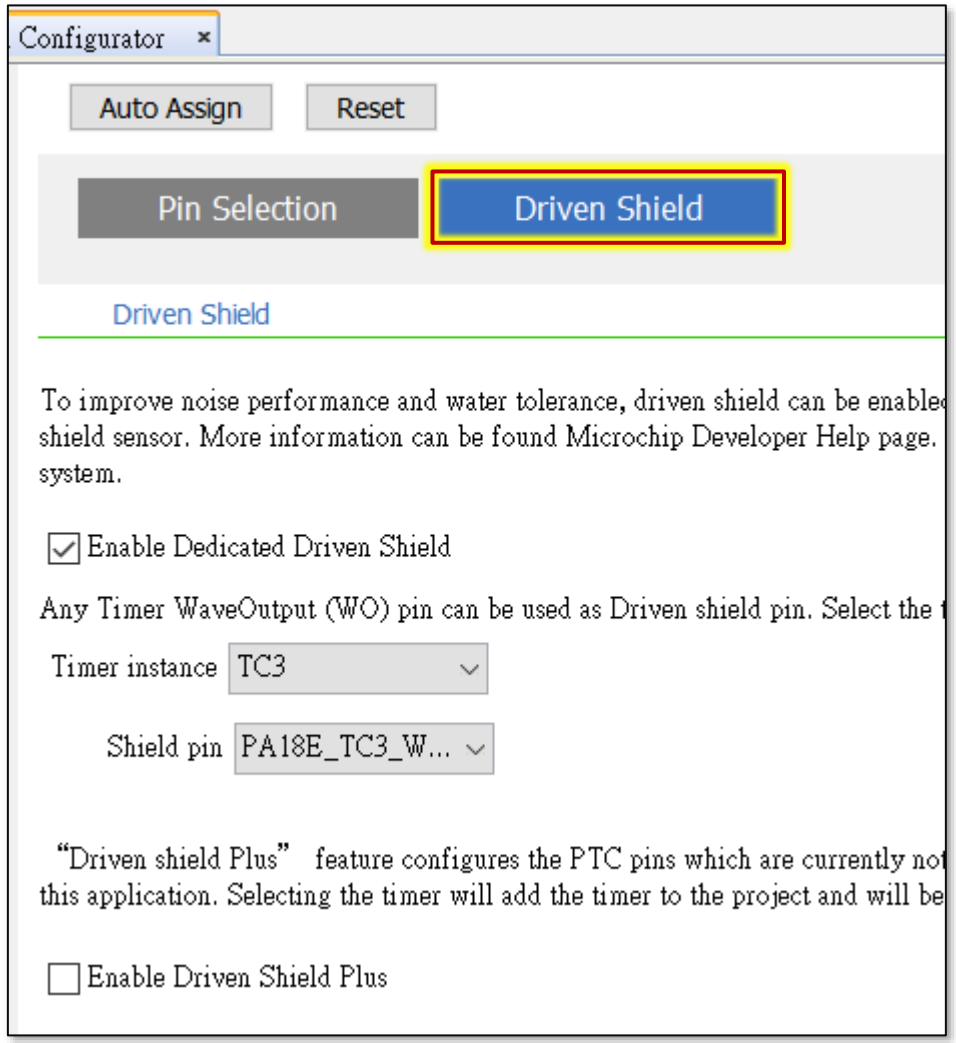


MICROCHIP

Harmony Touch Configurator

Pins : Self Capacitance Sensing Pin Selection and Driven Shield configuration

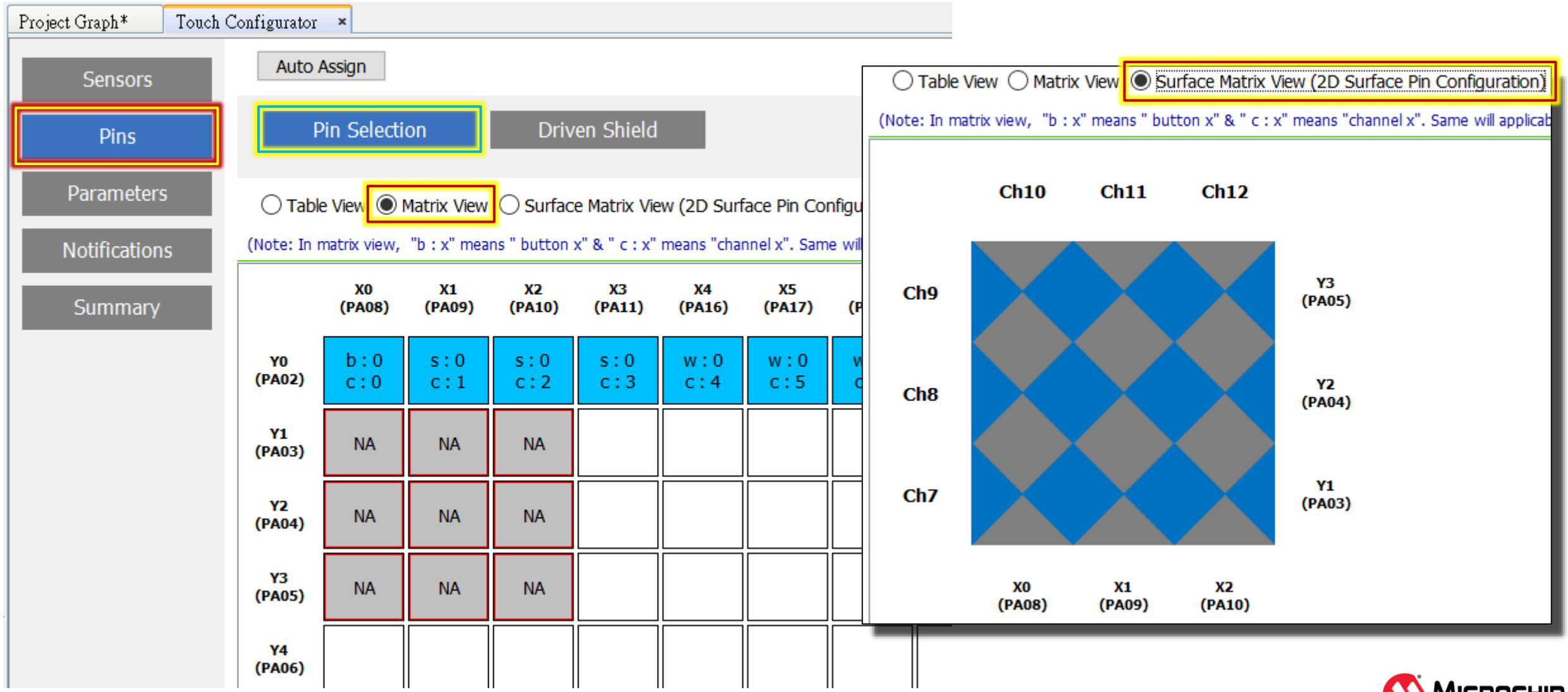




MICROCHIP

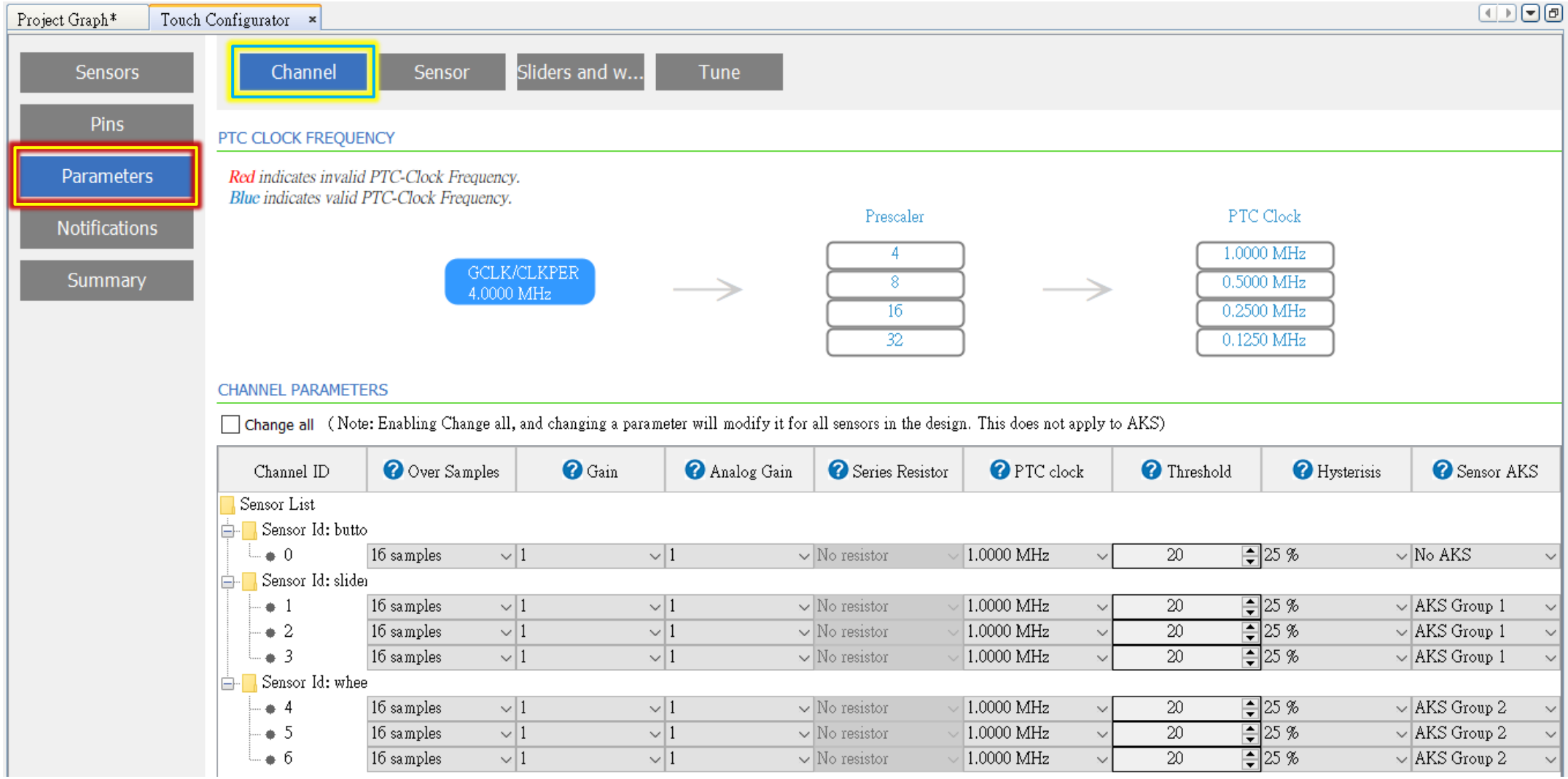
Harmony Touch Configurator

Pins : Mutual Capacitance Sensing kind of Pin Selection GUIs



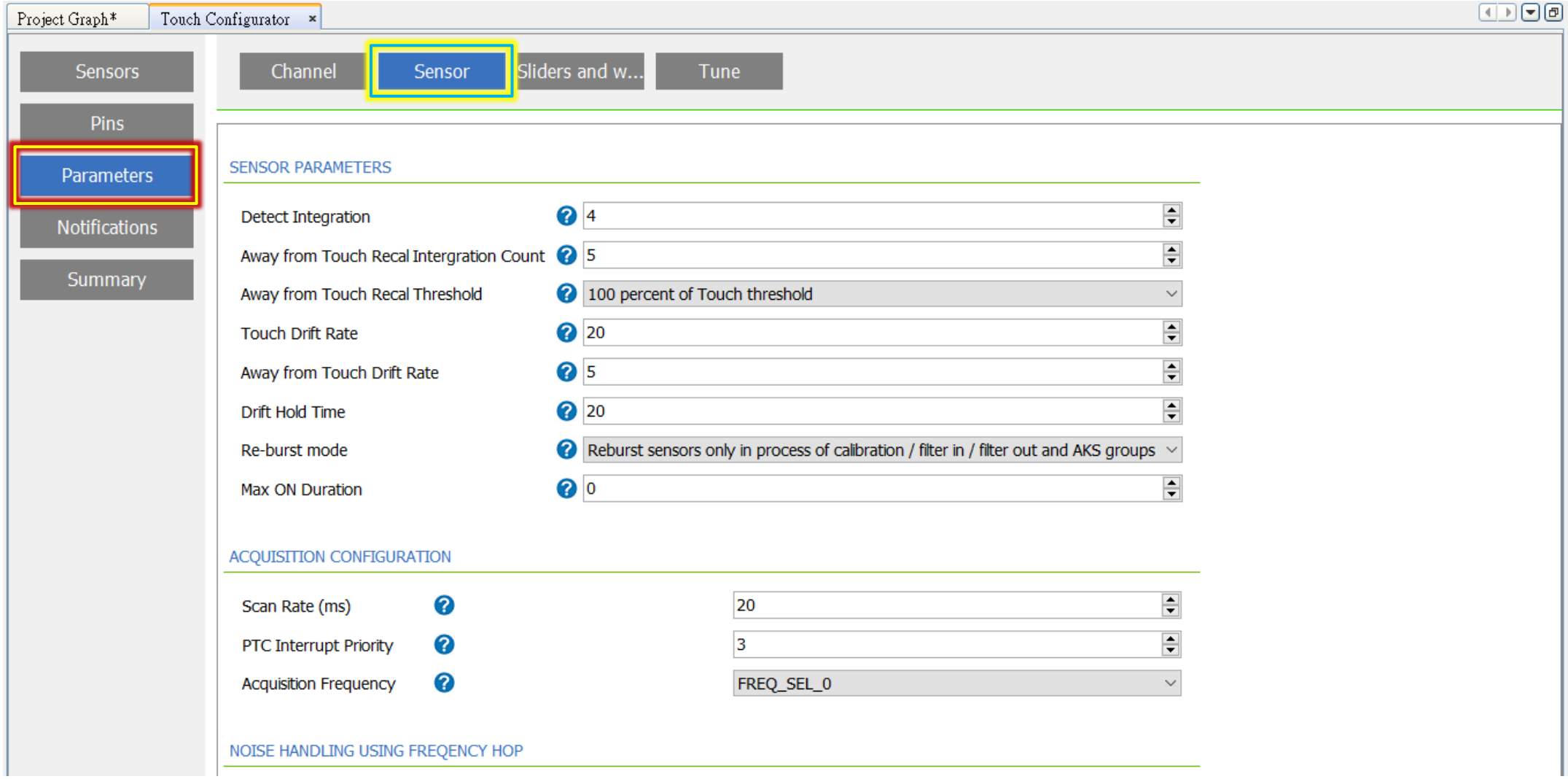
Harmony Touch Configurator

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



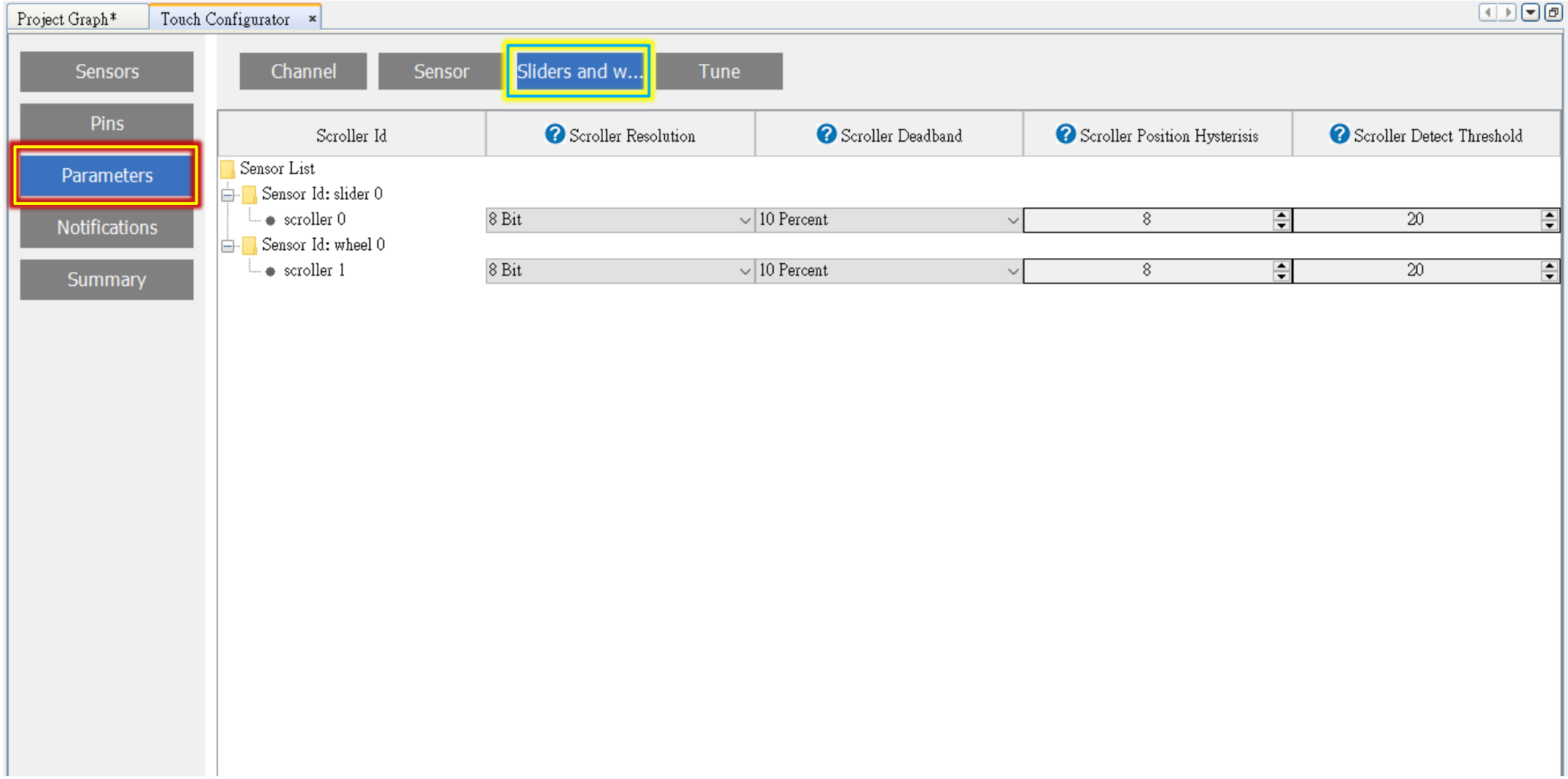
Harmony Touch Configurator

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



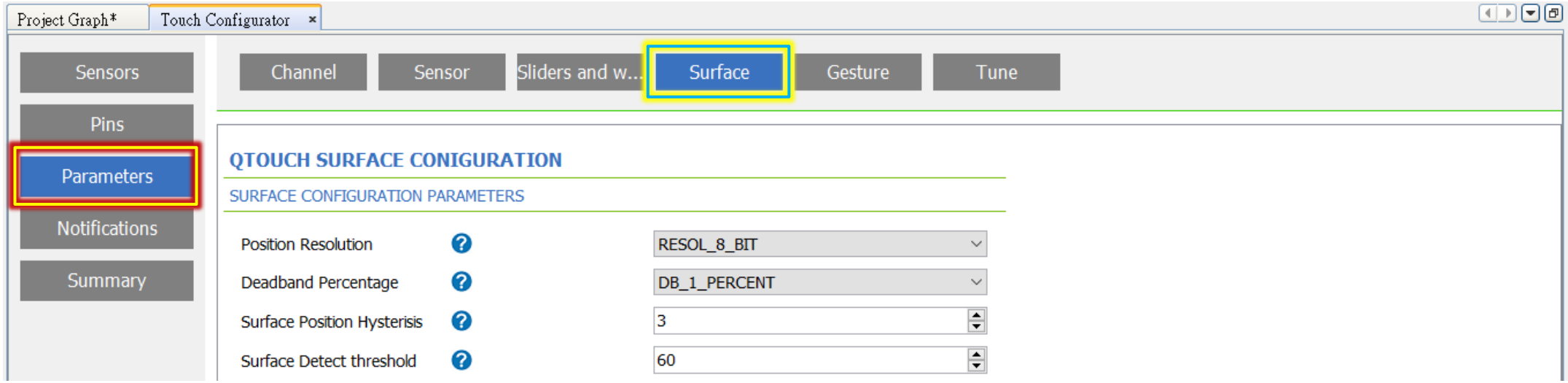
Harmony Touch Configurator

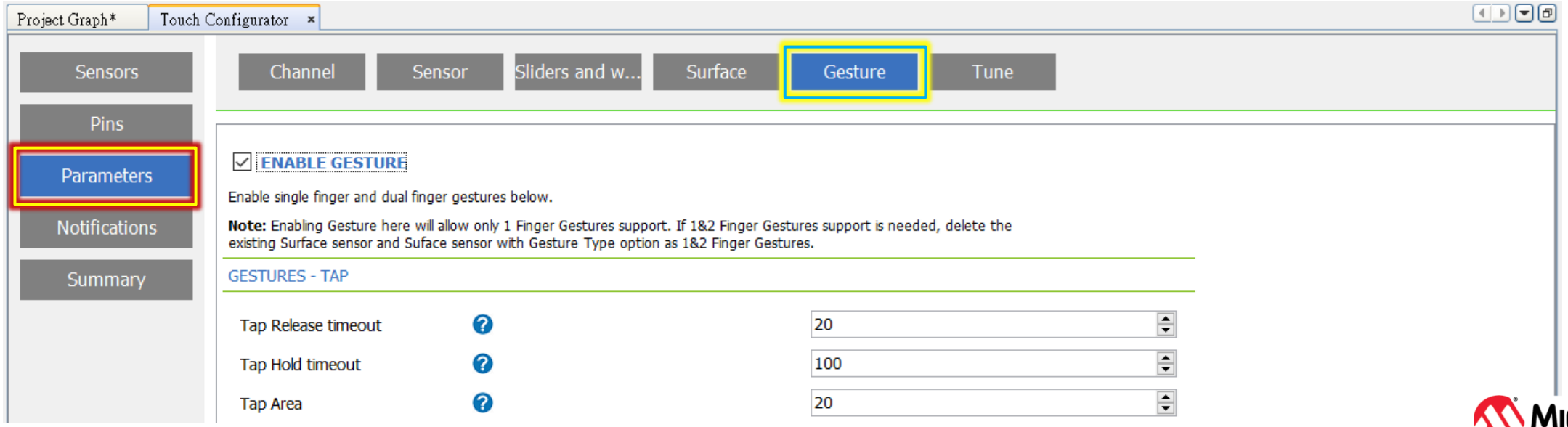
Parameters (Slider And Wheels): Scroller Resolution, Position Hysteresis and Threshold



Harmony Touch Configurator

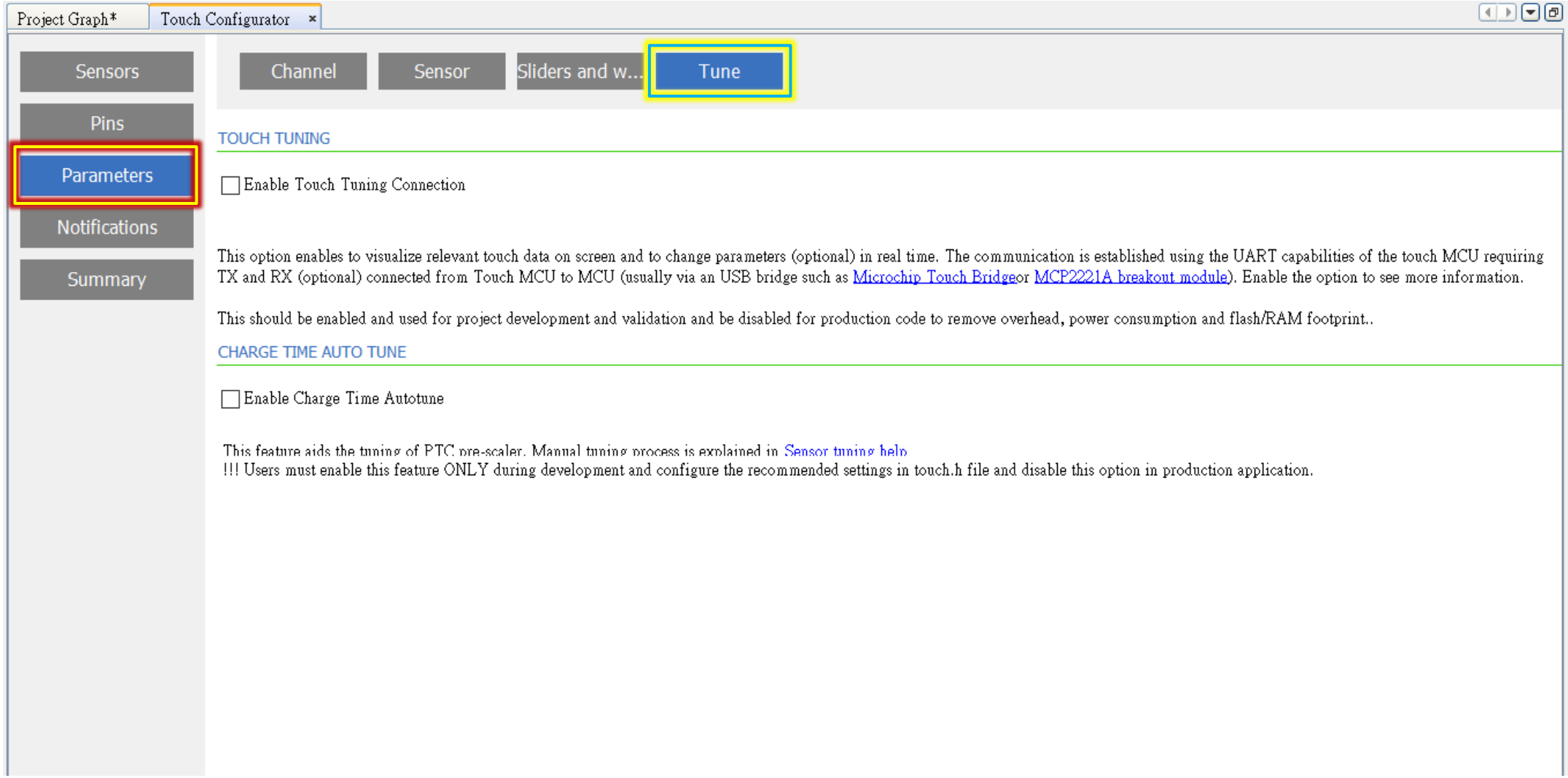
Parameters (Surface/Gesture): QTouch Surface and 2D Gesture configuration





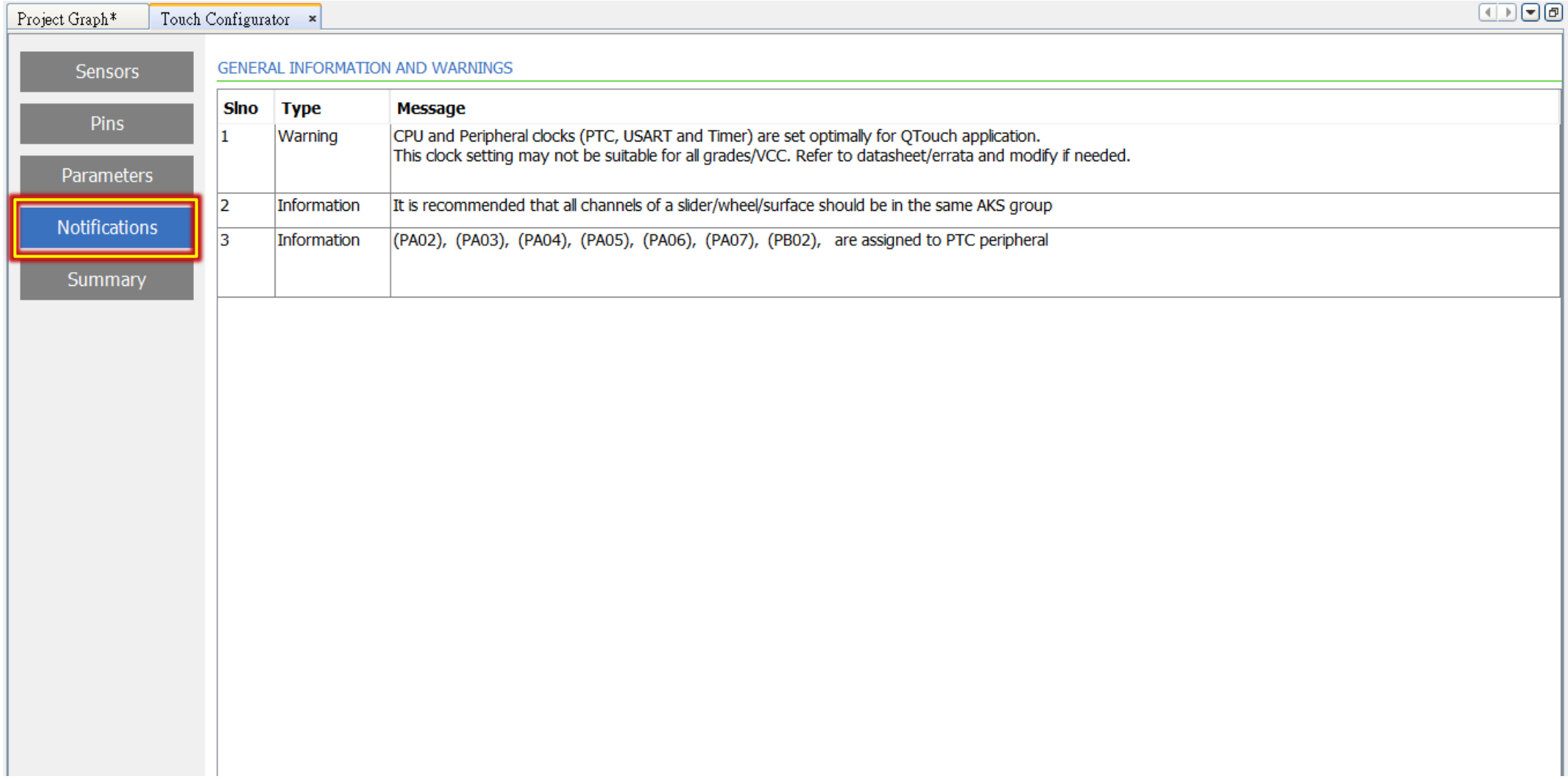
Harmony Touch Configurator

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



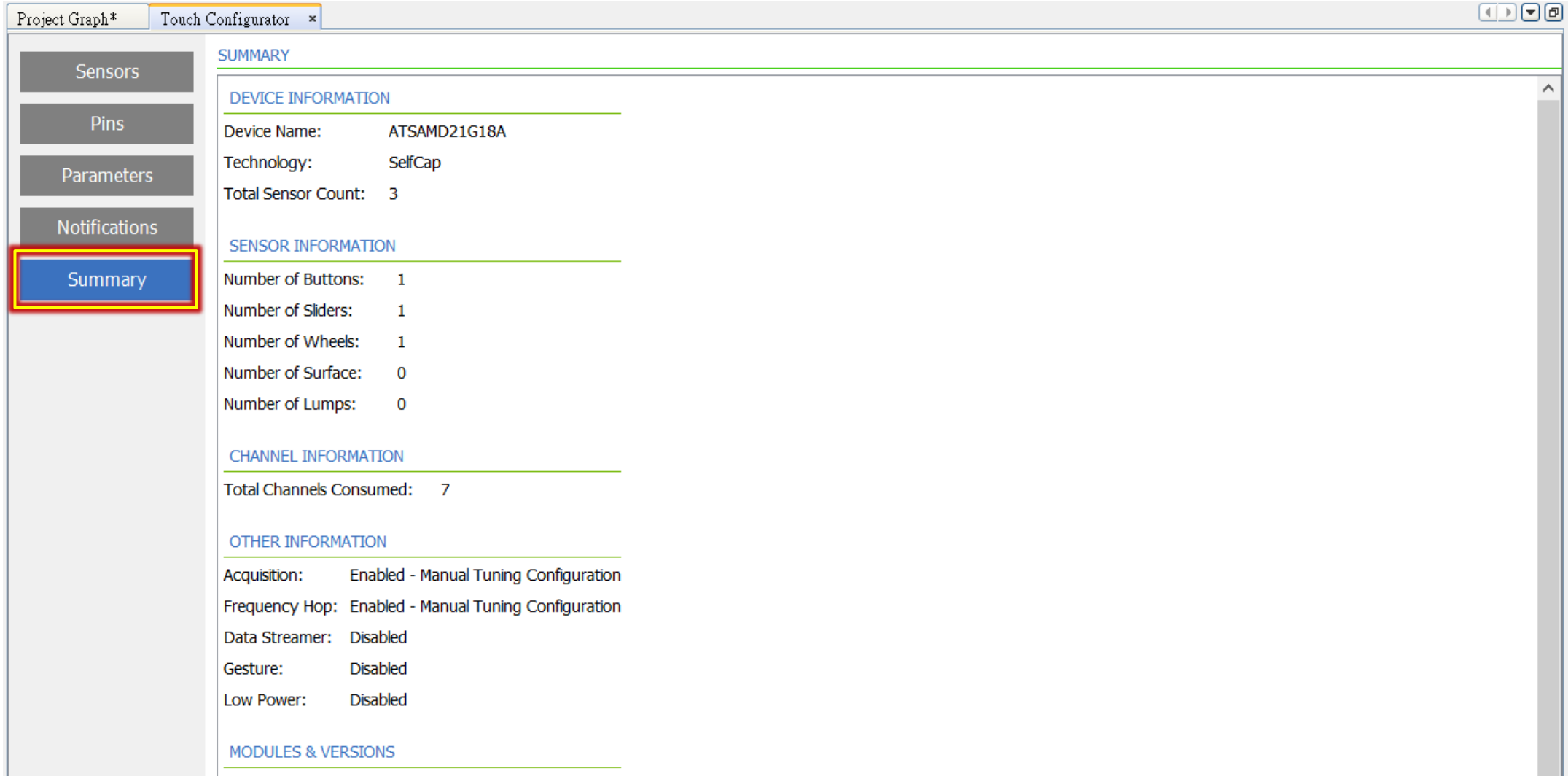
Harmony Touch Configurator

Notifications: Check if there are any warnings or guidelines



Harmony Touch Configurator

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



🛠 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)

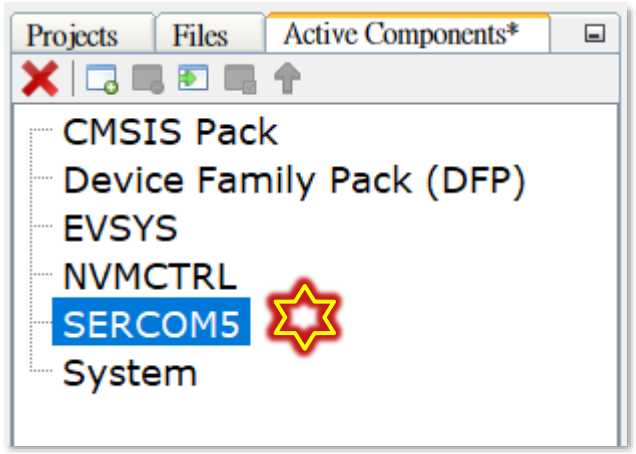
- **SAM2002ADV will use below tool versions for Labs.**
  - MPLAB® X IDE 6.00 https://www.microchip.com/mplab/mplab-x-ide
  - MPLAB® XC32 v4.10 https://www.microchip.com/mplab/compilers
  - MPLAB® Harmony 3 Launcher v3.6.4 (From X IDE plugin)
  - MPLAB® Harmony Framework (From X IDE Harmony 3 Content 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)

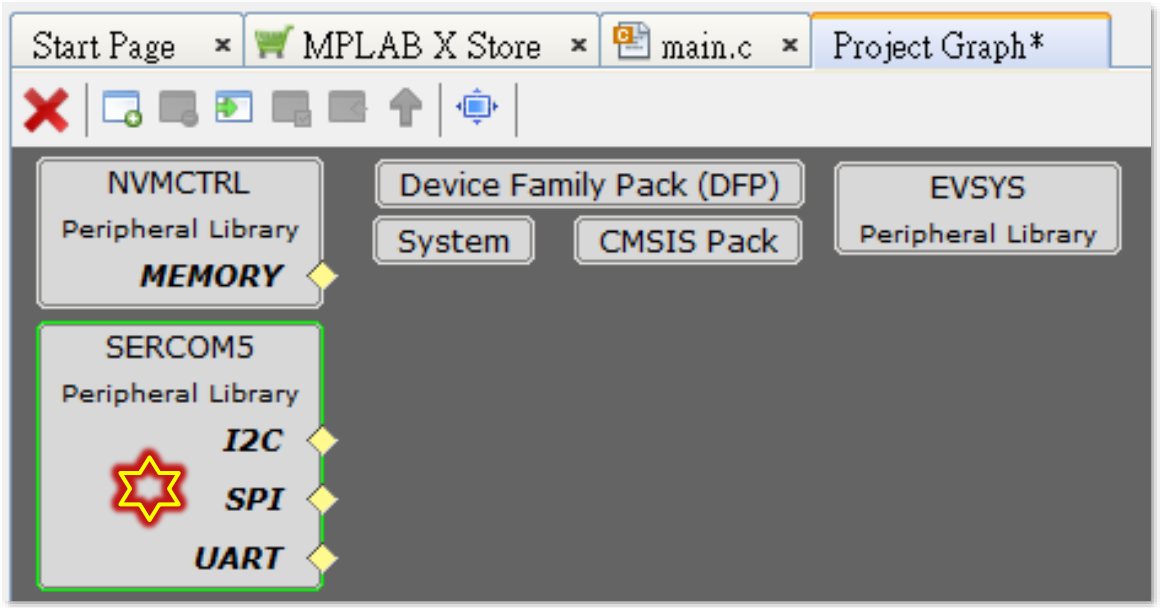
- **Let’s go !**

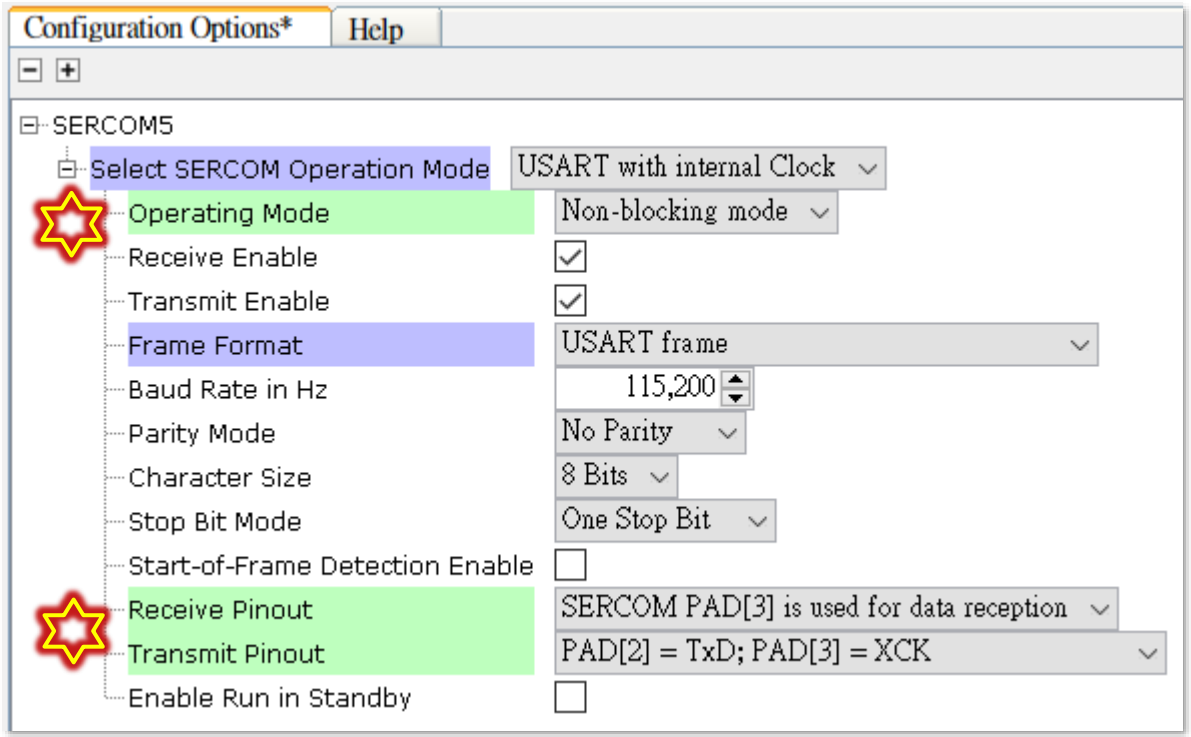
🛠 Lab1-1 Qtouch Buttons

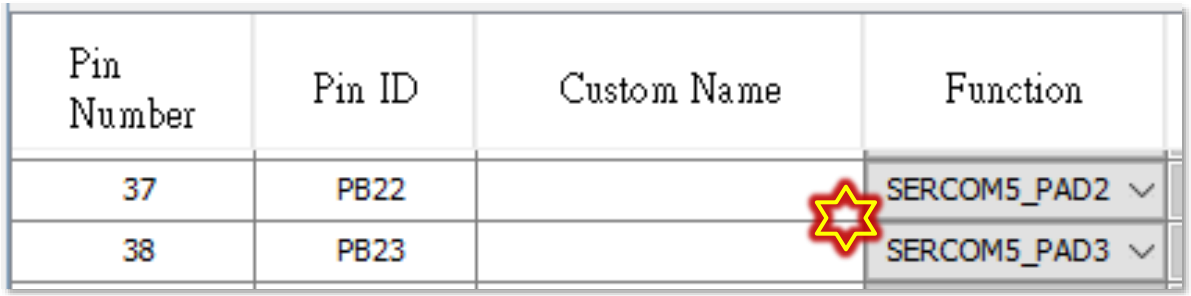
Lab1-1 Qtouch Buttons

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



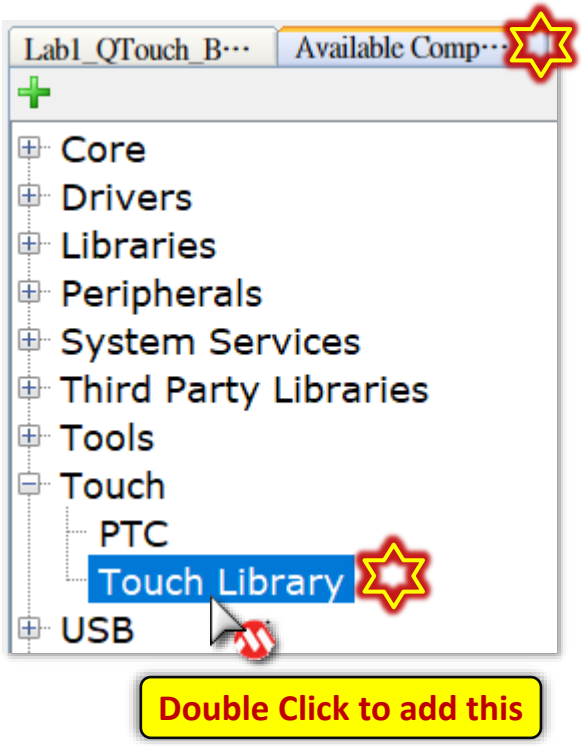


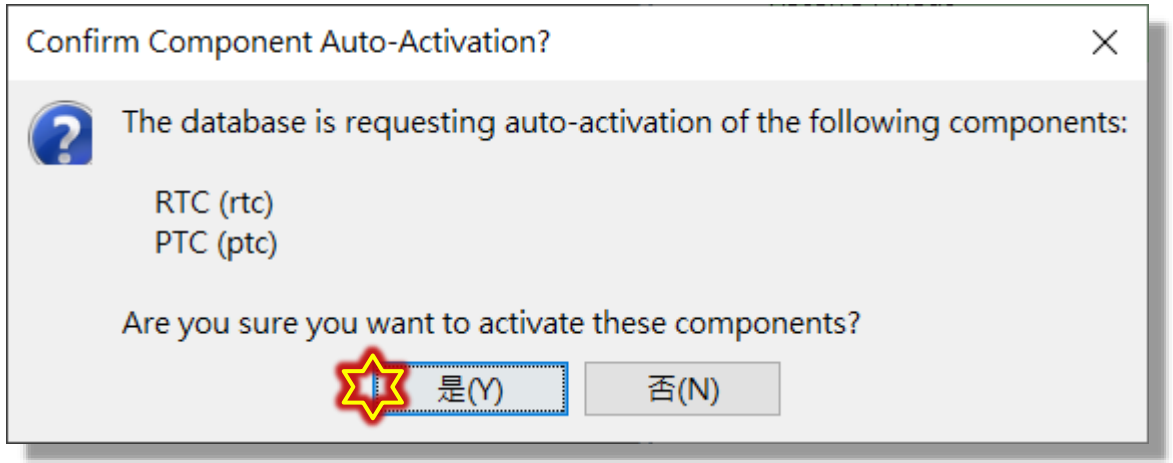


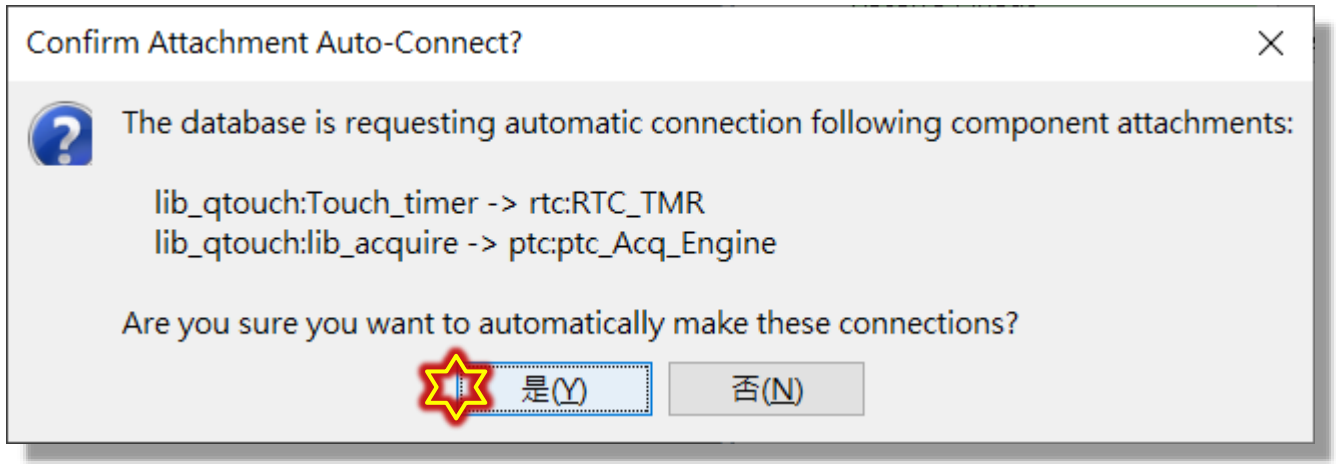


🛠 Lab1-1 Qtouch Buttons

- 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”.



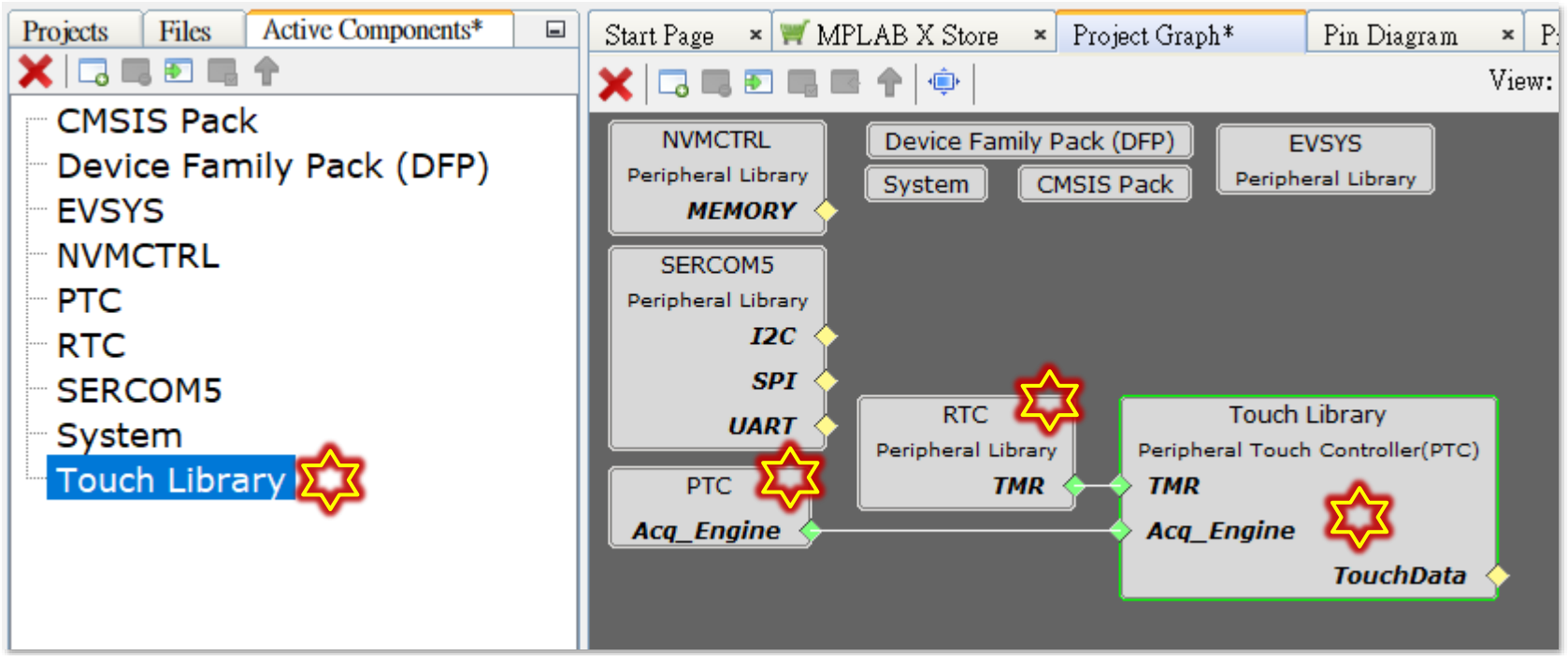


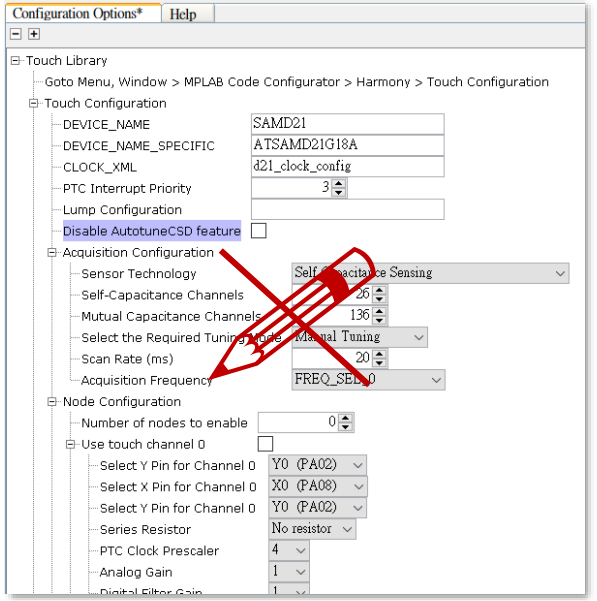


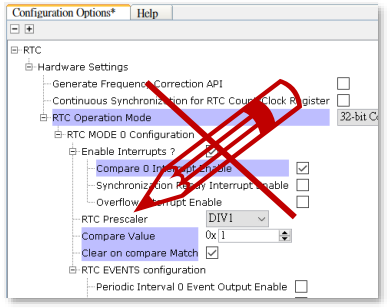
MICROCHIP

🛠 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.



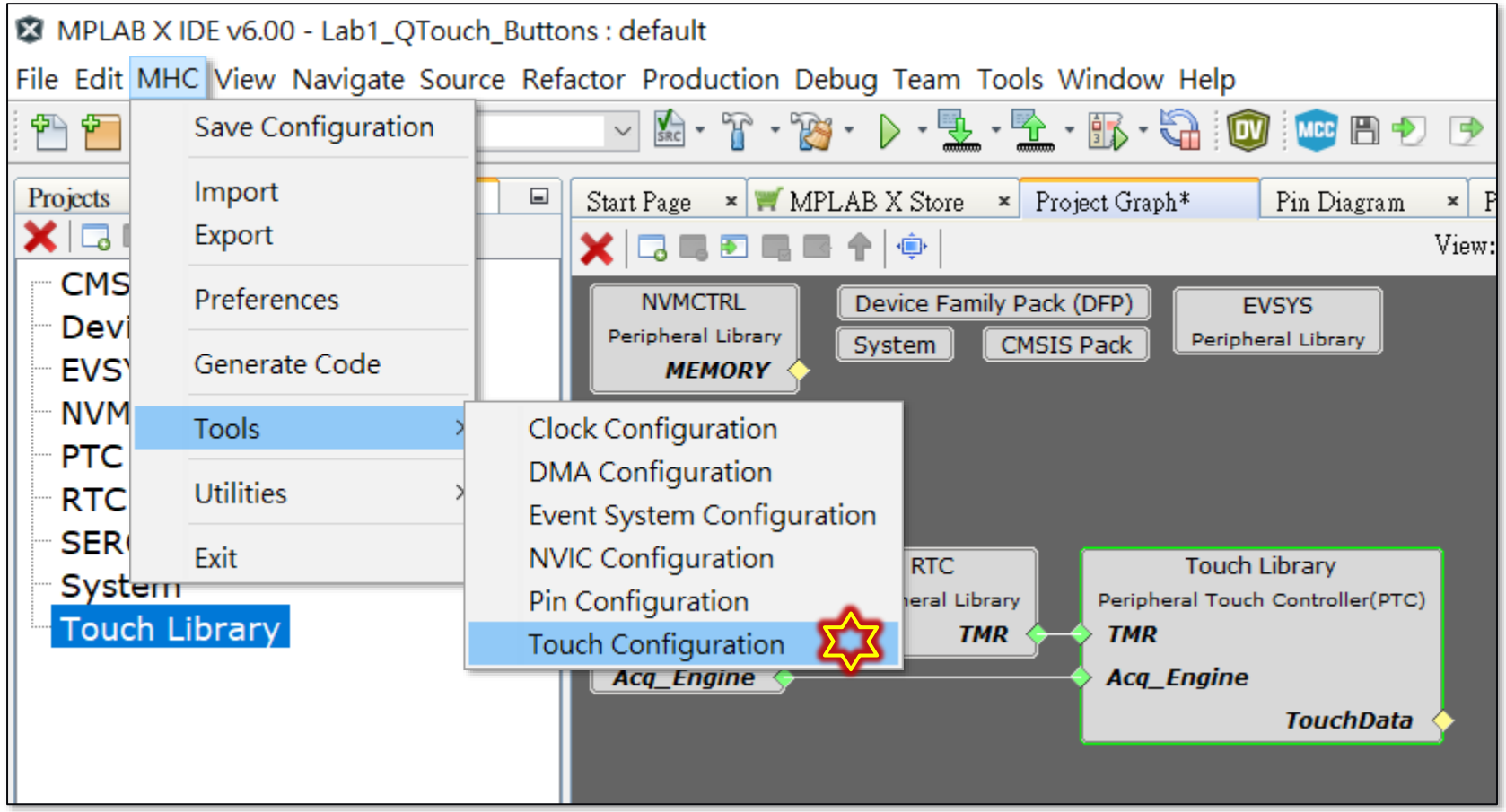




MICROCHIP

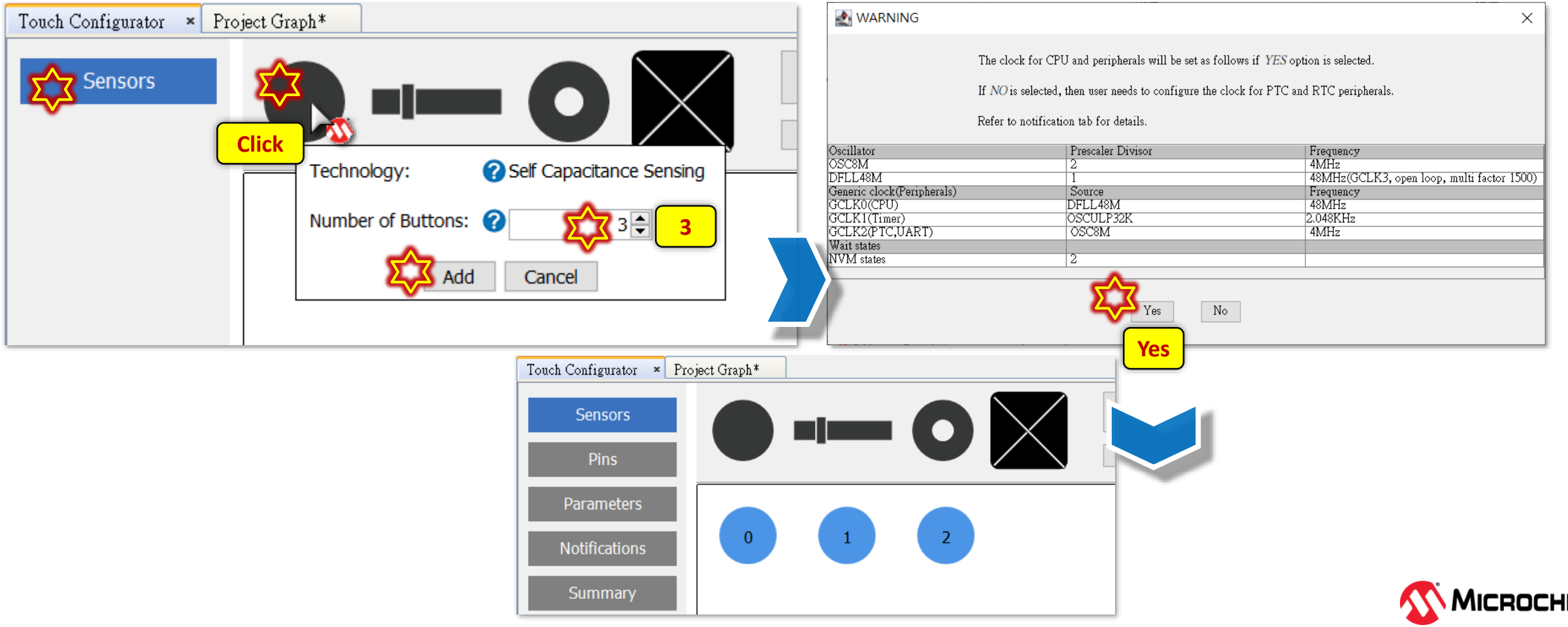
Lab1-1 Qtouch Buttons

- **Open Touch Configuration**



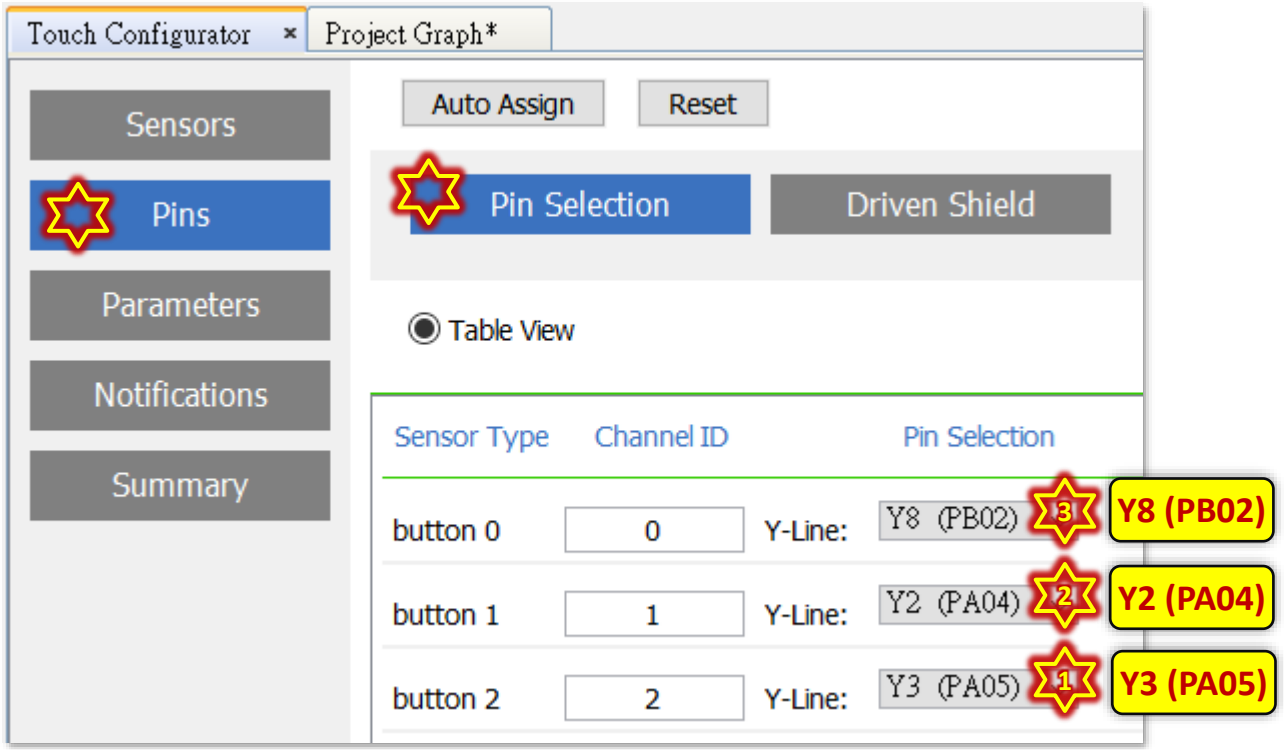
🛠 Lab1-1 Qtouch Buttons

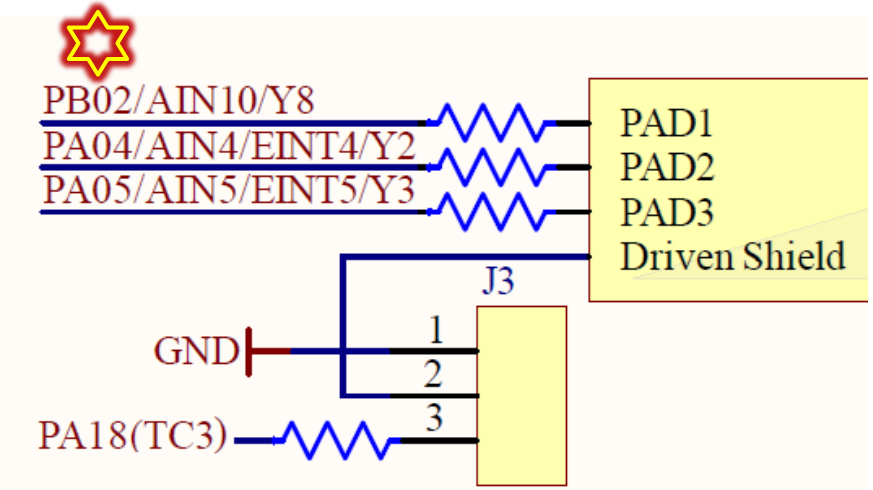
- In Sensor Tab, click on Button Icon and input Number of Buttons to 3.
- Click Add then a Warning dialog will show up to notice the clock of CPU and peripheral will be auto set as default Touch Function request if select “Yes”



🛠 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)

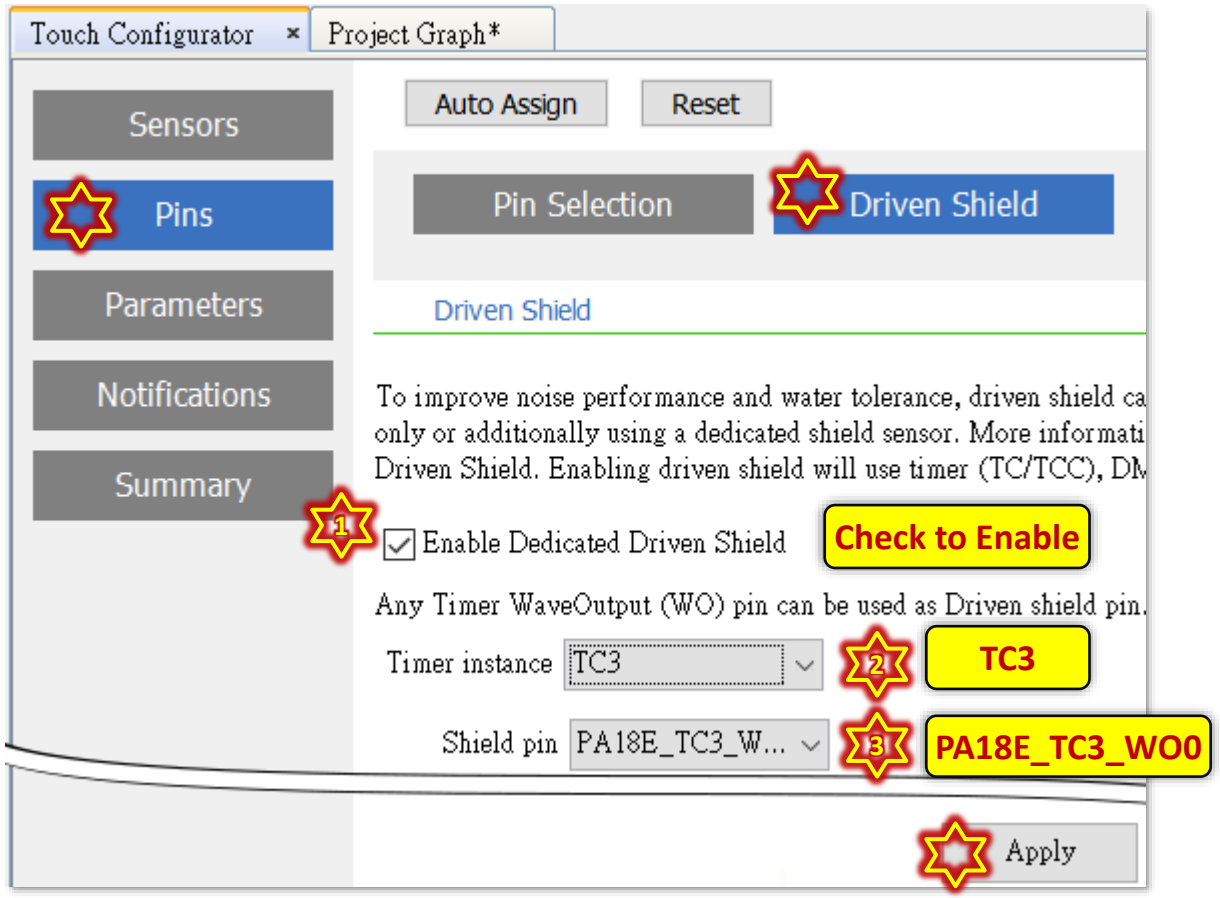


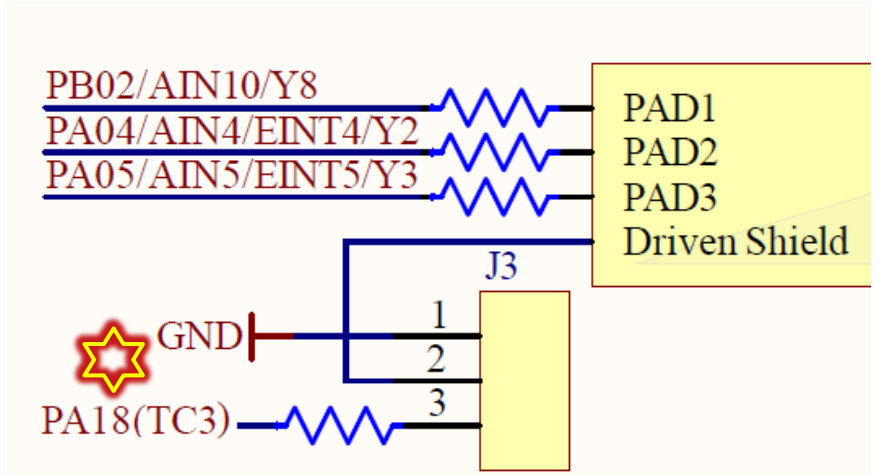


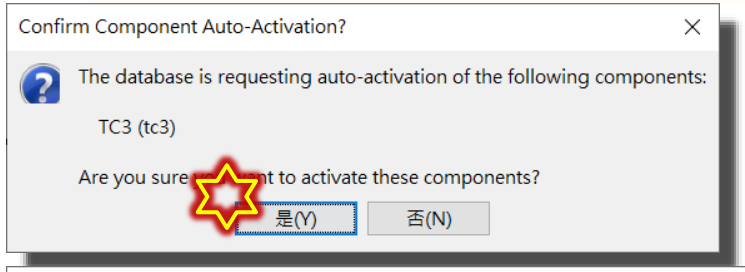
MICROCHIP

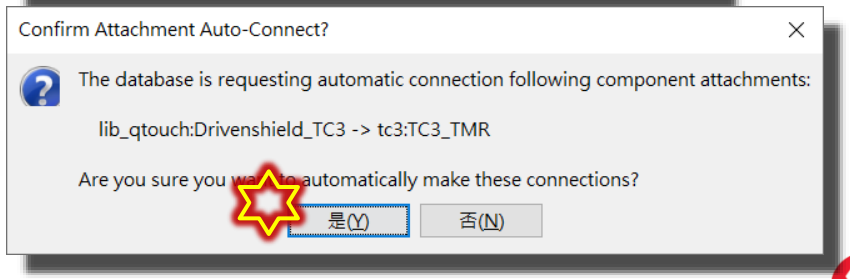
🛠 Lab1-1 Qtouch Buttons

- **In Pins Tab, Click on Driven Shield and configure Dirven Shield as below.**
  - Enable Dedicated Dirven Shield : ☑
  - Timer instance : TC3
  - Shield pin : PA18E_TC3_WO0





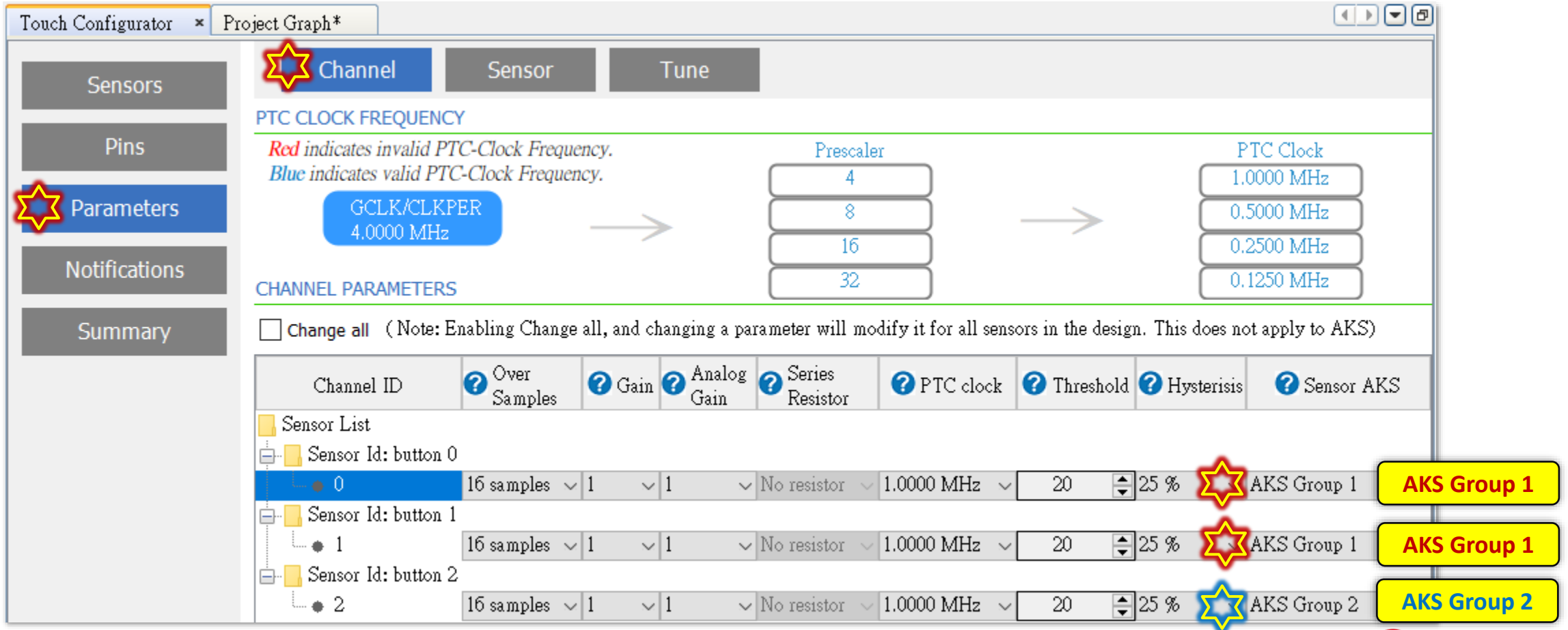




MICROCHIP

🛠 Lab1-1 Qtouch Buttons

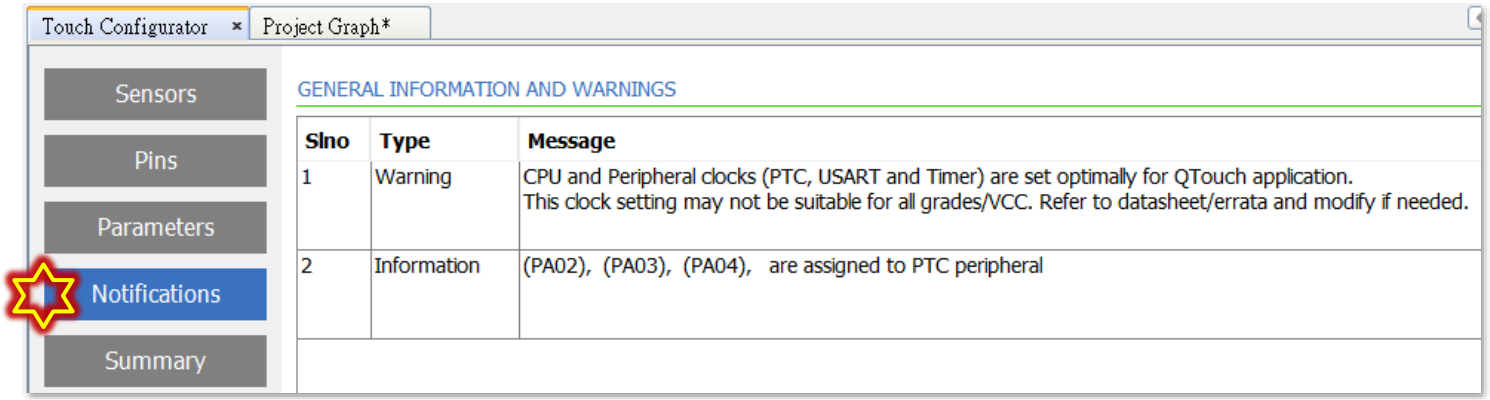
- **In Parameters Tab, configure Sensor AKS(Adjacent Keys Suppression) group as below.**
  - Button0 and Button1 : AKS Group 1
  - Button2 : AKS Group 2

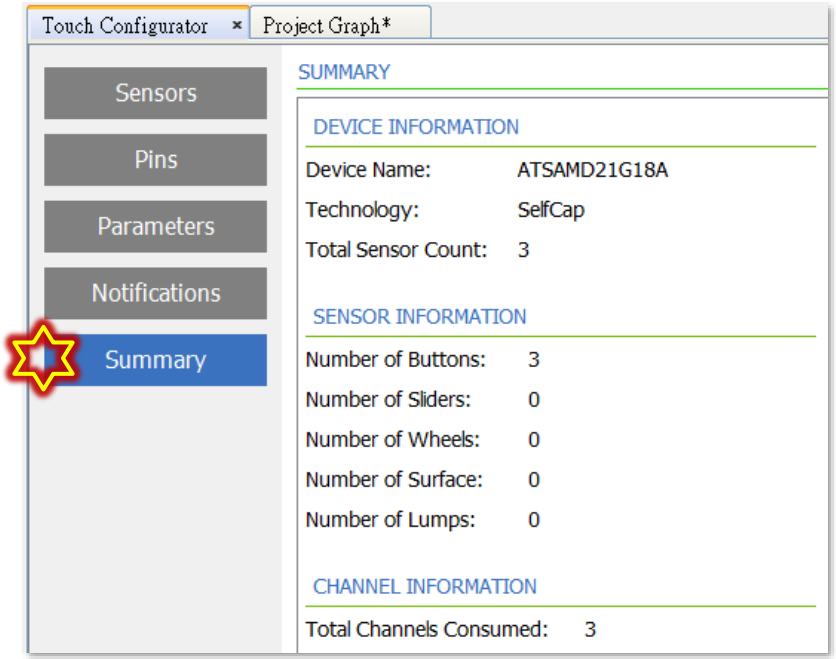


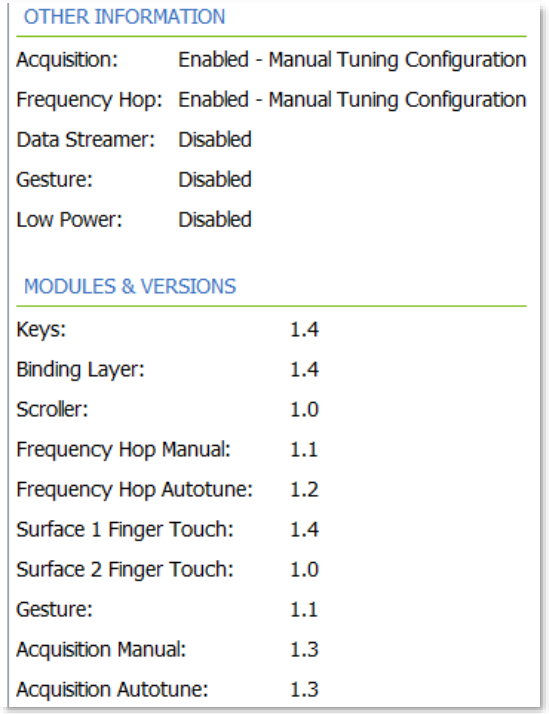
MICROCHIP

🛠 Lab1-1 Qtouch Buttons

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







MICROCHIP

Lab1-1 Qtouch Buttons

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

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

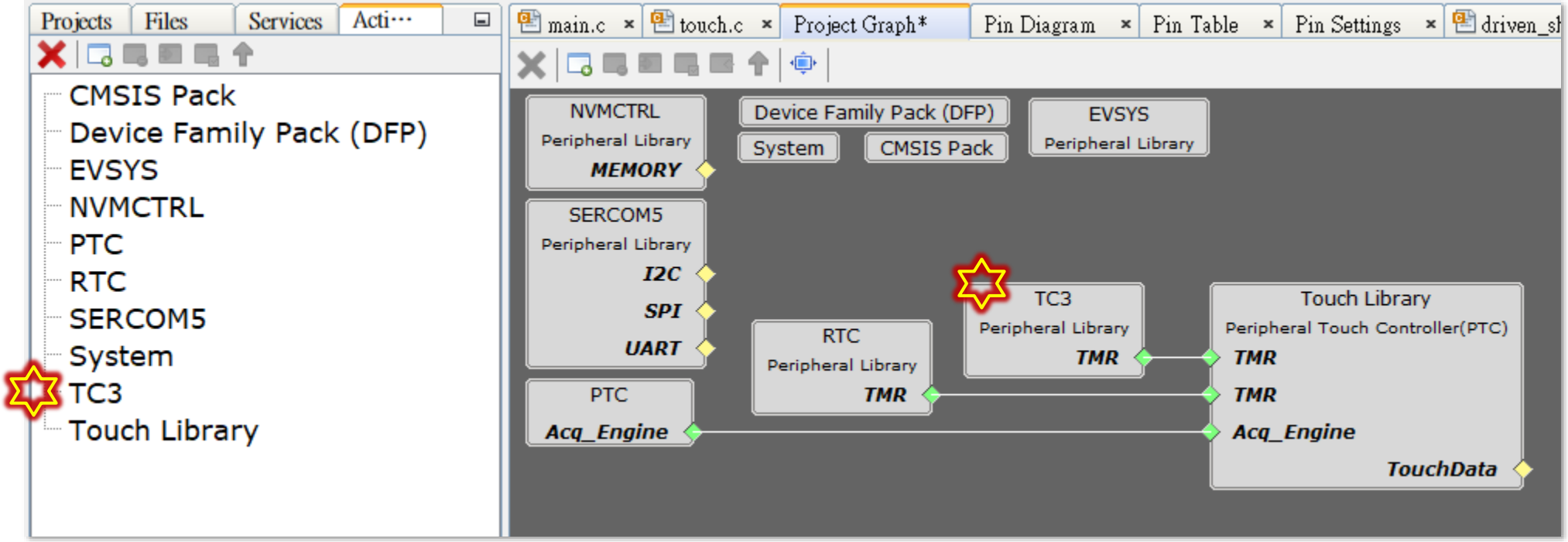
Order: Ports | Table View | Easy View

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

MICROCHIP

🛠 Lab1-1 Qtouch Buttons

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



🛠 Lab1-1 Qtouch Buttons

API functions in Touch.c

uint8_t measurement_done_touch	Touch acquisition done flag
touch_process ()	Main processing function of touch library. This function initiates the acquisition, calls post processing after the acquisition complete and sets the flag for next measurement based on the sensor status.
touch_init ()	Initialization PTC, timer, and datastreamer modules of touch library.
touch_timer_config ()	Initialization necessary RTC timer for Touch in touch_init()
touch_sensors_config ()	Initialization of touch key sensors in touch_init()
get_sensor_node_signal ()	Get sensor node aquiction signals.
update_sensor_node_signal ()	Update sensor node aquiction signals.
get_sensor_node_reference ()	Get sensor node channel reference.
update_sensor_node_reference ()	Update sensor node channel reference.
get_sensor_cc_val ()	Get sensor node compansation capacitor value.
update_sensor_cc_val ()	Update sensor node compansation capacitor value.
get_sensor_state ()	Get sensor node state
update_sensor_state ()	Update sensor node state
calibrate_node ()	Sensor node calibration

API functions in driven_shield.c

drivenshield_configure ( )	Iinitialization Driven Shield for Touch in touch_init()

MICROCHIP

🛠 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
    }
}
```

MICROCHIP

🛠 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");
    ...
```

MICROCHIP

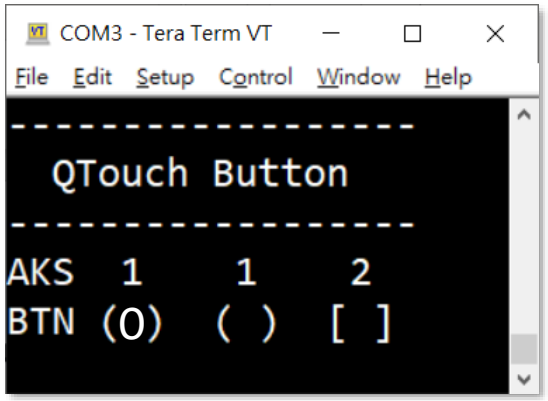
🛠 Lab1-1 Qtouch Buttons

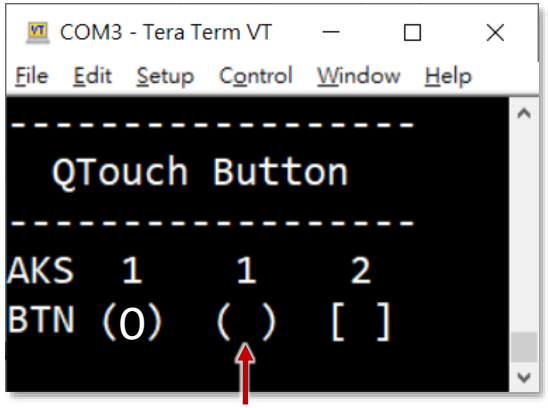
- **Implement below function in while loop of main.c to make Touch Buttons works.**

```
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':' ') );
        }
        ...
    }
```

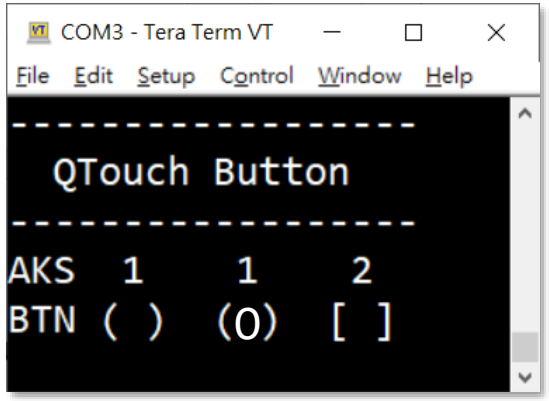
🛠 Lab1-1 Qtouch Buttons

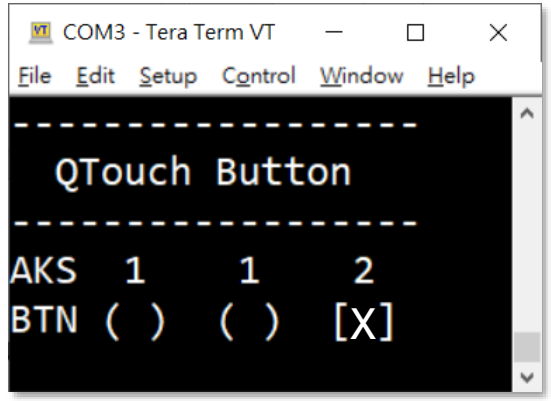
- **Result of Touch Key implementation.**

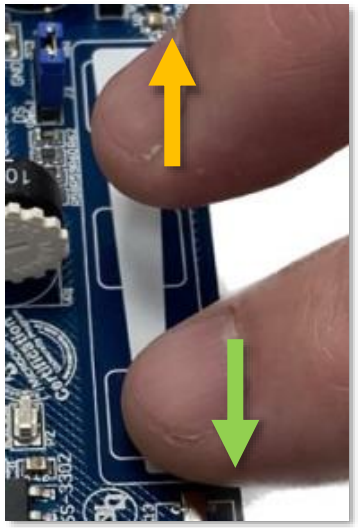


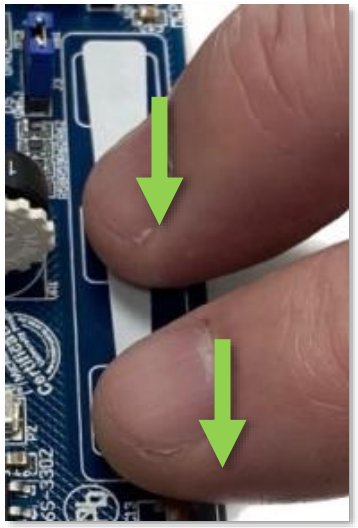


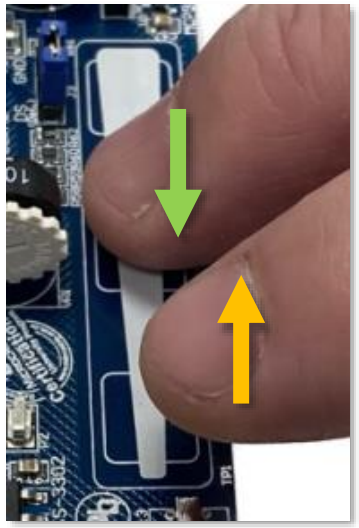
(Adjacent Keys Suppression)

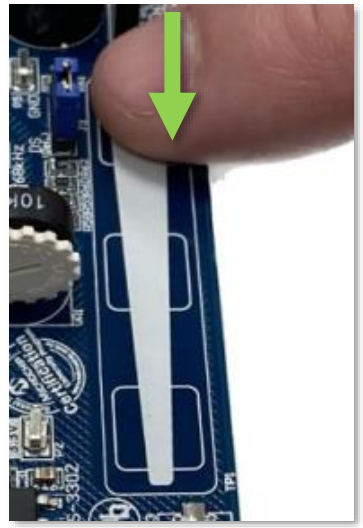












🛠 Lab1-1 Qtouch Buttons

Discuss

- **Implement manual Touch Calibration feature.**

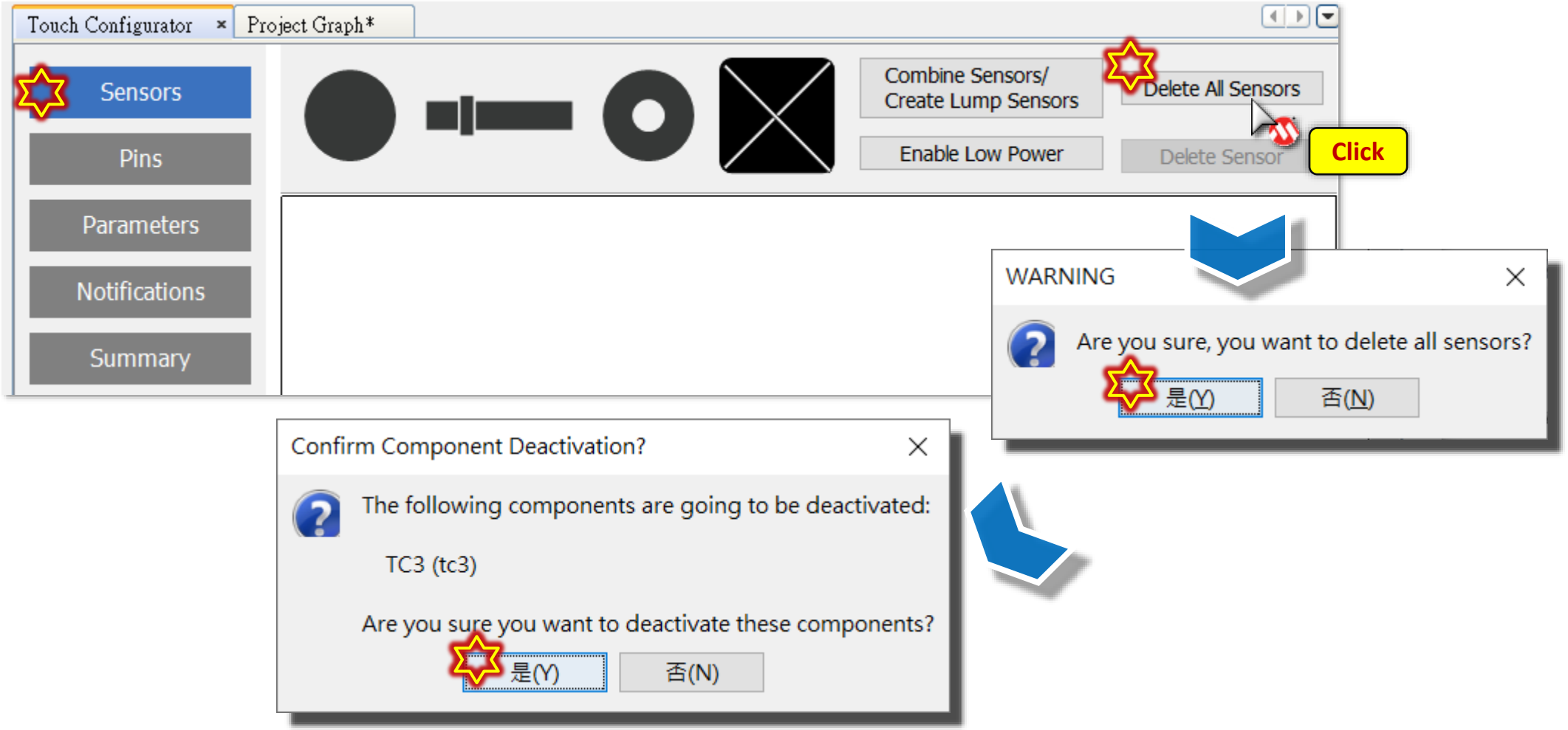
```
int main ( void )
{
    bool KeyStatus[3] = {false, false, false};
    ...

    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;
            }
            ...
        }
    }
```

MICROCHIP

🛠 Lab1-2 Qtouch Slider

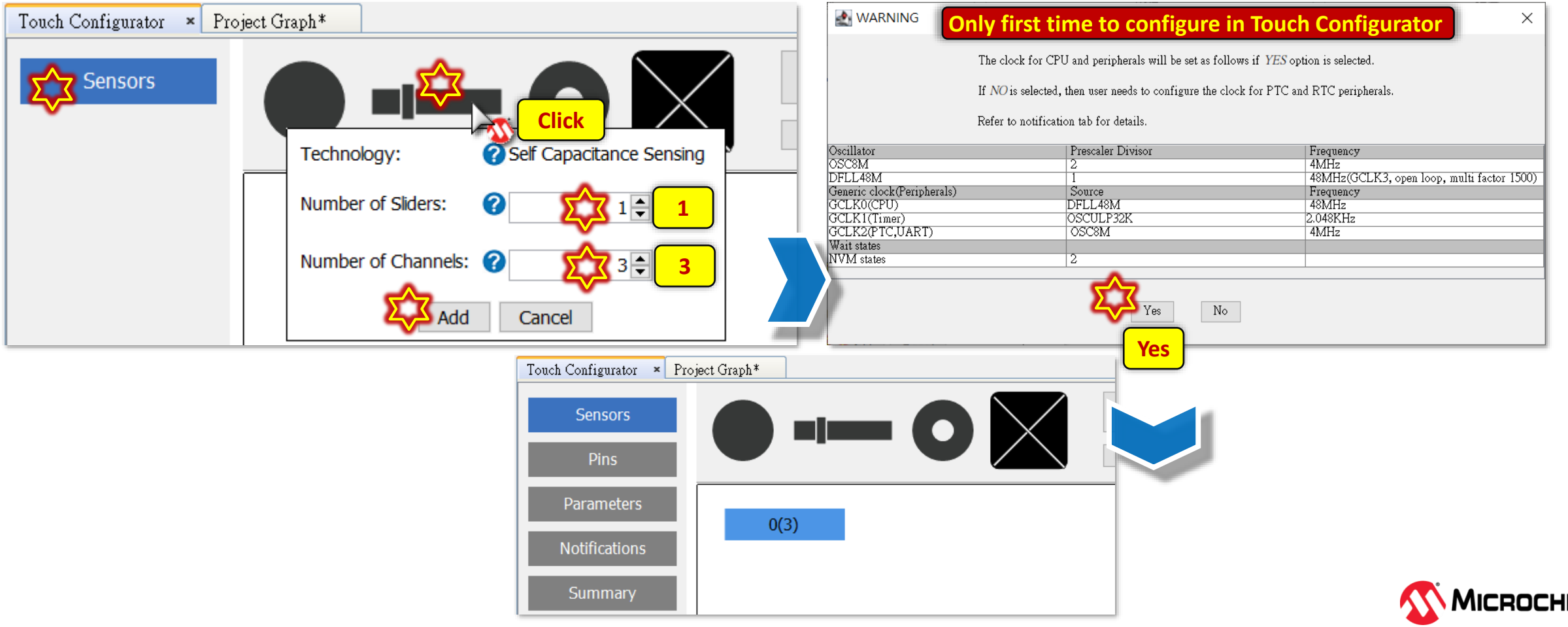
🛠 Lab1-2 Qtouch Slider

- 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

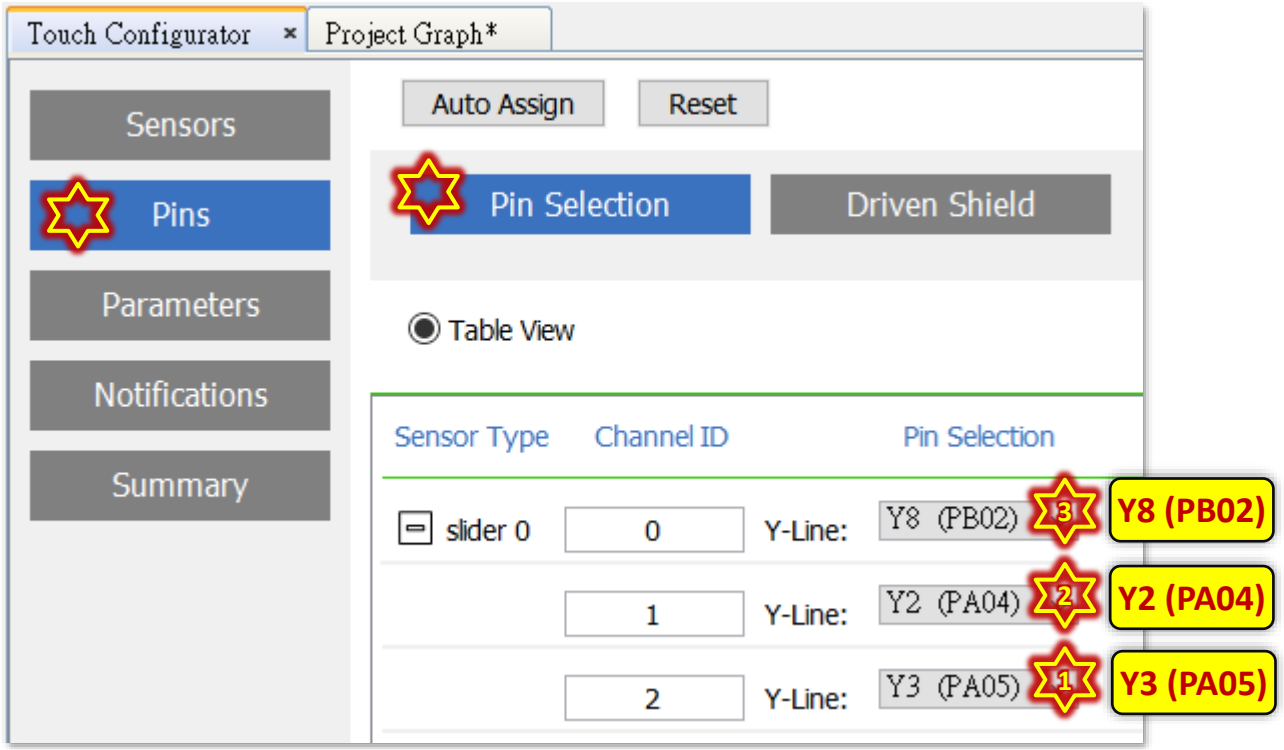
- Click on Slider(Scroller) Icon, default Number of Sliders : 1 and Number of Channels : 3
- Click Add, if you are first configuring in Touch Configurator, a Warning dialog will show up to notice the clock will be auto set as default Touch Function request if select “Yes”

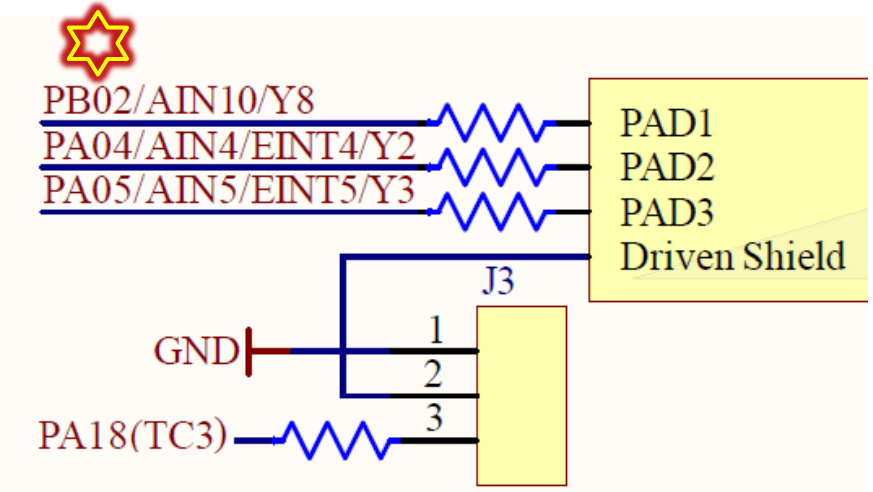


MICROCHIP

Lab1-2 Qtouch Slider

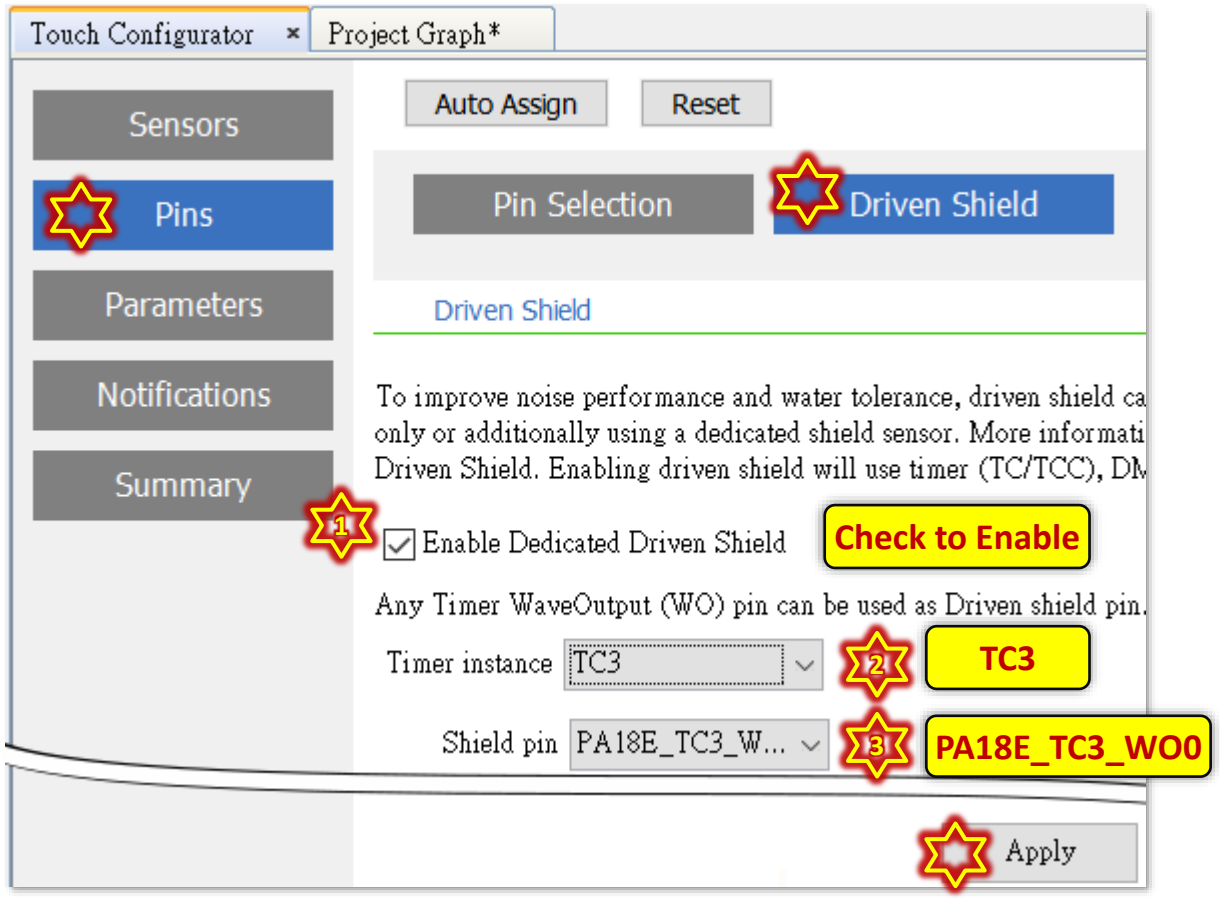
- **In Pins Tab, Click on Pin Selection and configure button pins in below sequence.**
  - Slider0 (Channel ID 2) : Y3 (PA05)
  - (Channel ID 1) : Y2 (PA04)
  - (Channel ID 0) : Y8 (PB02)

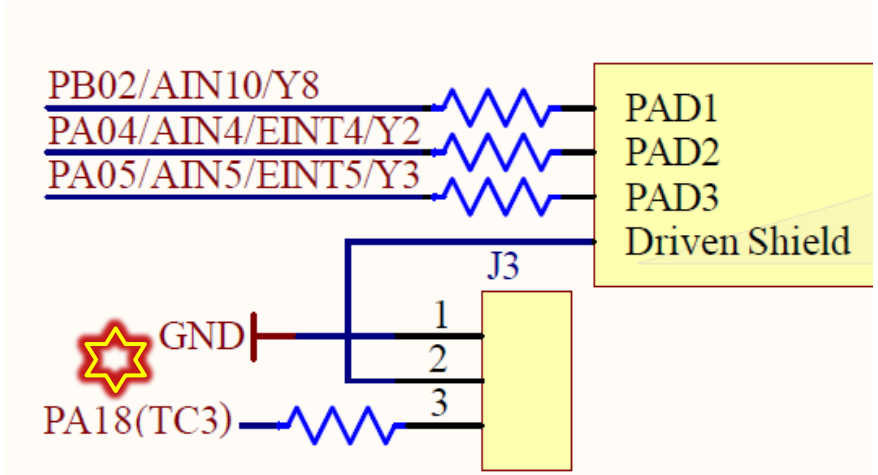


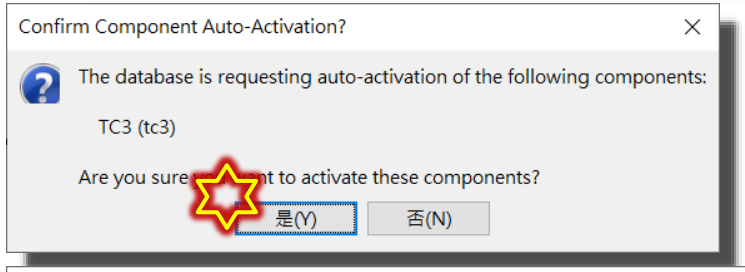


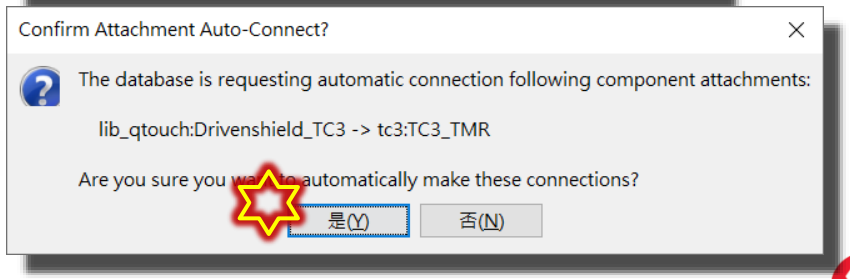
🛠 Lab1-2 Qtouch Slider

- **In Pins Tab, Click on Driven Shield and configure Dirven Shield as below.**
  - Enable Dedicated Dirven Shield : ☑
  - Timer instance : TC3
  - Shield pin : PA18E_TC3_WO0





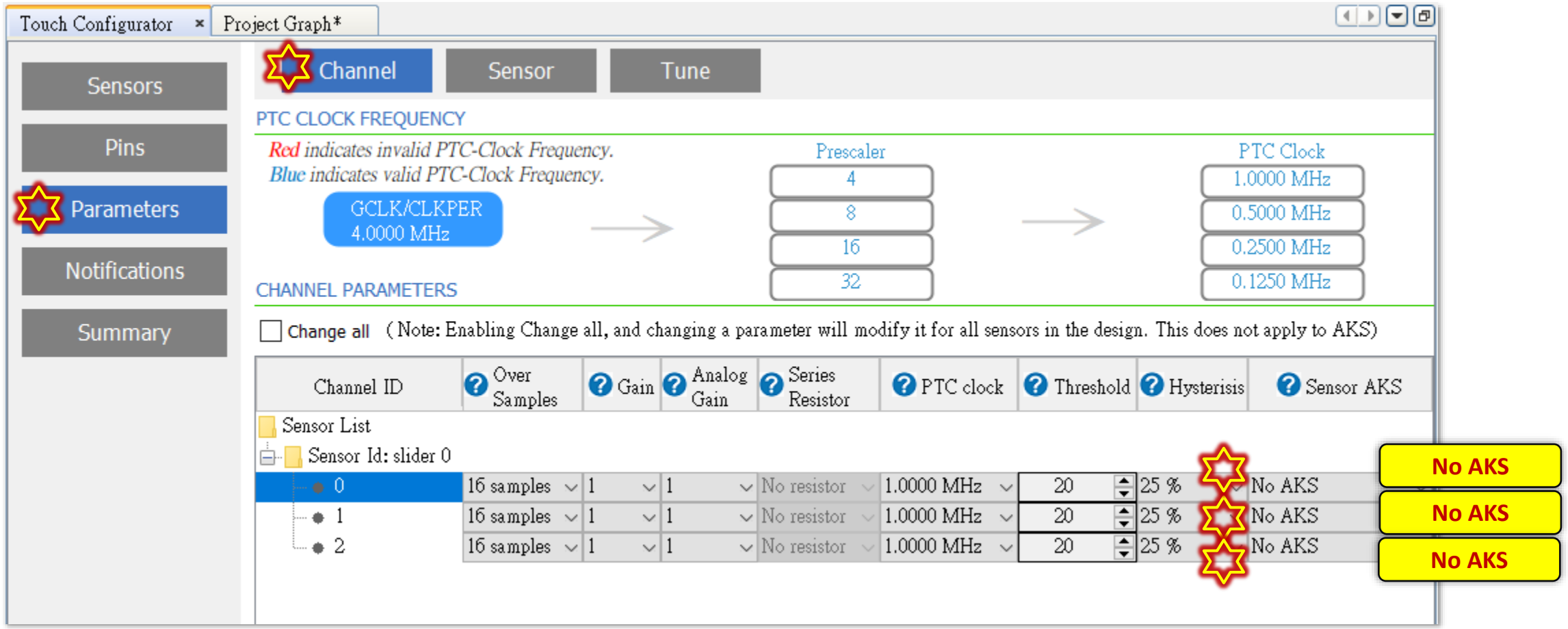




MICROCHIP

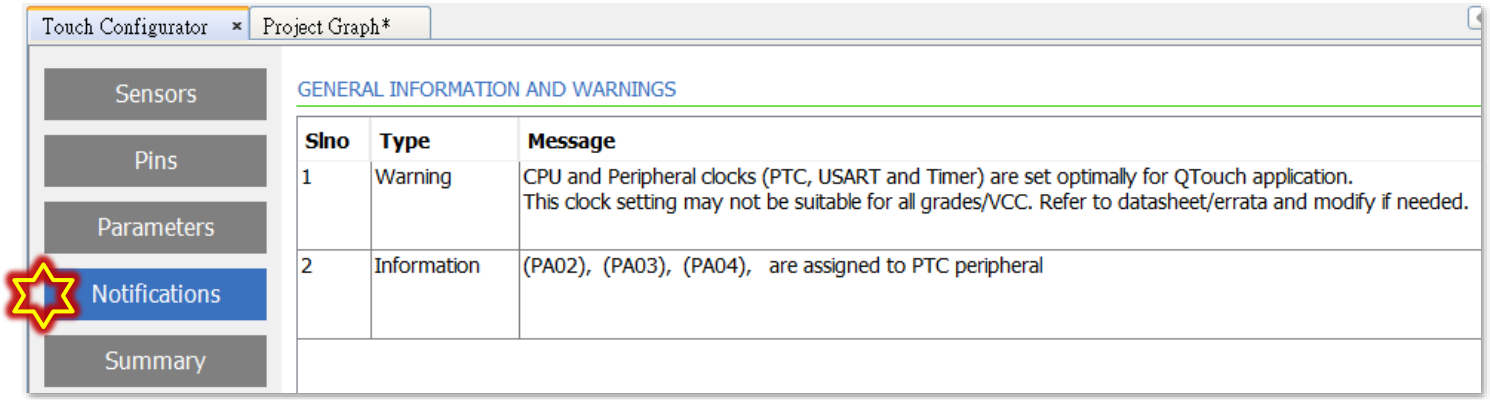
🛠 Lab1-2 Qtouch Slider

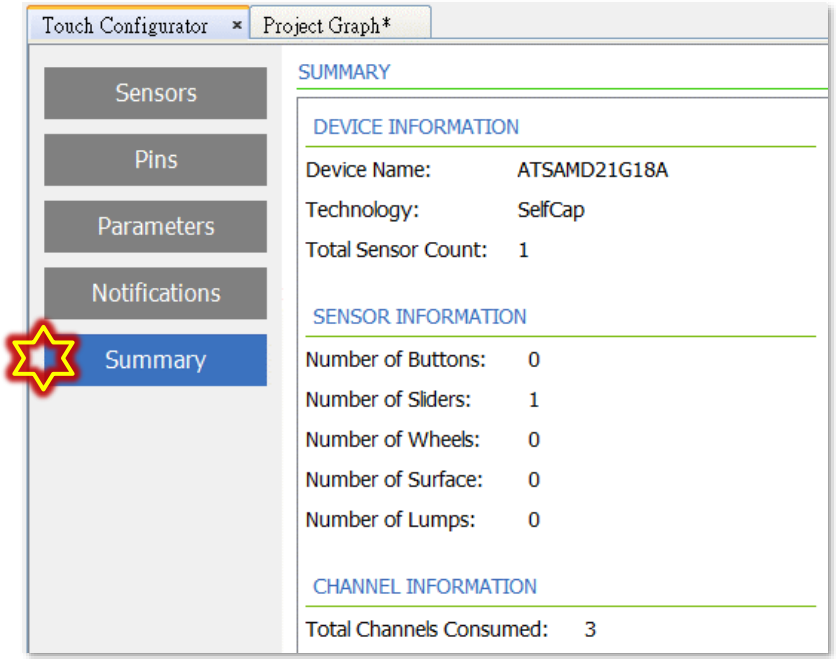
- **In Parameters Tab, disable Sensor AKS(Adjacent Keys Suppression) as below.**
  - Channel 0/1/2 : No AKS

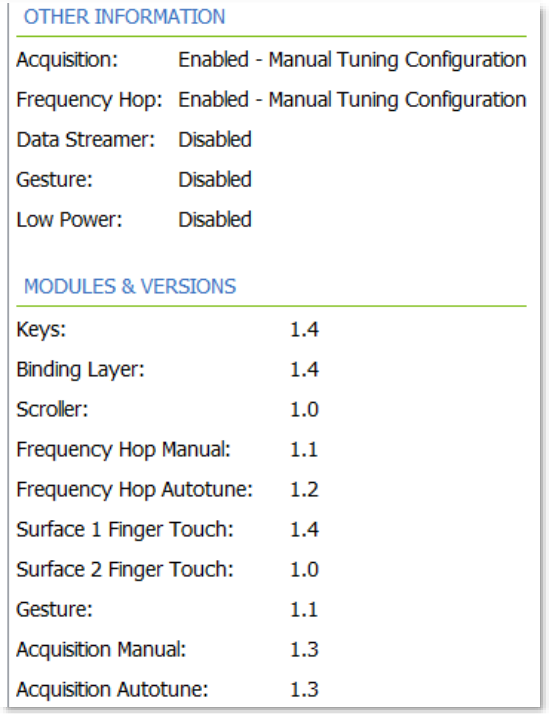


🛠 Lab1-2 Qtouch Slider

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







MICROCHIP

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

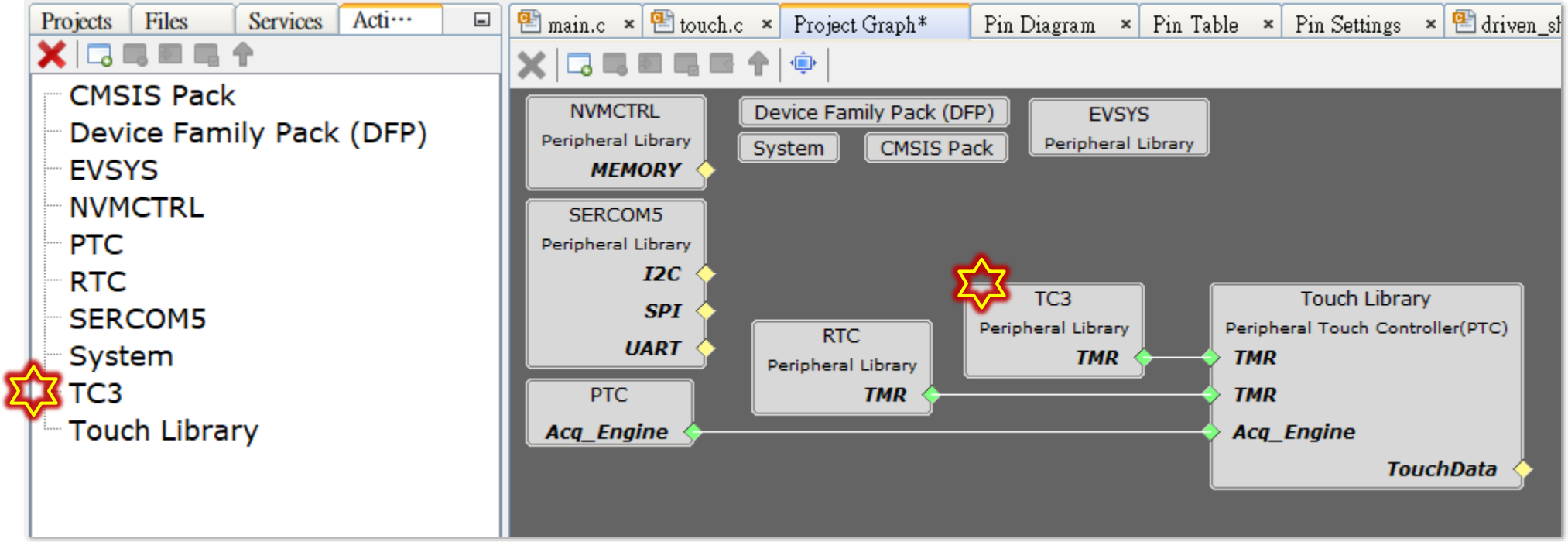
Order: Ports | Table View | Easy View

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

MICROCHIP

🛠 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 acquisition done flag
touch_process ()	Main processing function of touch library. This function initiates the acquisition, calls post processing after the acquisition complete and sets the flag for next measurement based on the sensor status.
touch_init ()	Initialization PTC, timer, and datastreamer modules of touch library.
touch_timer_config ()	Initialization necessary RTC timer for Touch in touch_init()
touch_sensors_config ()	Initialization of touch key sensors in touch_init()
get_sensor_node_signal ()	Get sensor node aquiction signals.
update_sensor_node_signal ()	Update sensor node aquiction signals.
get_sensor_node_reference ()	Get sensor node channel reference.
update_sensor_node_reference ()	Update sensor node channel reference.
get_sensor_cc_val ()	Get sensor node compansation capacitor value.
update_sensor_cc_val ()	Update sensor node compansation capacitor value.
get_sensor_state ()	Get sensor node state
update_sensor_state ()	Update sensor node state
calibrate_node ()	Sensor node calibration
get_scroller_state	Get scroller sensor state
get_scroller_position	Get scroller position

API functions in driven_shield.c

drivenshield_configure ( )	Iinitialization Driven Shield for Touch in touch_init()

MICROCHIP

🛠 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("____________________");
    ...
```

MICROCHIP

🛠 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("O");
                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

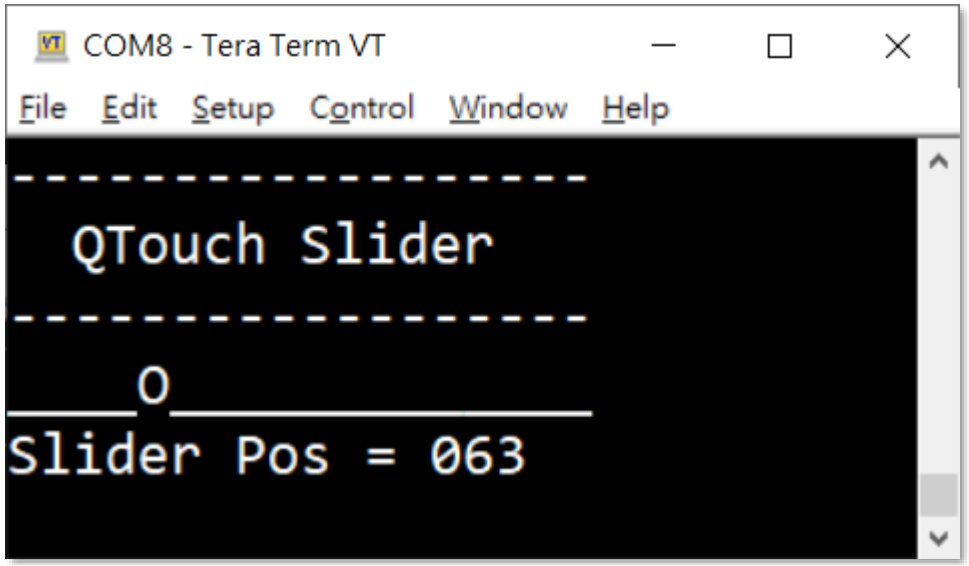
- **Implement manual Touch Calibration feature.**

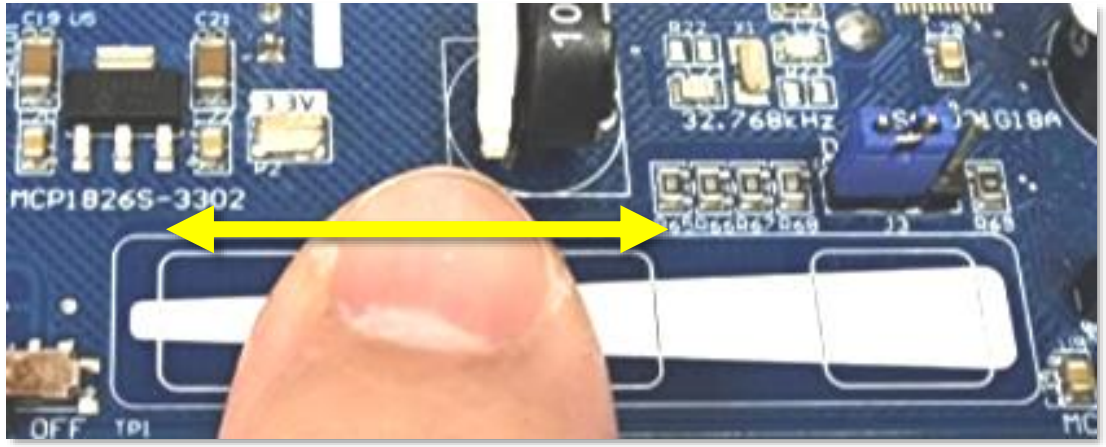
```
int main ( void )
{
    uint16_t ScrollPos, PreScrollPos=256;
    ...

    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("O");
                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;
                ...
```

MICROCHIP

🛠 Lab1-2 Qtouch Slider

- **Result of Touch Slider(Scroller) implementation.**



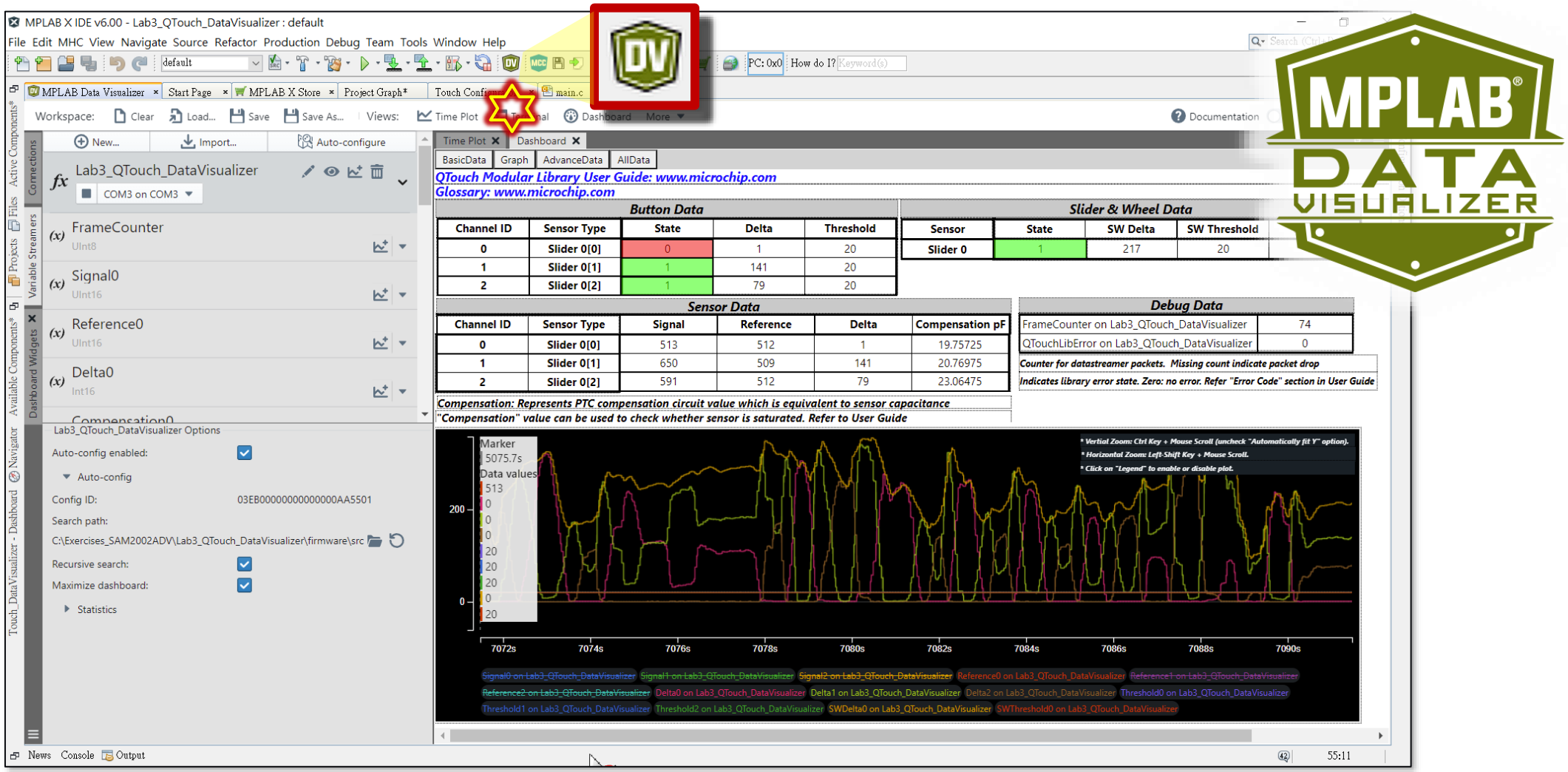


MICROCHIP

🛠 Lab1-3 Data Visualizer for Touch

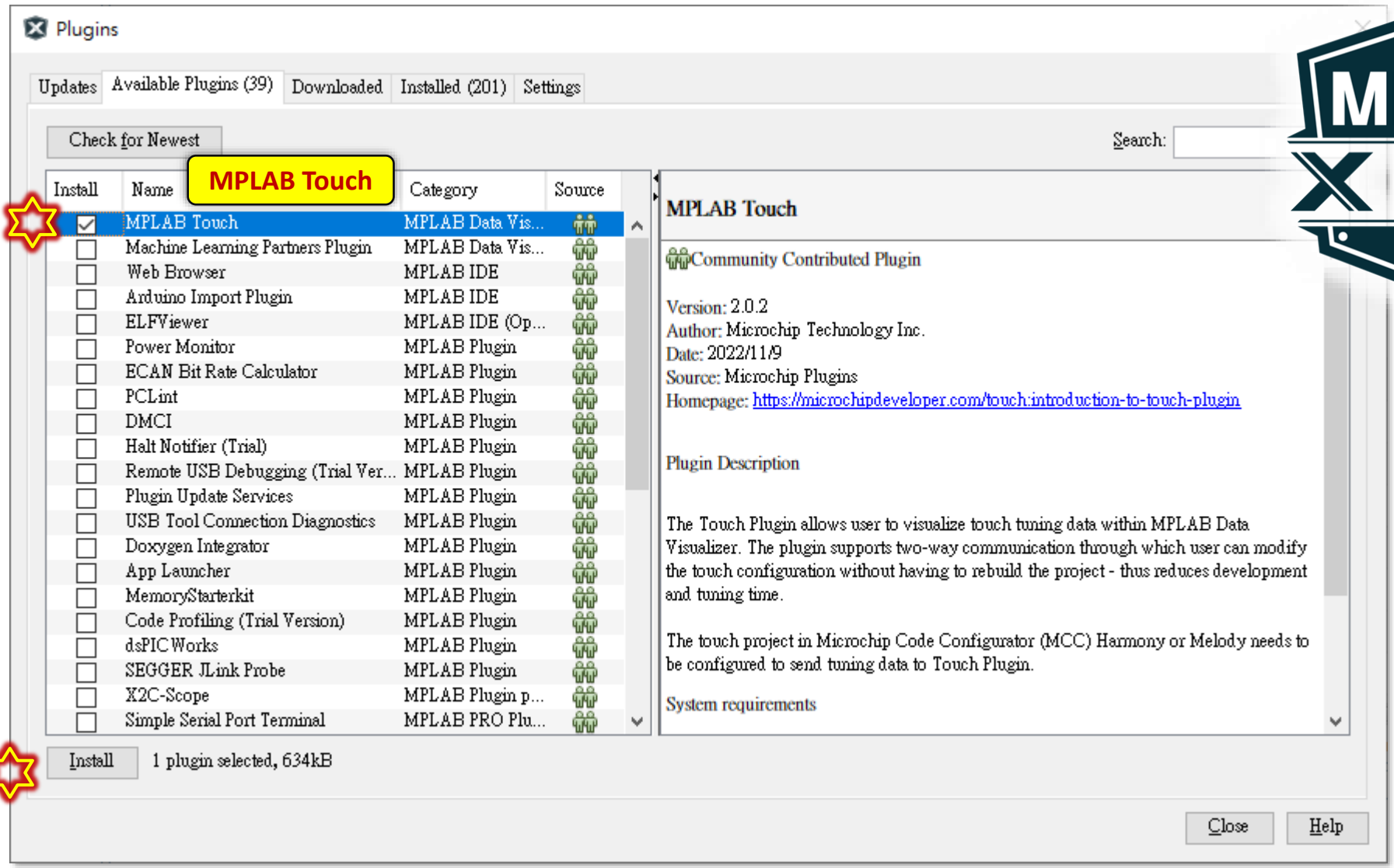
Lab1-3 Data Visualizer for Touch

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



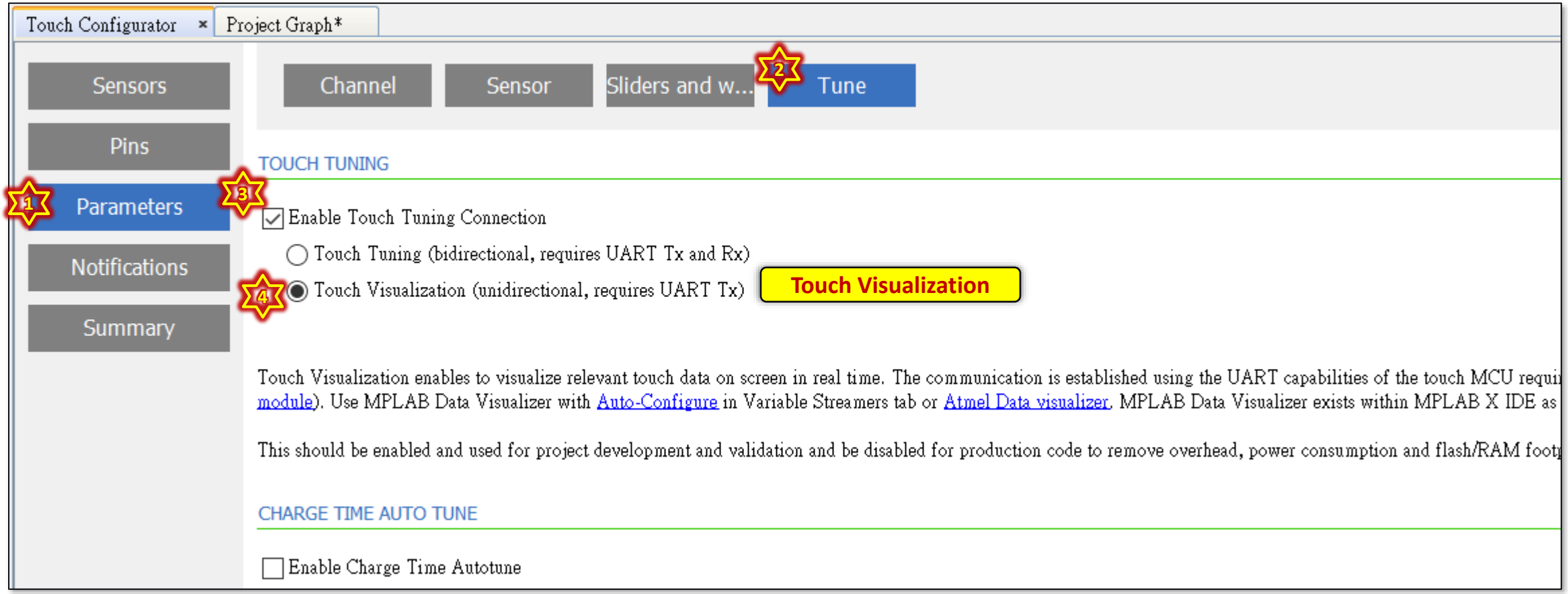
Lab1-3 Data Visualizer for Touch

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



🛠 Lab1-3 Data Visualizer for Touch

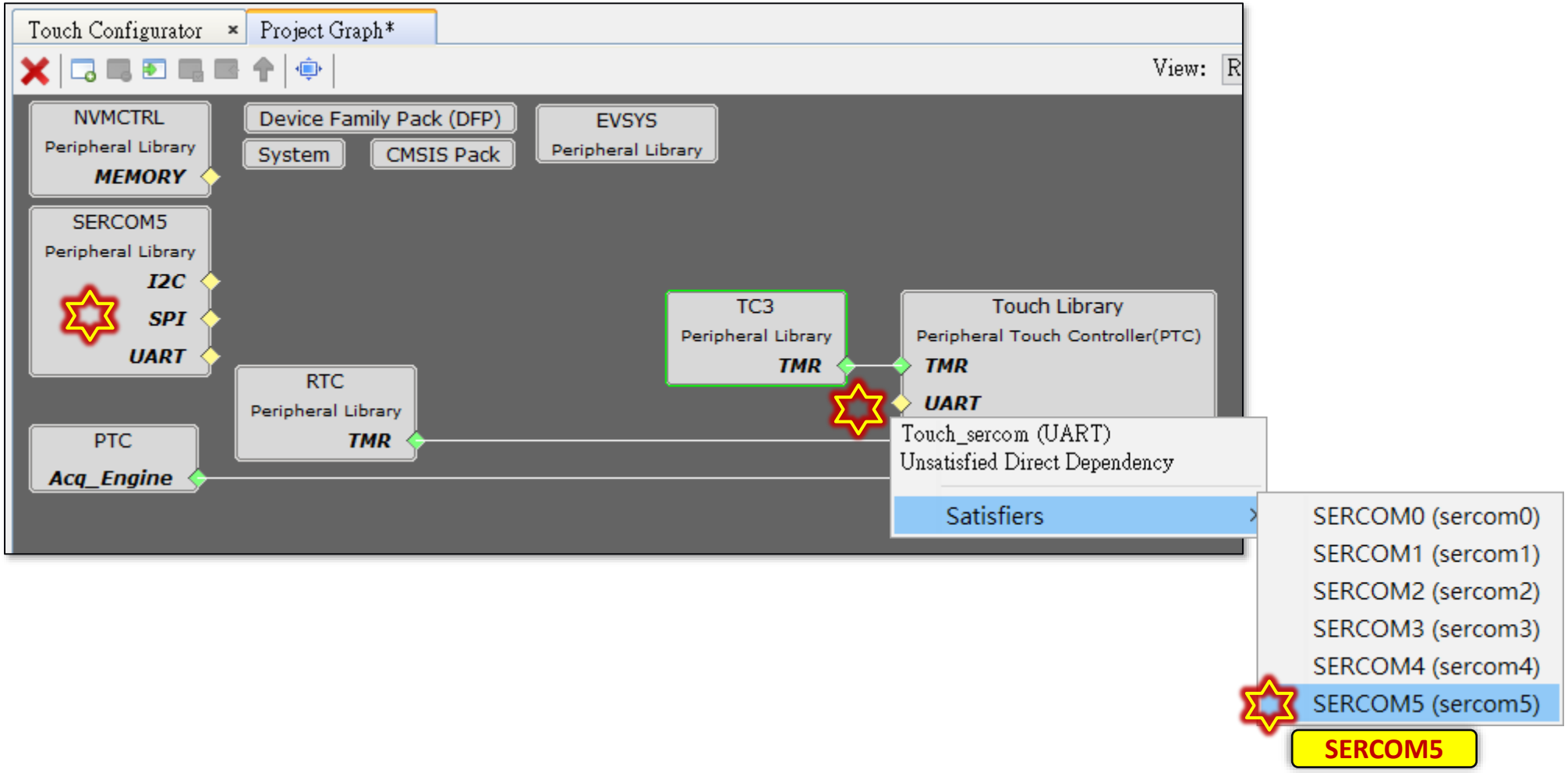
- Return Harmony Touch Configurator if you would like continue from Lab1-2.
- In Parameters Tab, please click on Tune.
- Enable Touch Tuning Connection and select to Touch Visualization (unidirectional…)



MICROCHIP

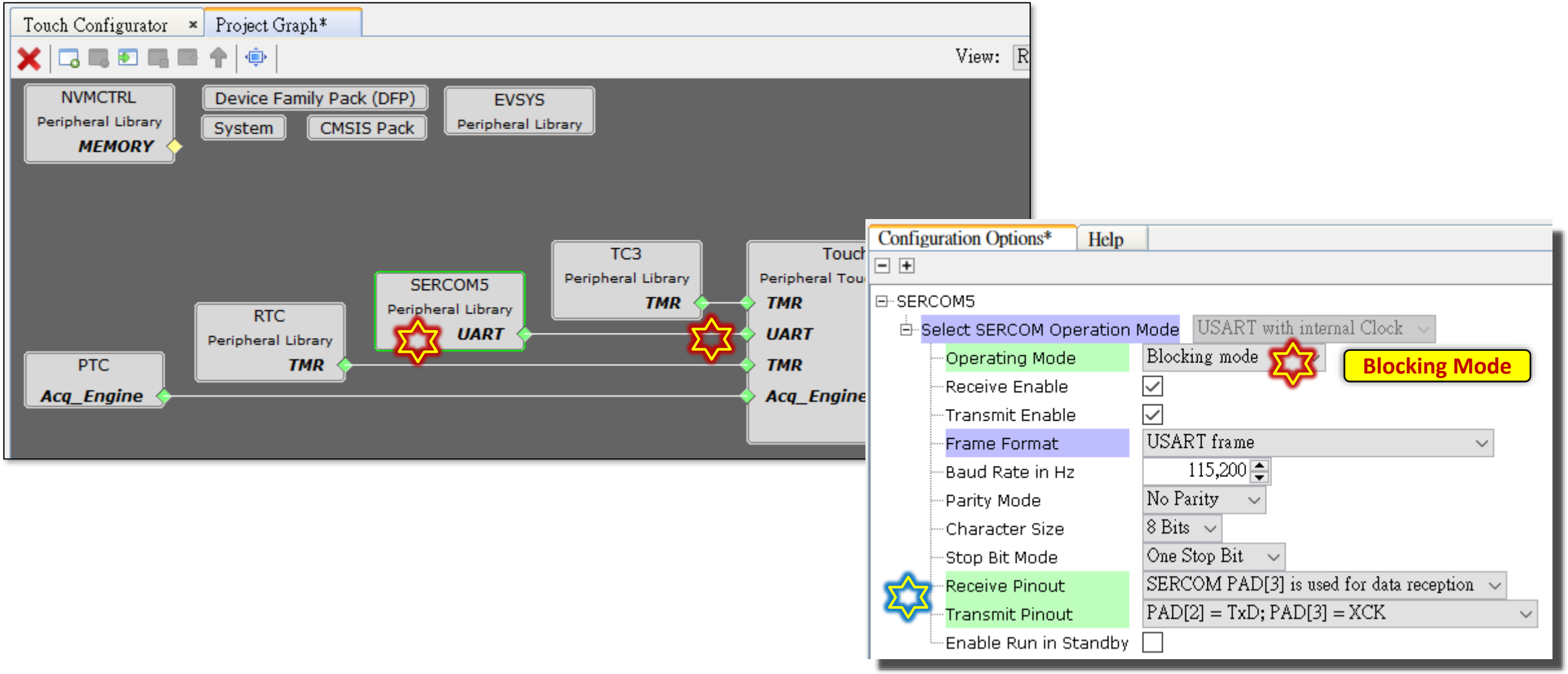
Lab1-3 Data Visualizer for Touch

- 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.



🛠 Lab1-3 Data Visualizer for Touch

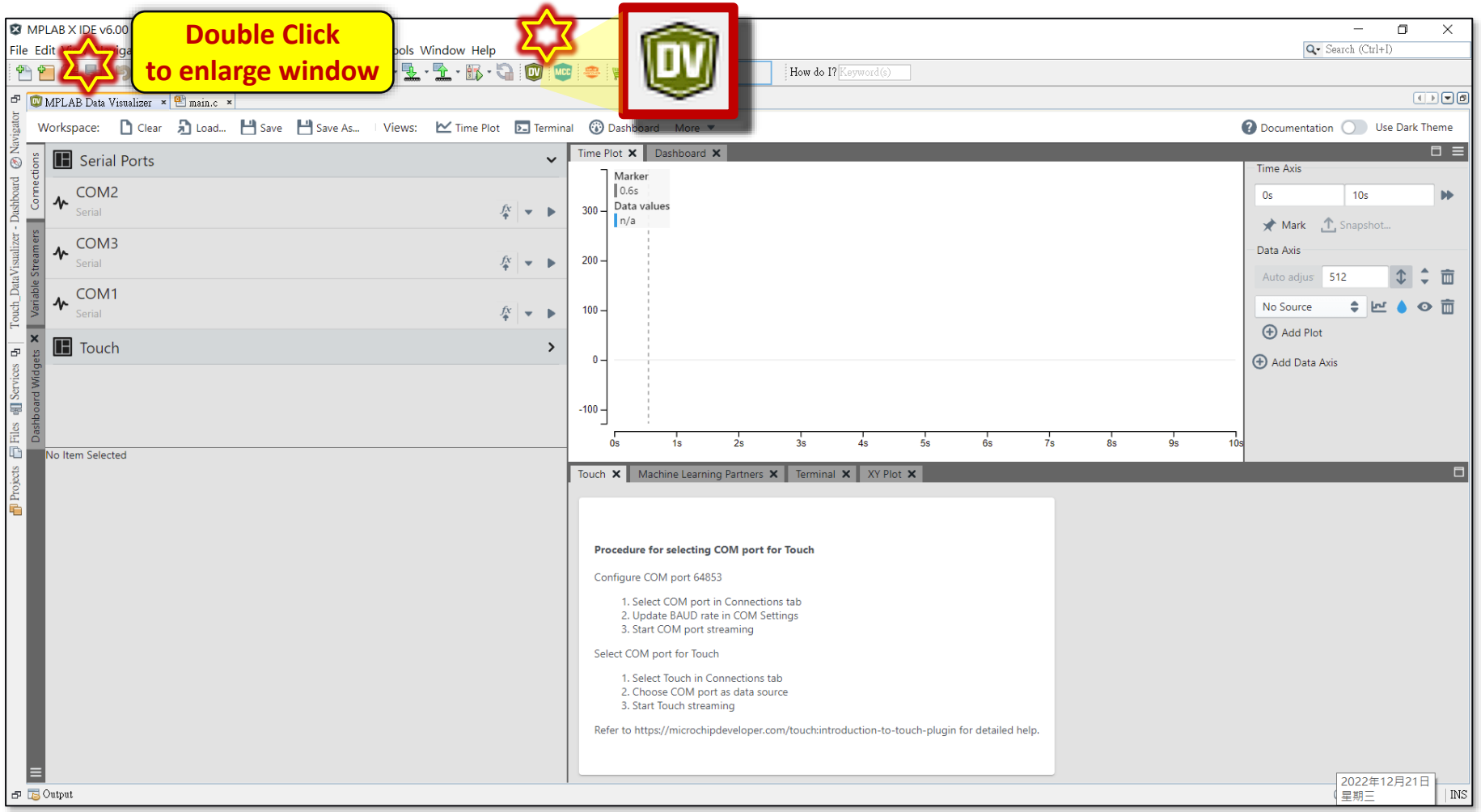
- Modify myprintf() to use TransmitComplete() function of SERCOM Blocking Mode and common out all debug myprintf() in while loop of main.c.

```
...

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("O");
                //myprintf("\033[5;1H");
                //myprintf("Slider Pos = %03d", ScrollPos);

                switch( ScrollPos*4/255 )
```

MICROCHIP

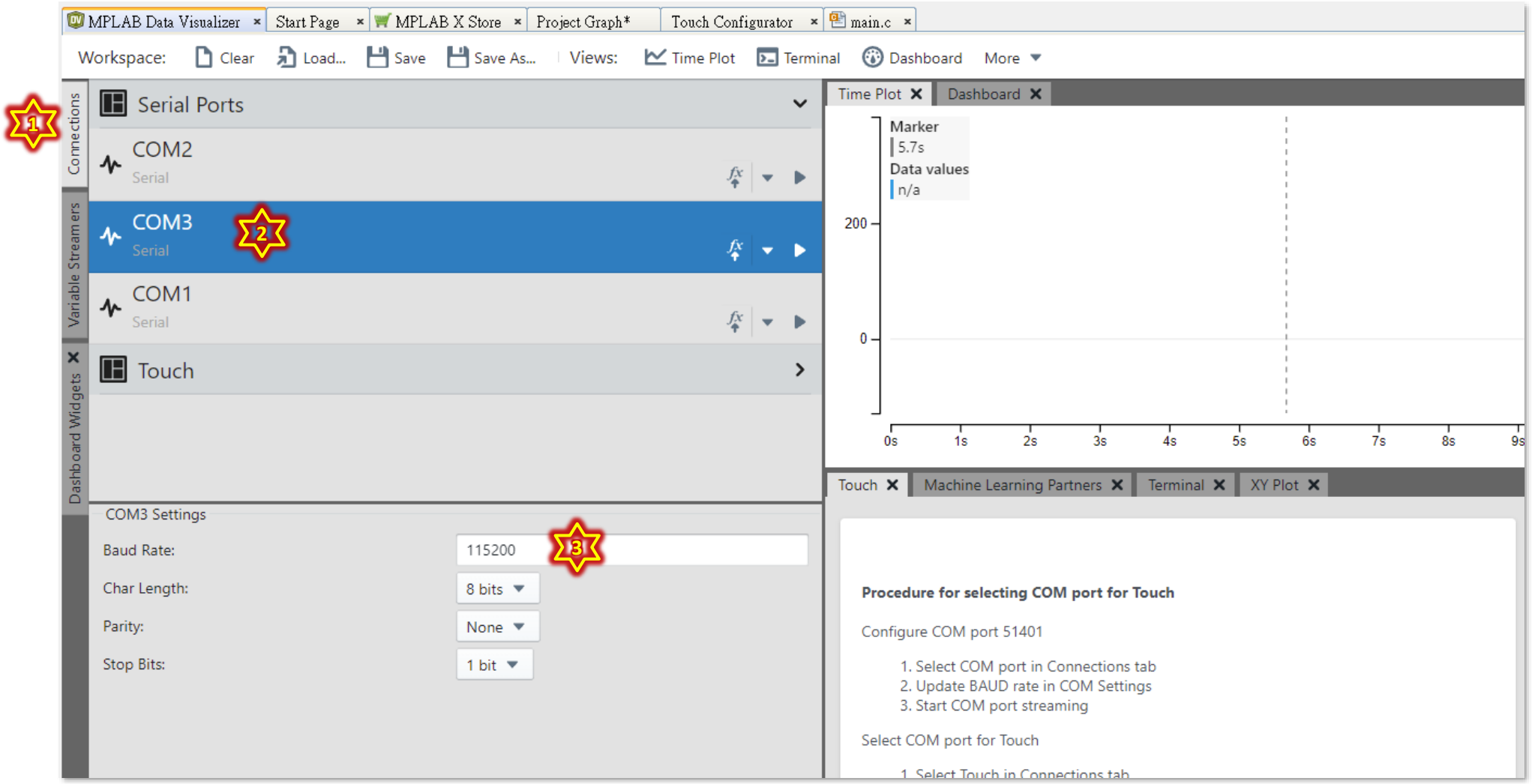
Lab1-3 Data Visualizer for Touch

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



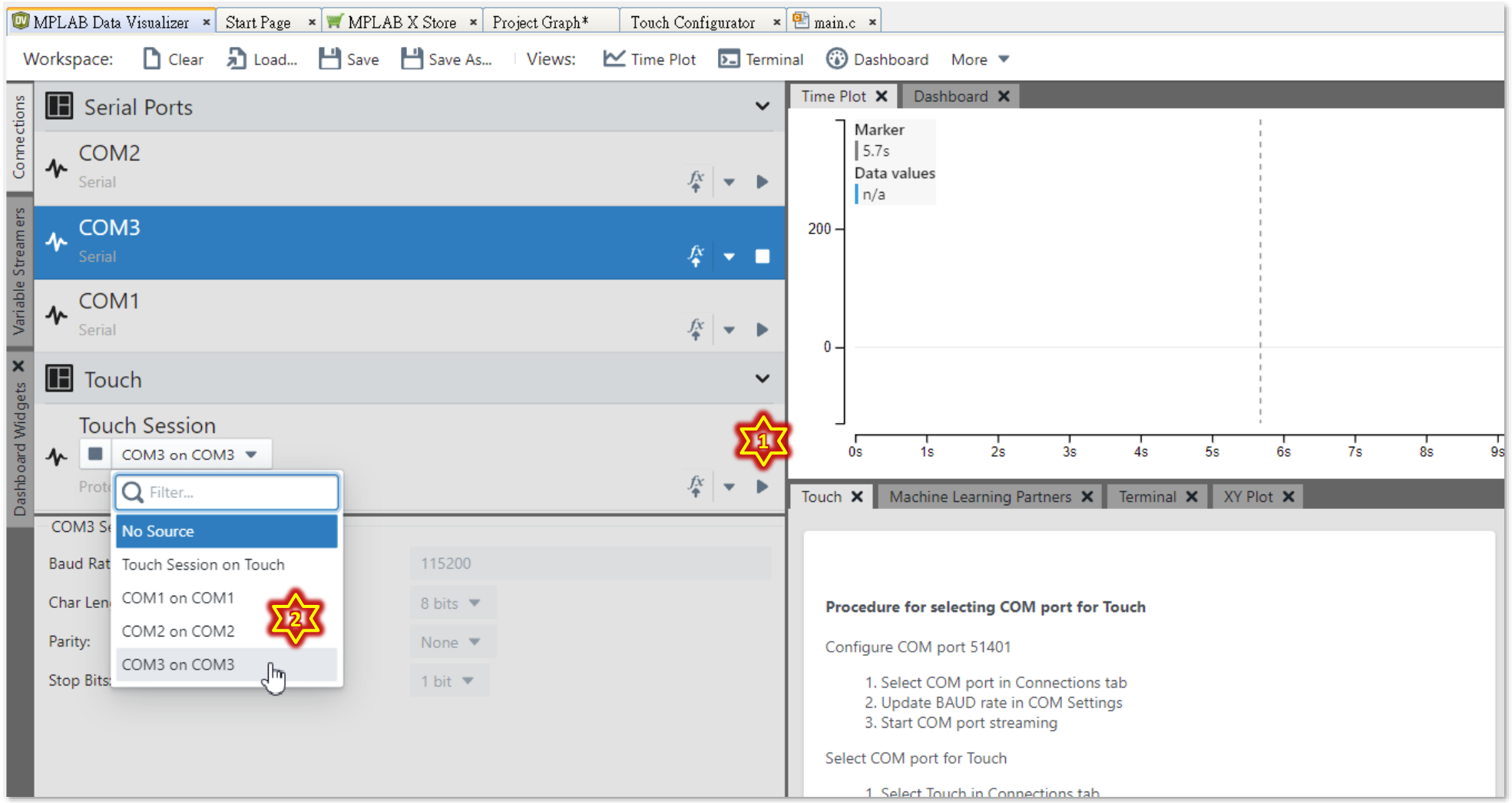
🛠 Lab1-3 Data Visualizer for Touch

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



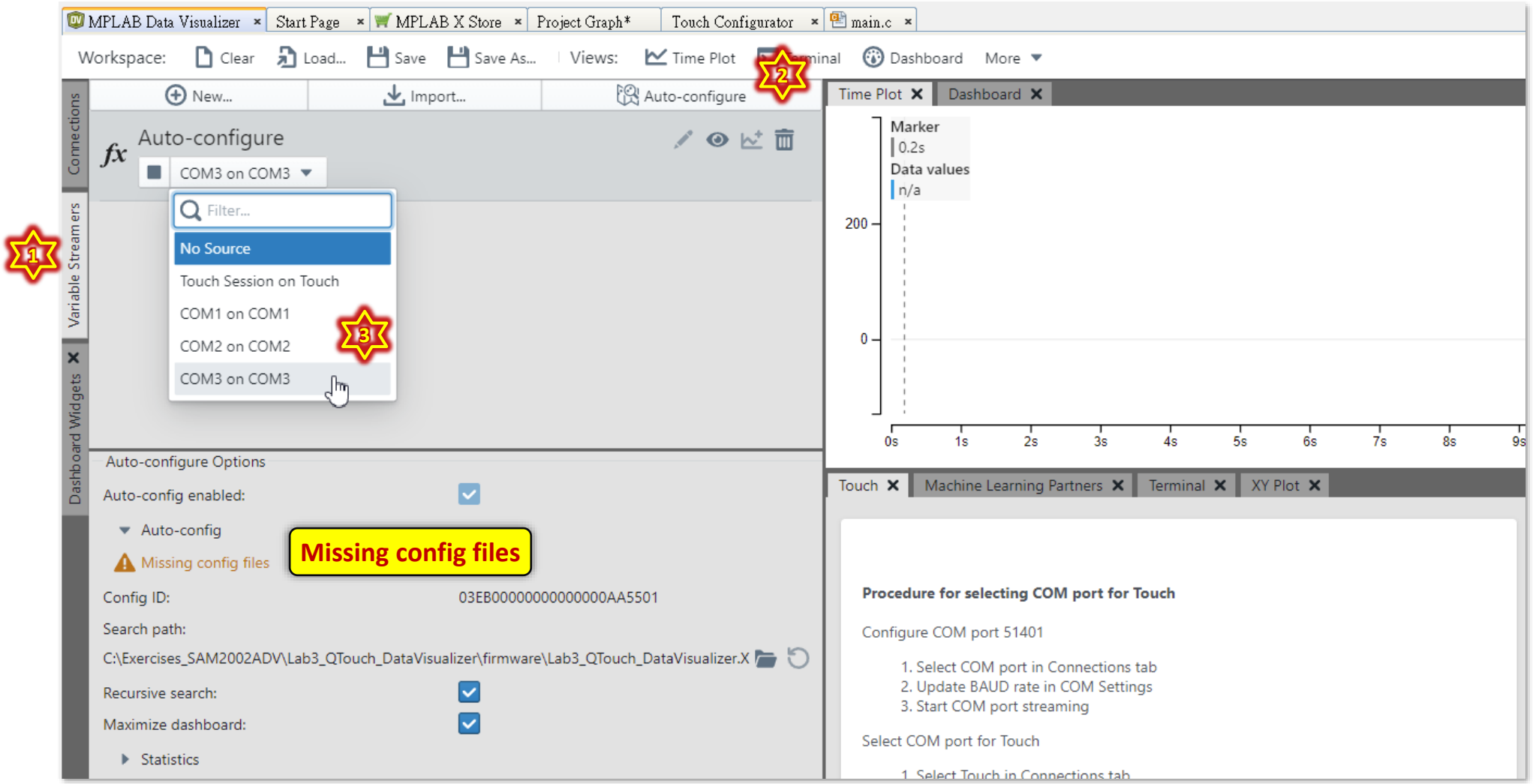
Lab1-3 Data Visualizer for Touch

- 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.



🛠 Lab1-3 Data Visualizer for Touch

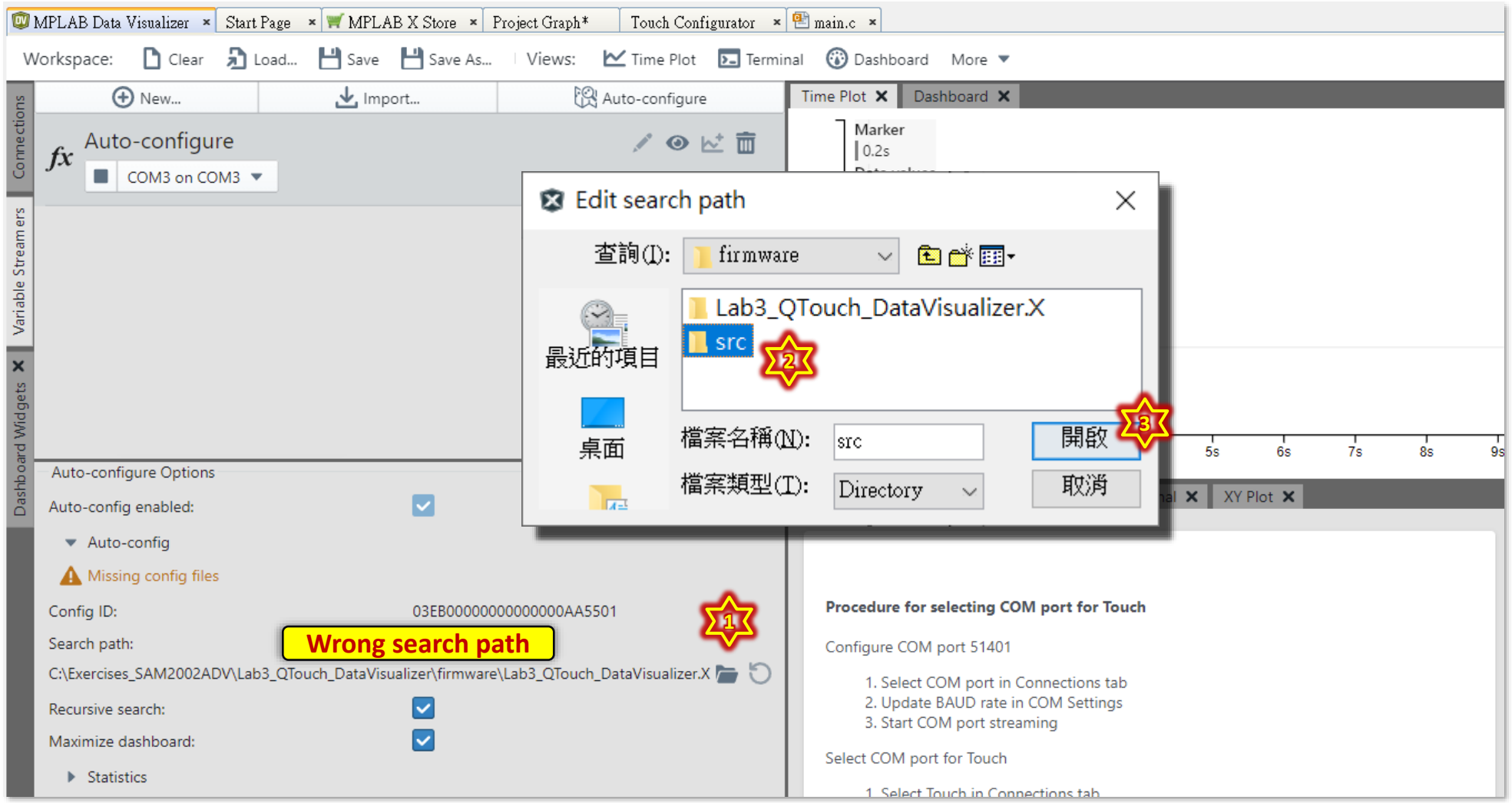
- Click to Variable Stream ers, then click on **Auto-configure** to add a new Streamer.
- Please select the same **COM Port** as Auto-configuration source.



MICROCHIP

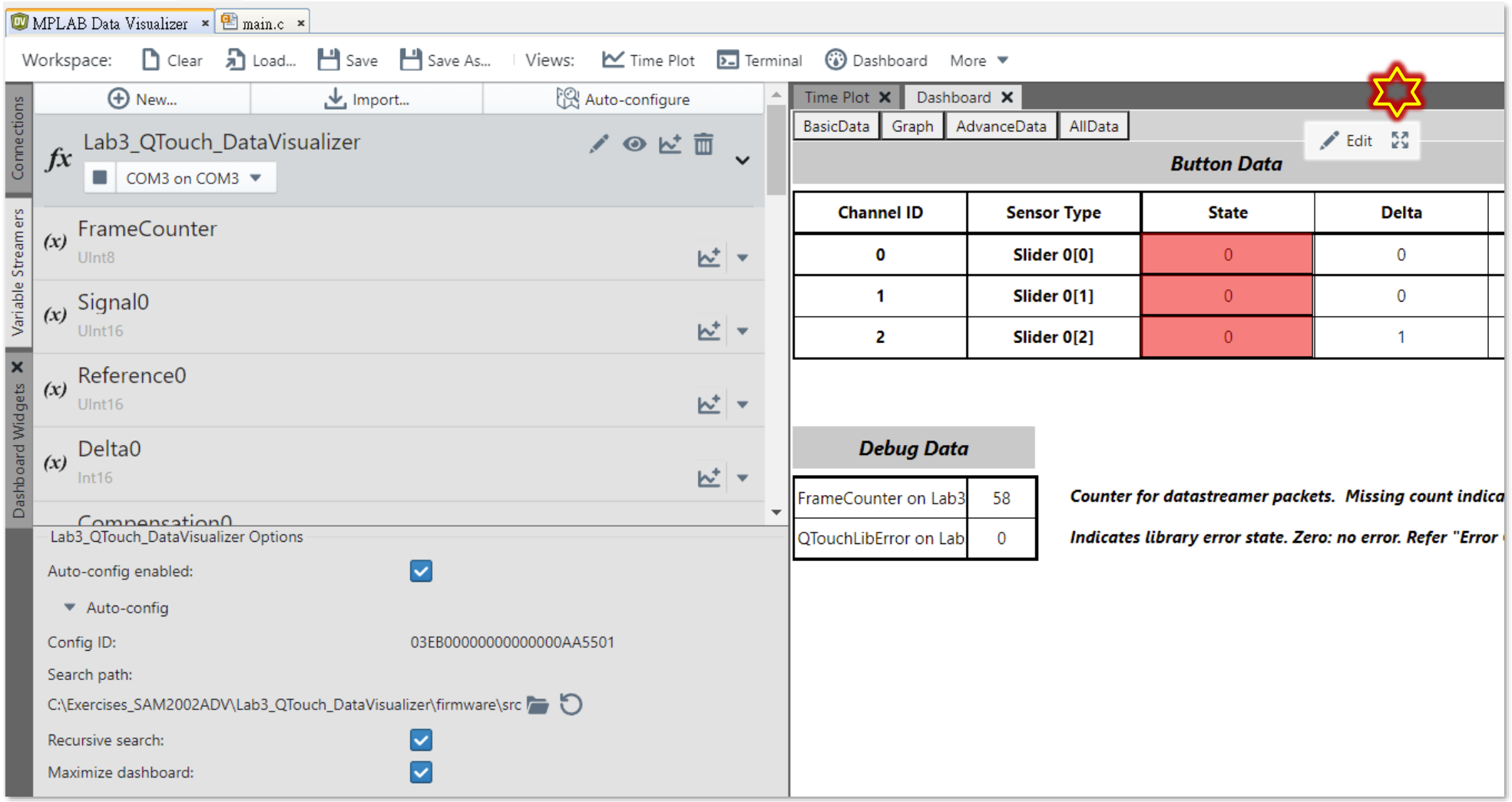
Lab1-3 Data Visualizer for Touch

- **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.**



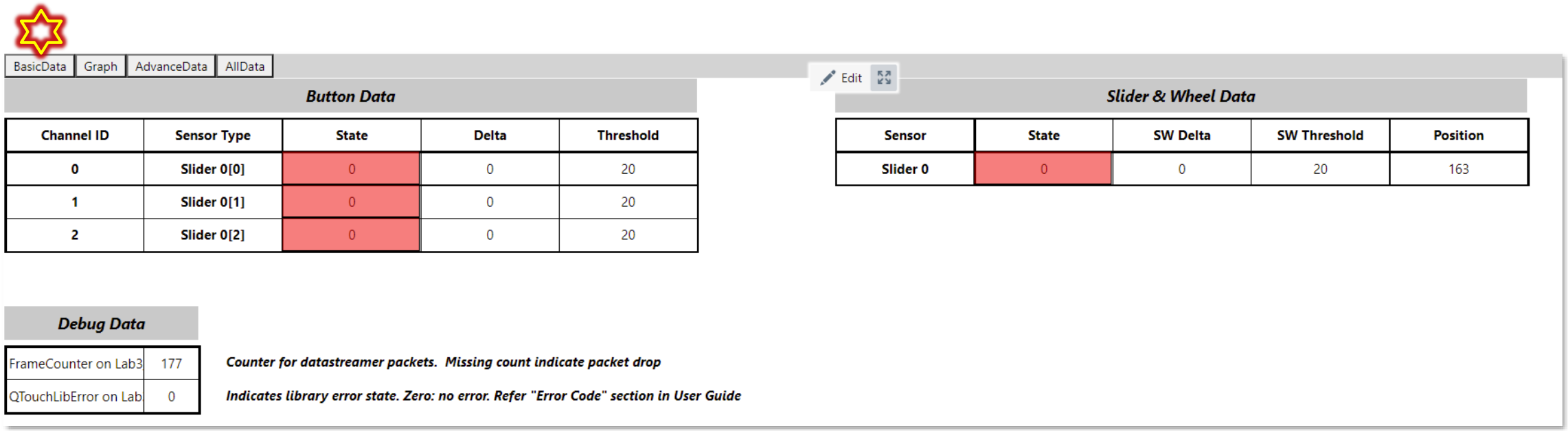
Lab1-3 Data Visualizer for Touch

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



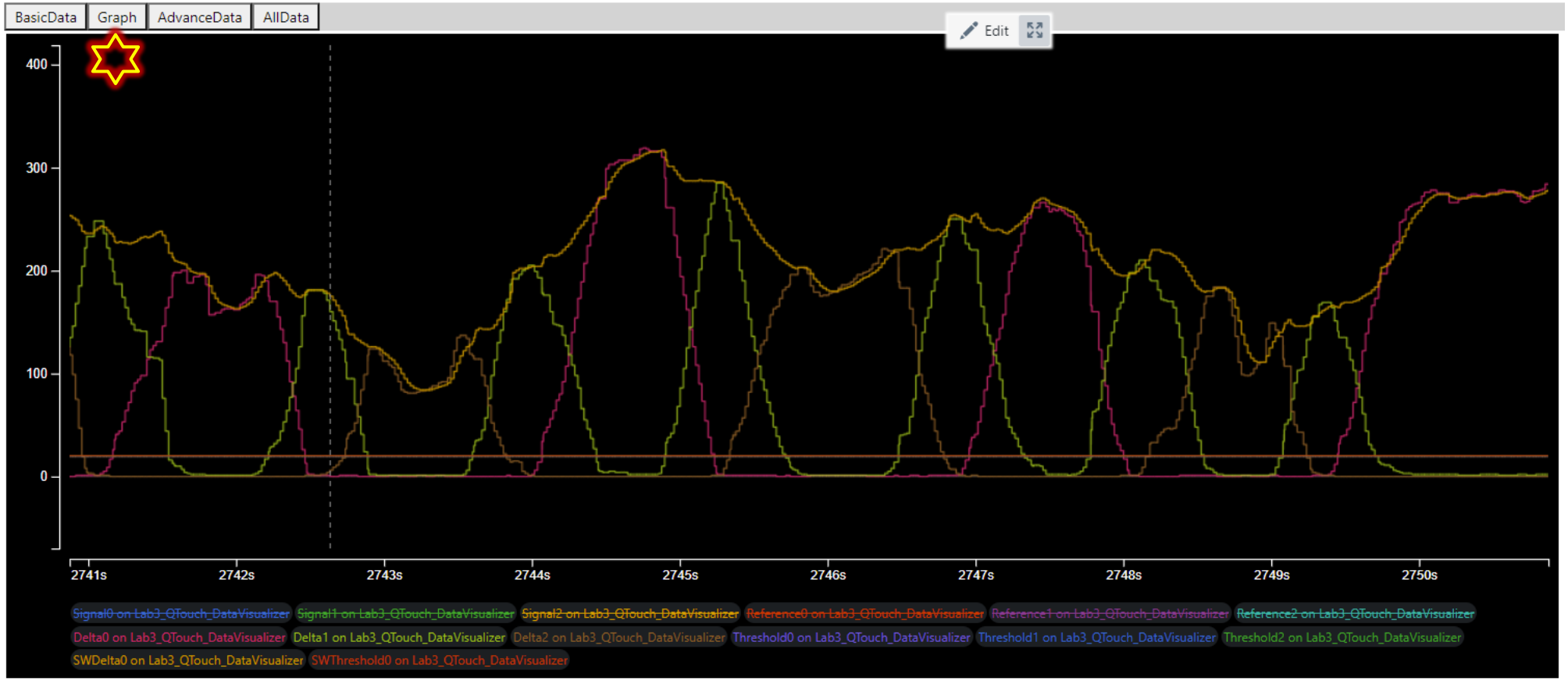
🛠 Lab1-3 Data Visualizer for Touch

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



🛠 Lab1-3 Data Visualizer for Touch

- **Graph** window of Touch Dashboard.



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

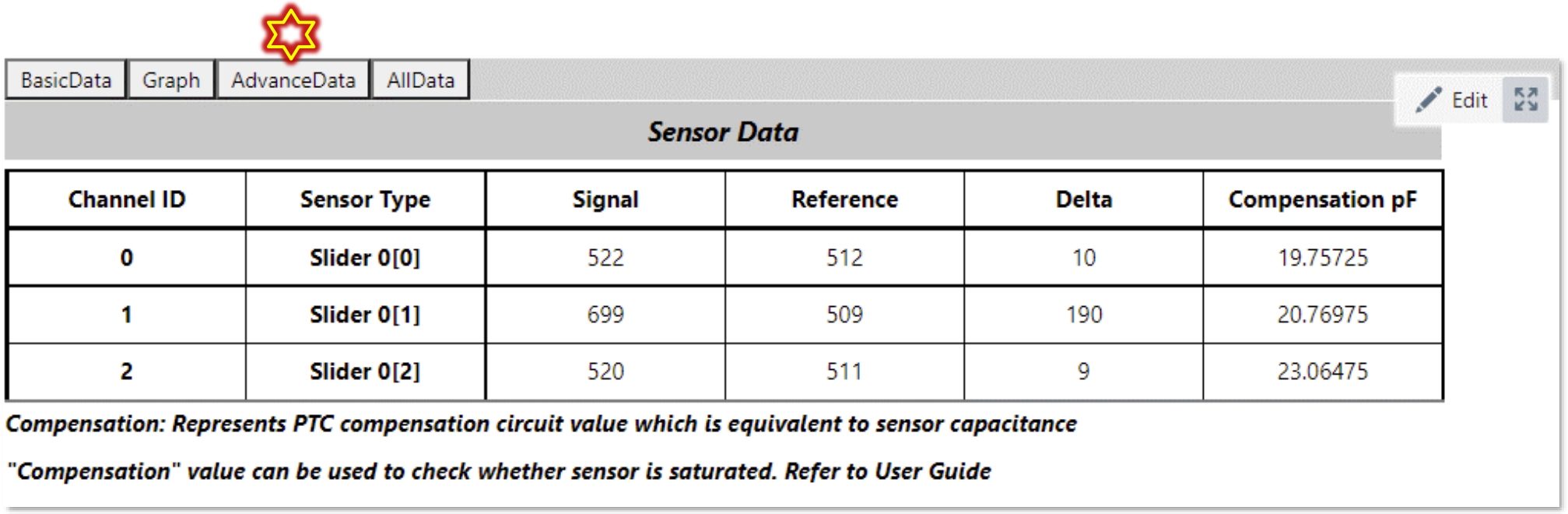
** Horizontal Zoom: Left-Shift Key + Mouse Scroll.*

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

MICROCHIP

🛠 Lab1-3 Data Visualizer for Touch

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



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.

BasicData | Graph | AdvanceData | AllData

Edit

QTouch Modular Library User ... www.microchip.com

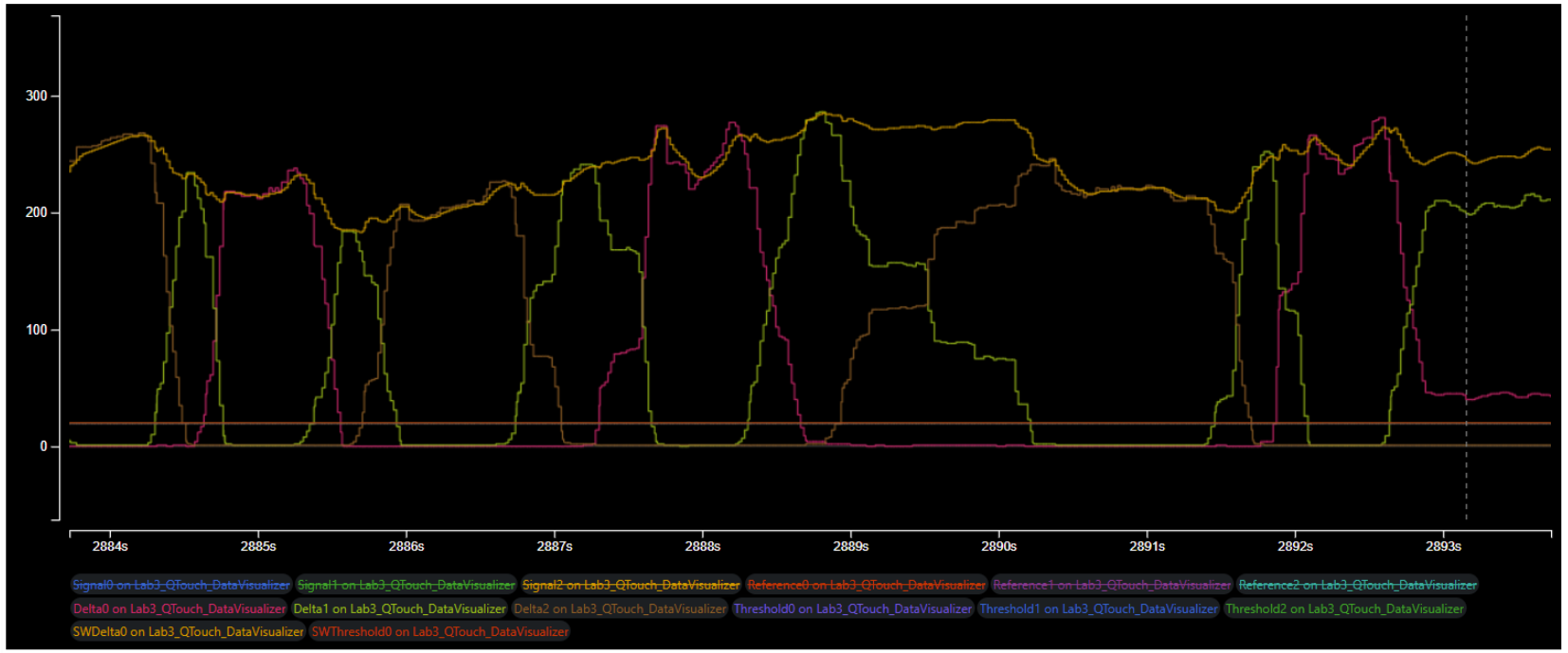
Glossary: www.microchip.com

Button Data

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

Slider & Wheel Data

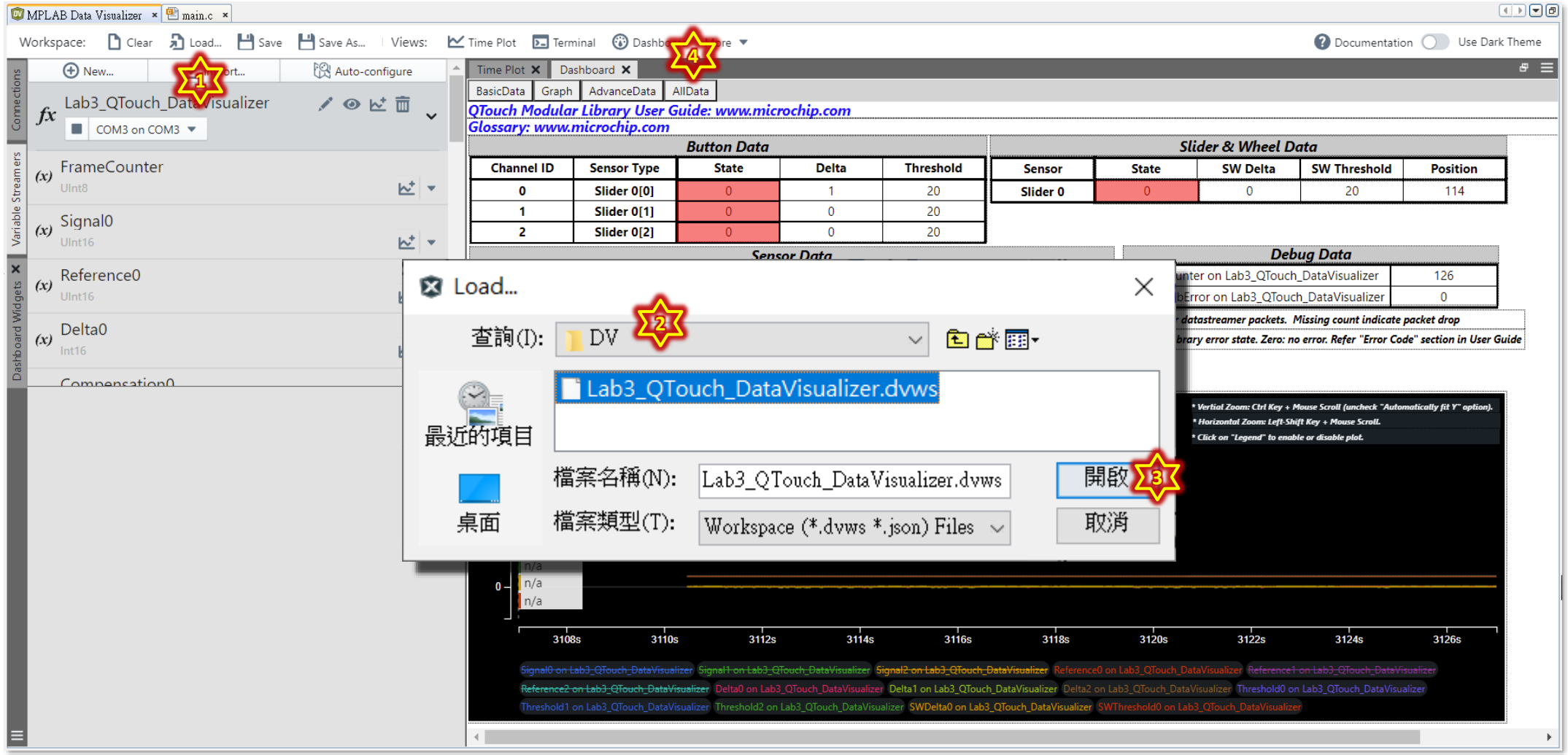
Sensor	State	SW Delta	SW Threshold	Position
Slider 0	0	0	20	114



MICROCHIP

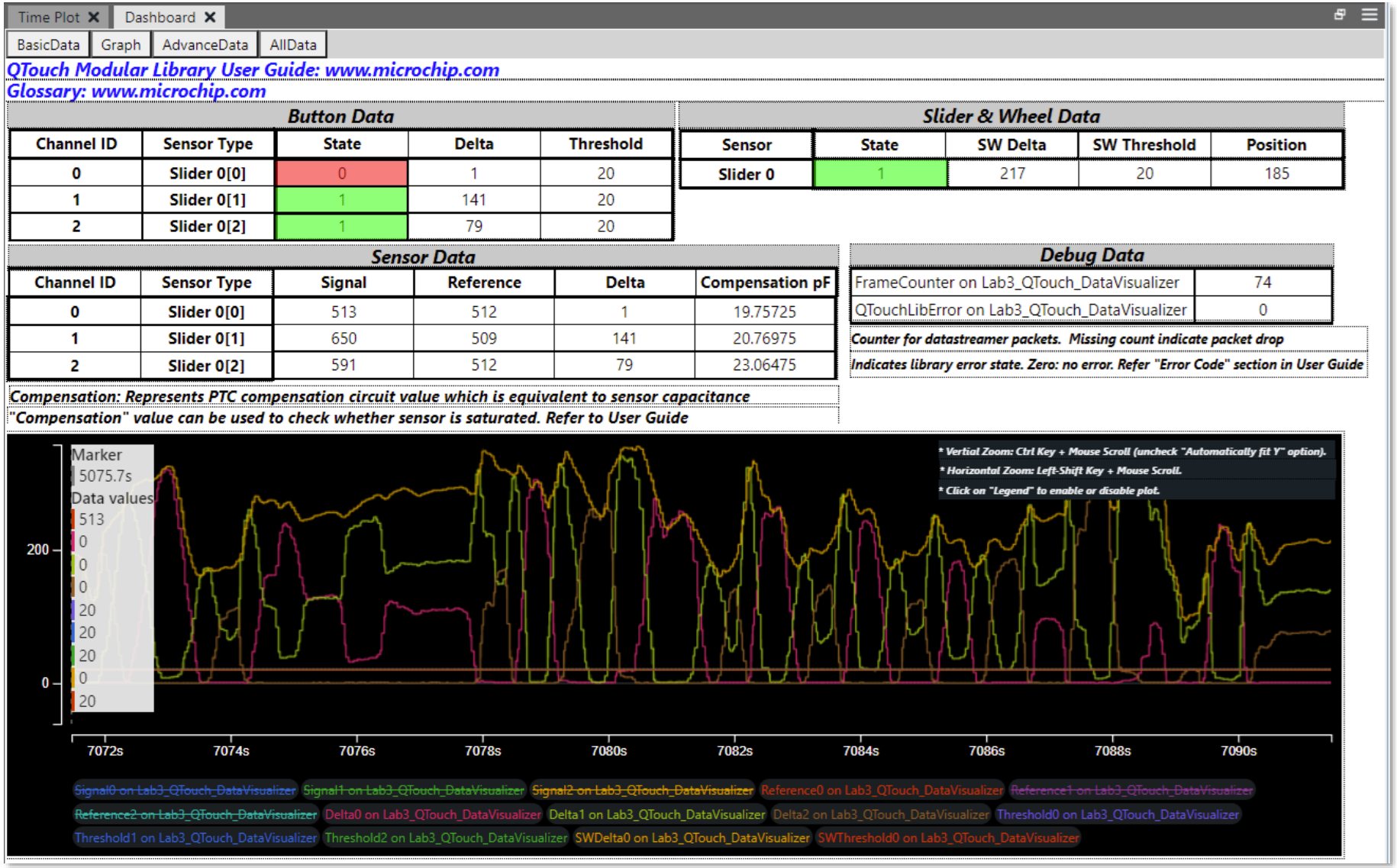
Lab1-3 Data Visualizer for Touch

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



Lab1-3 Data Visualizer for Touch

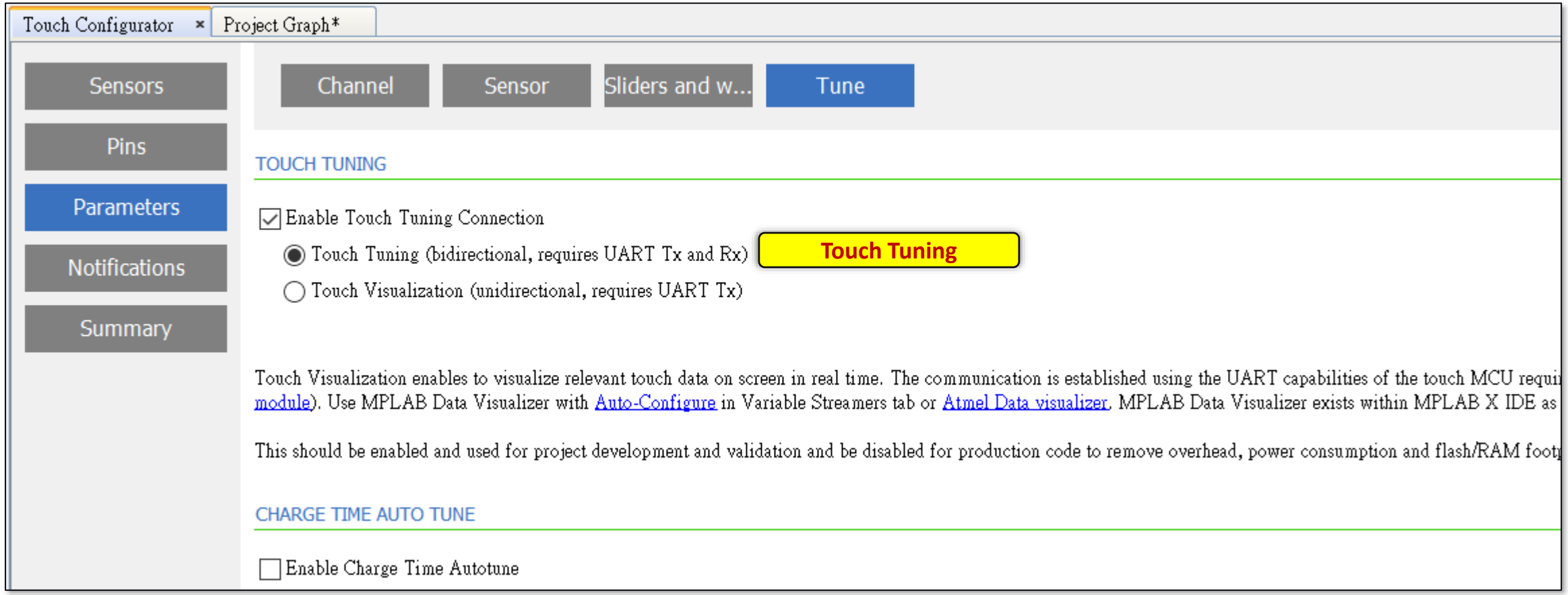
- The customized **All Data** of Dashboard



🛠 Lab1-3 Data Visualizer for Touch

Discuss

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

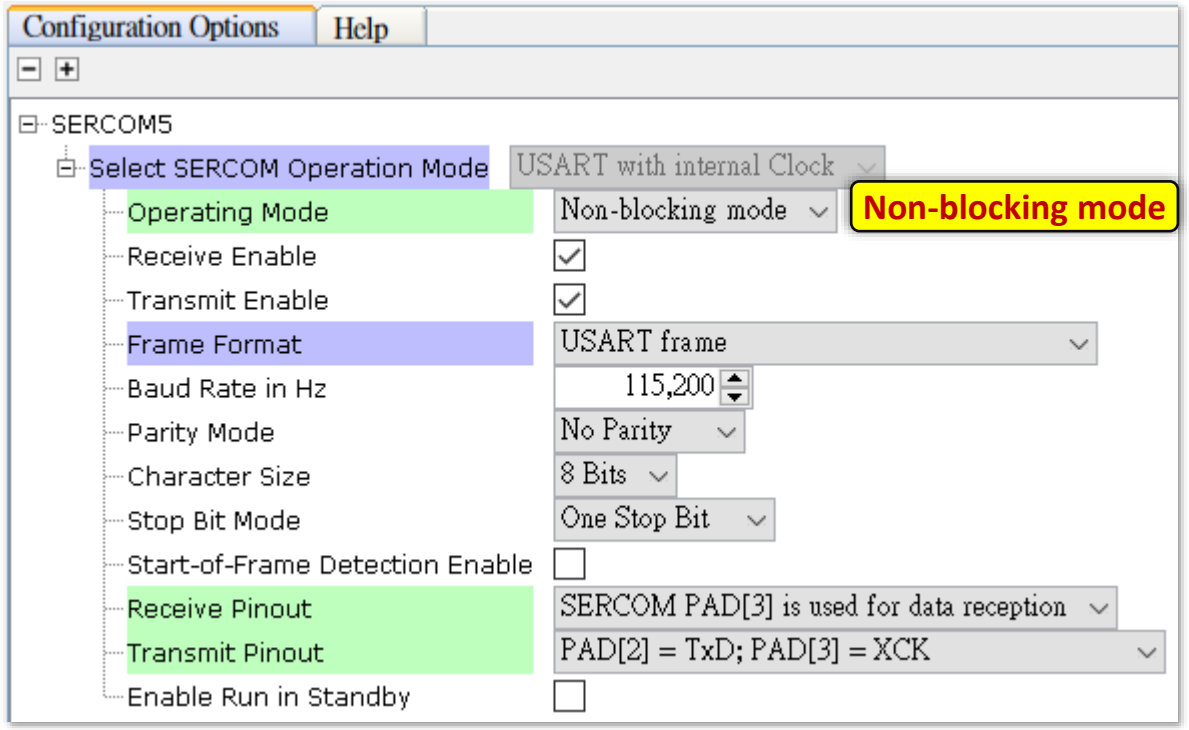


MICROCHIP

🛠 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() );
    }
}
```

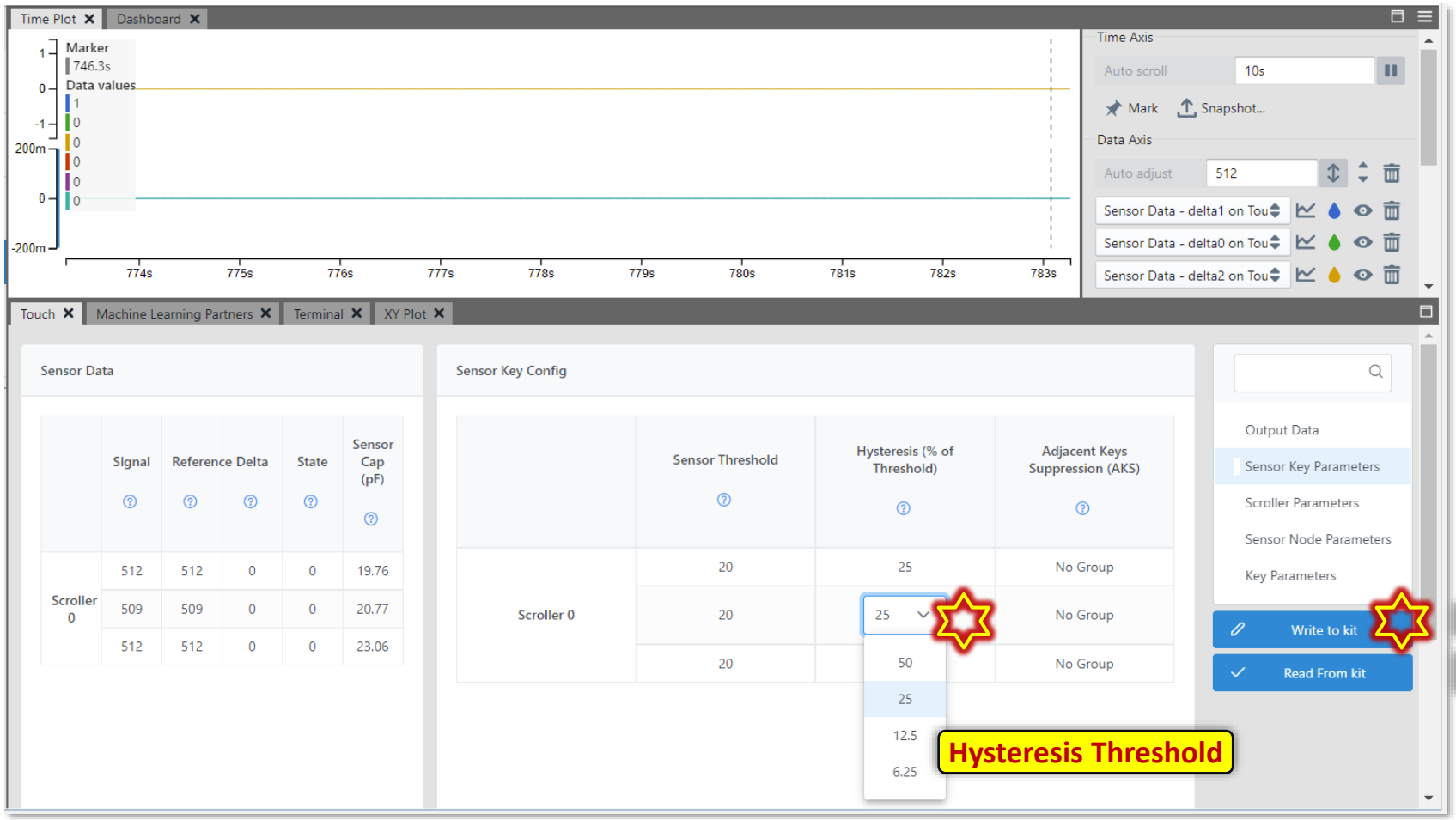
Non-blocking mode function

MICROCHIP

🛠 Lab1-3 Data Visualizer for Touch

Discuss

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



MICROCHIP

USB Fundamentals

USB, Universal Serial Bus

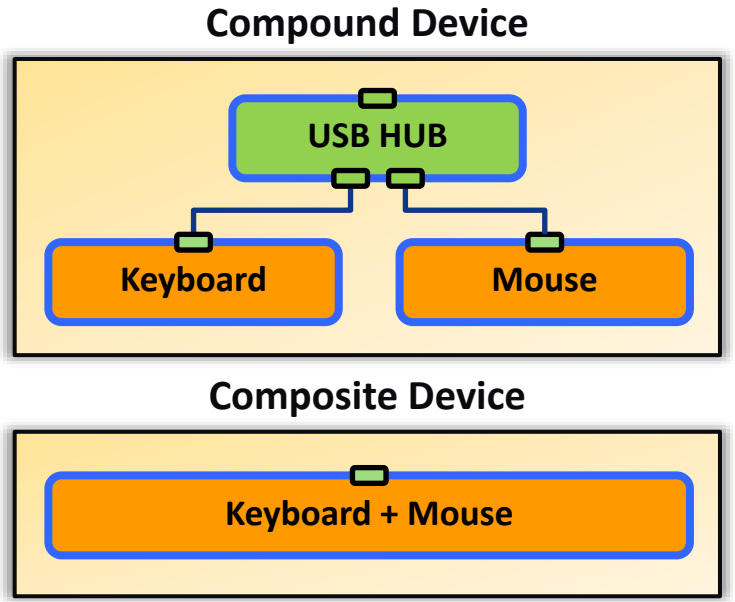
- **USB was co-developed by a group of companies, Compaq, Intel, Microsoft and NEC etc. Who wanted to make it much easier to add/remove peripheral devices from PCs**
- **History**
  - **USB 0.7 – 1994 -> USB 1.0 - Jan. 1996 -> USB 1.1 - Sep. 1998 ->**
  - **USB 2.0 - Apr. 2000**
    - USB 1.0 - > Low 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

MICROCHIP

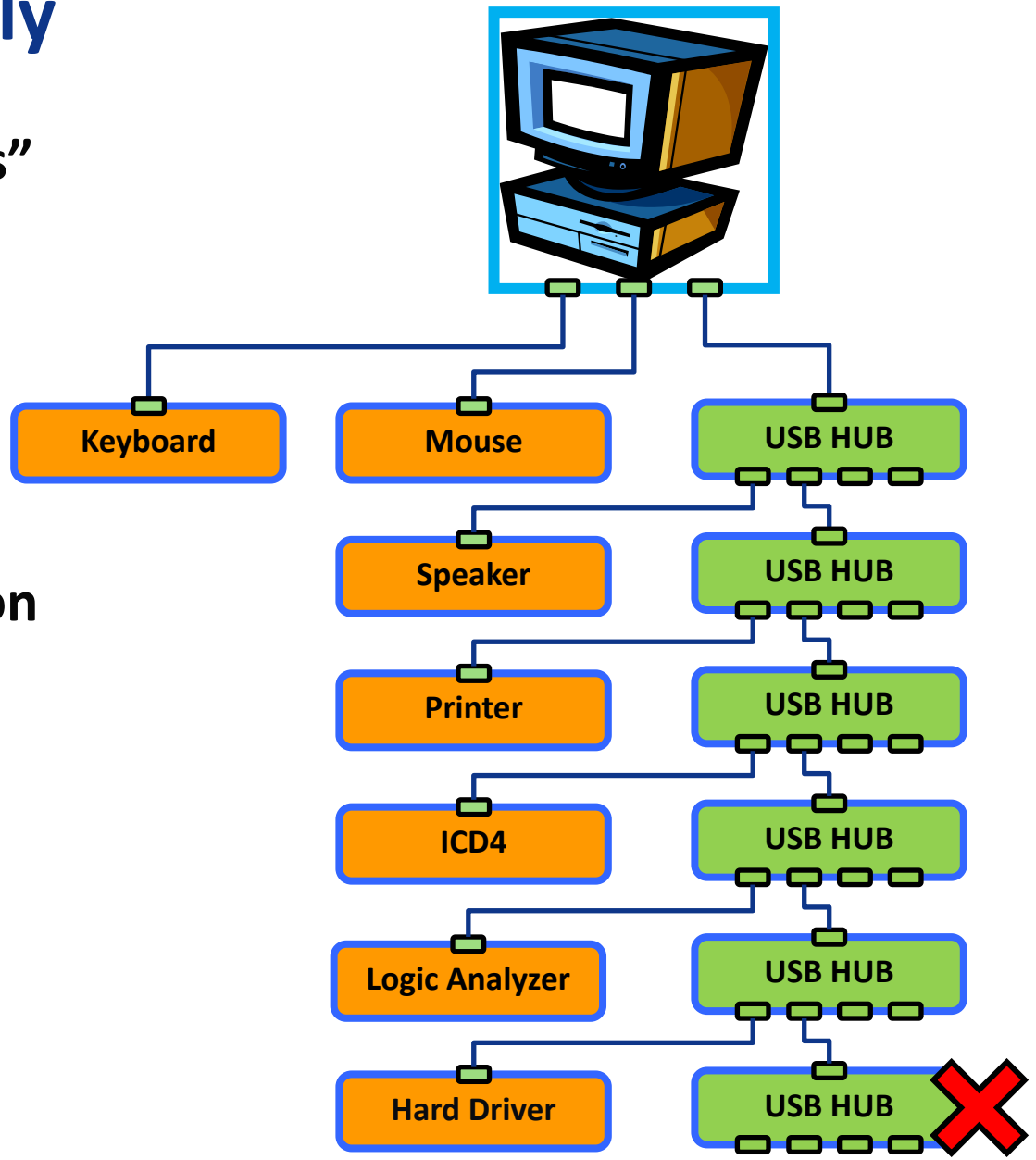
USB Device Types

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



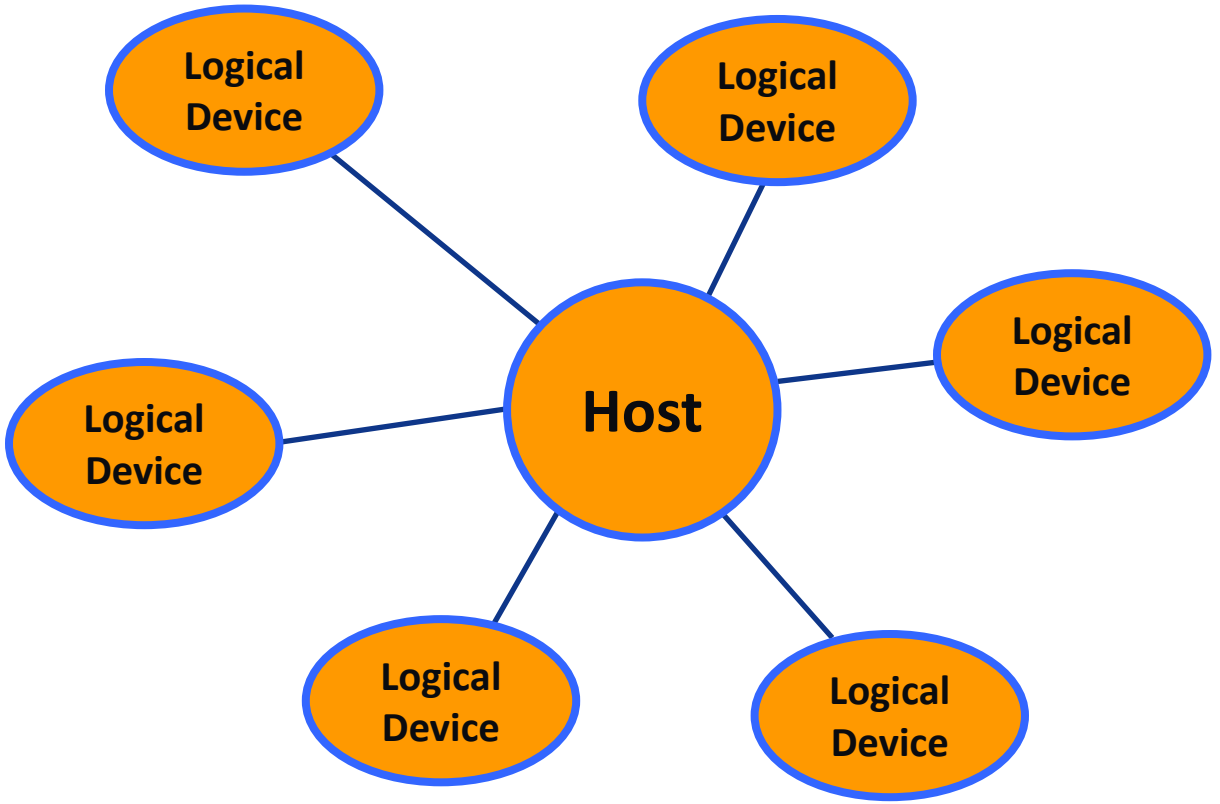
Tiered Star Topology, Physically

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



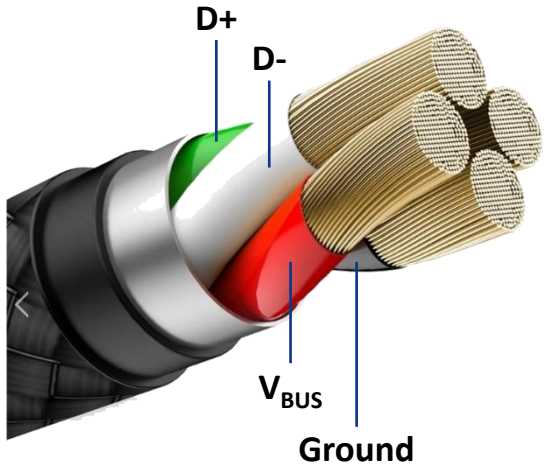
Star Topology, Logical

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



USB 2.0 cable and Physical Interface

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



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

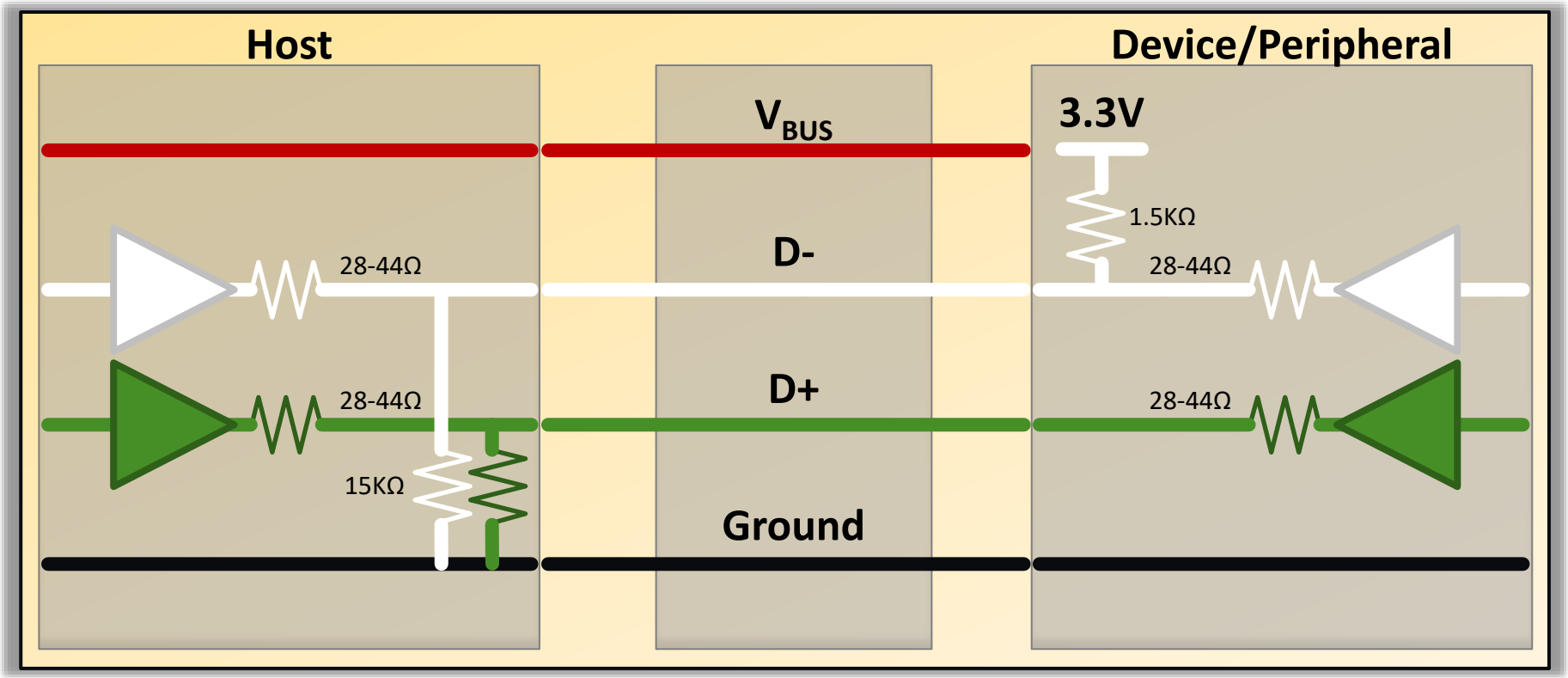
VBUS	~ 5.0V
D+/D-	~ 3.3V

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

MICROCHIP

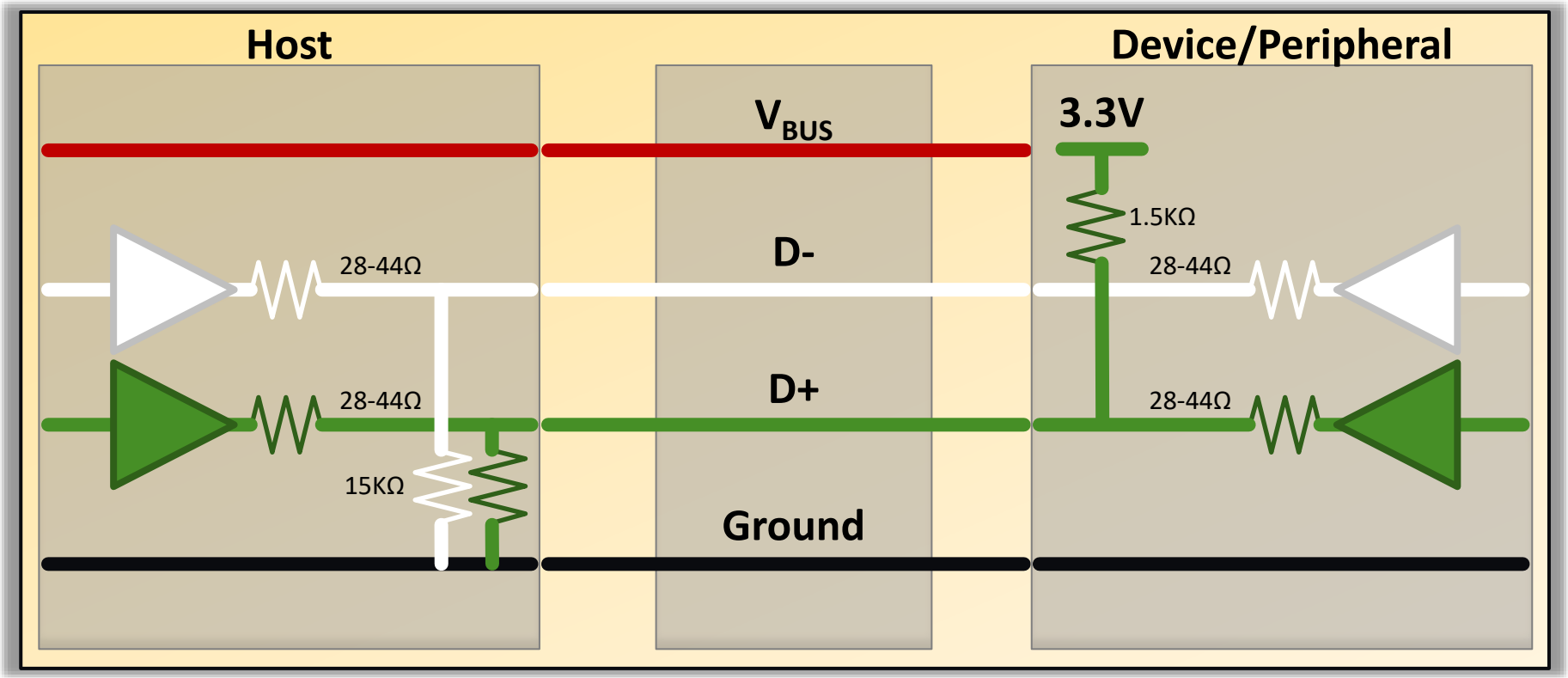
Attached : Low-Speed

- 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.



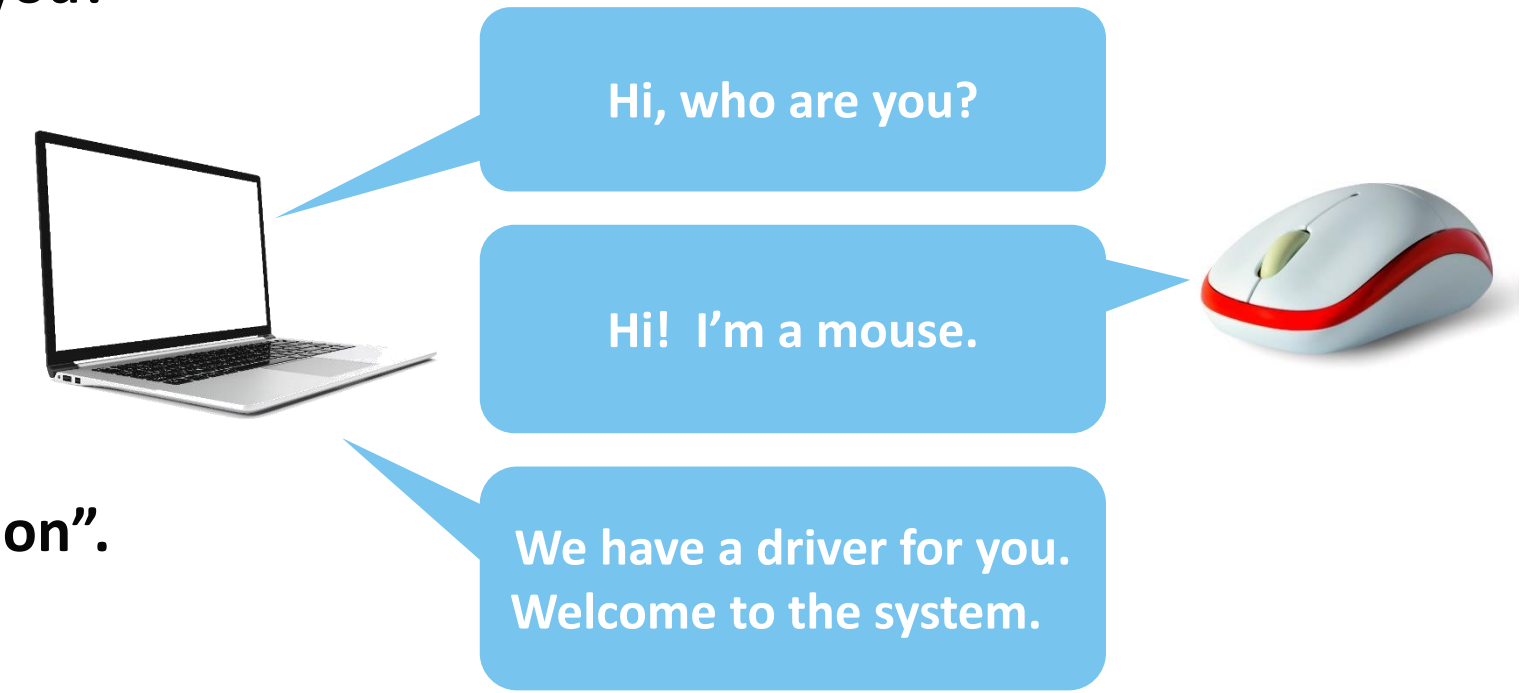
Attached : Full/High-Speed

- 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.

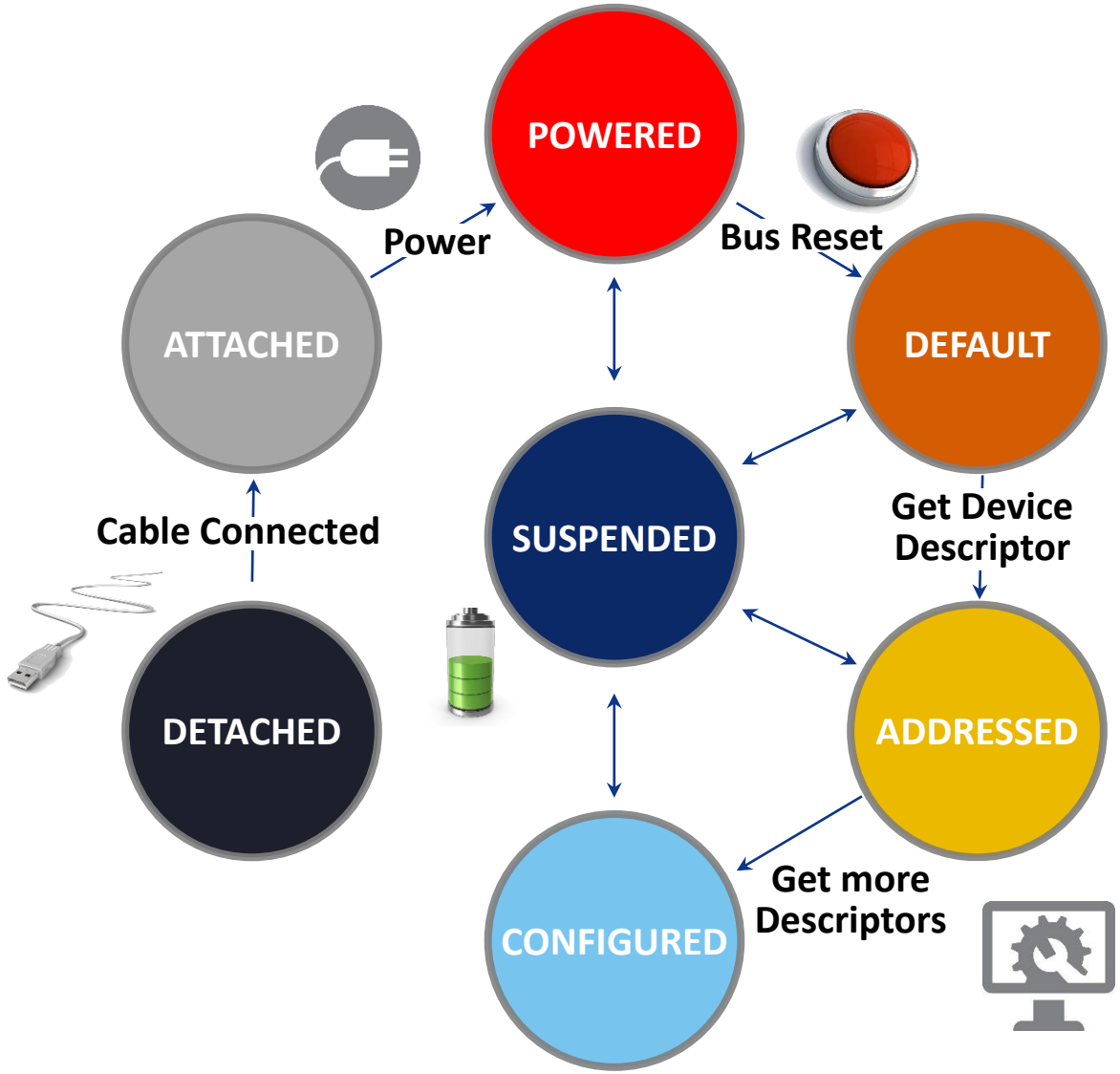


Enumeration

- 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”.



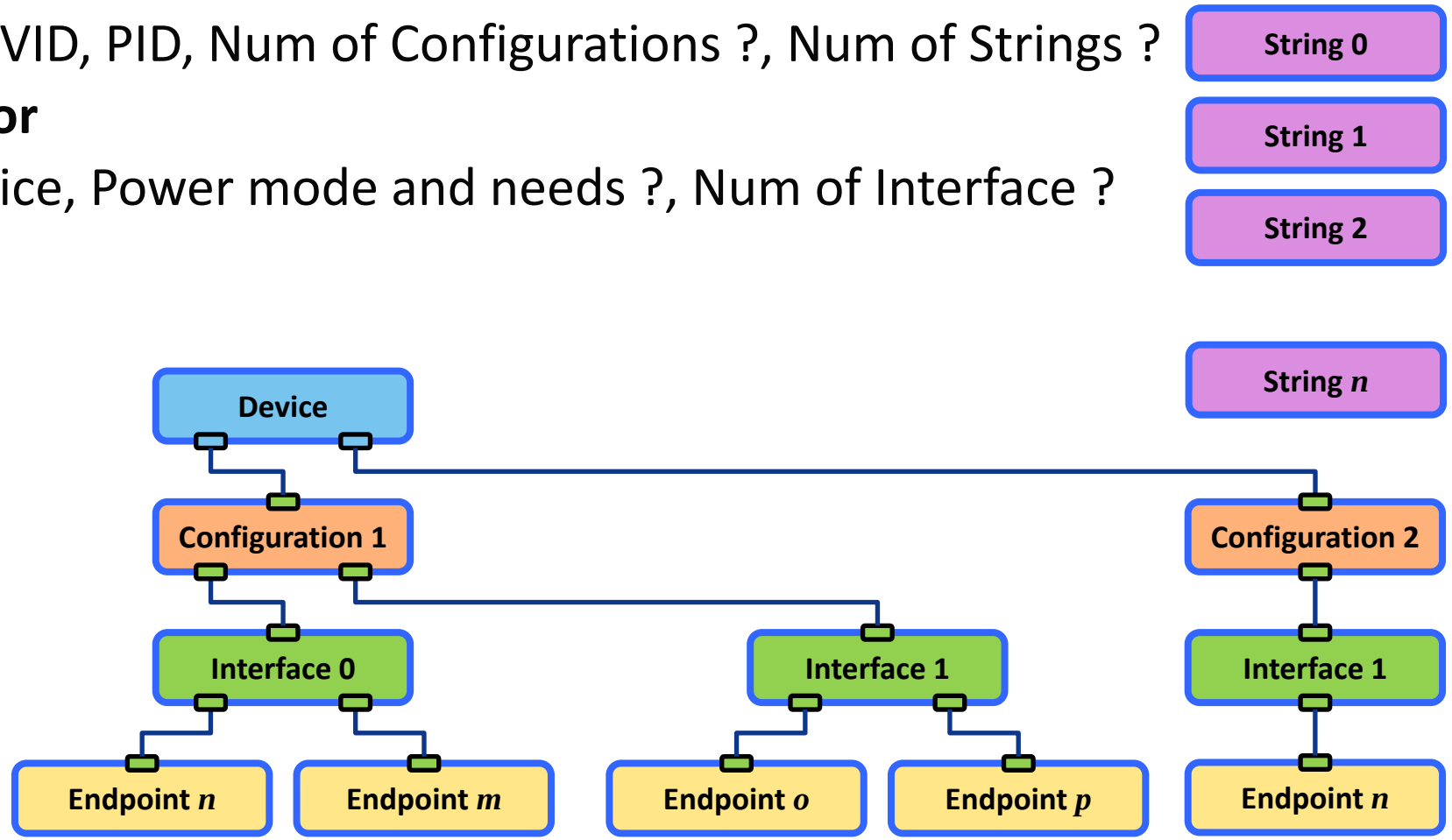
Enumeration Process



MICROCHIP

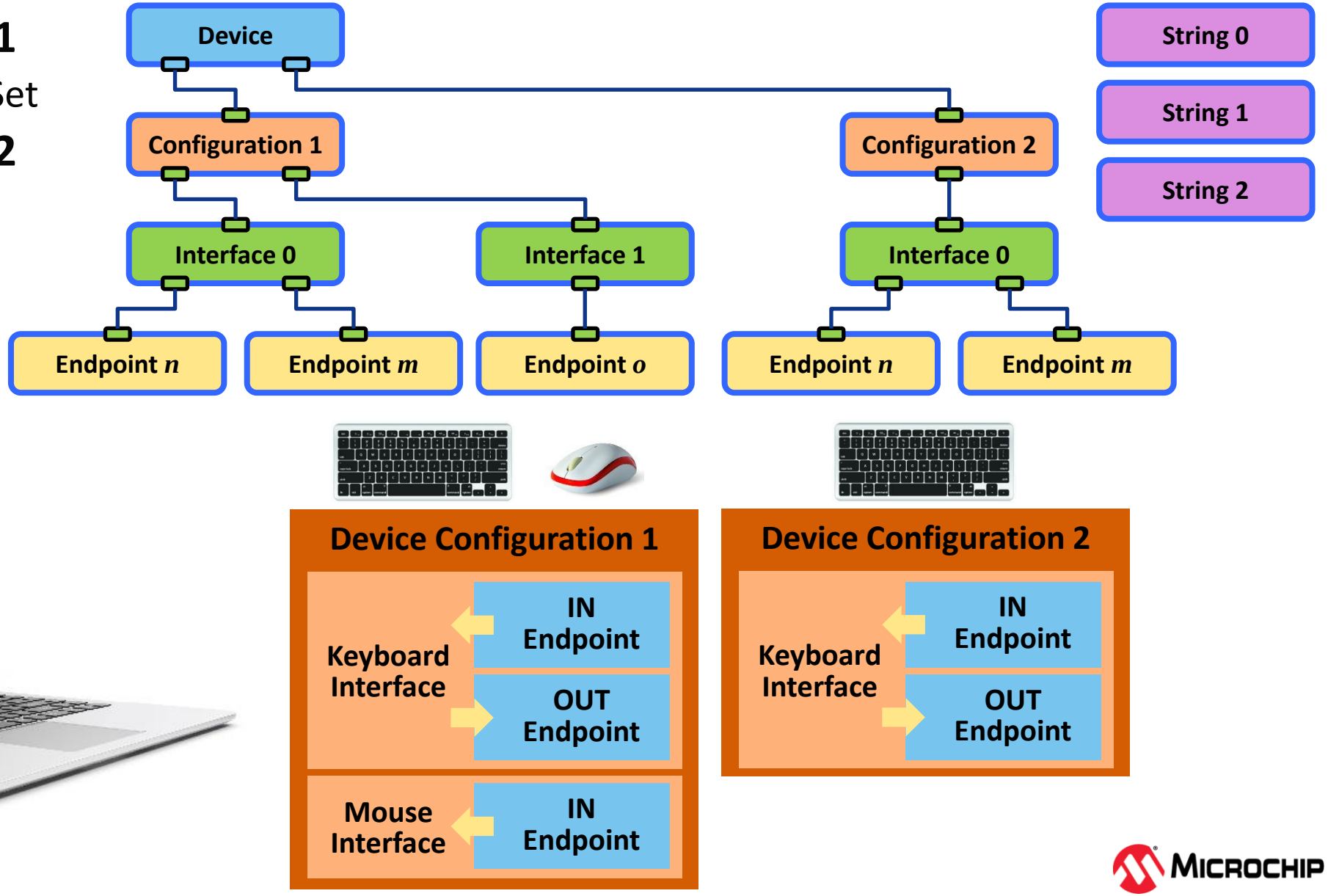
Descriptors

- **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…**



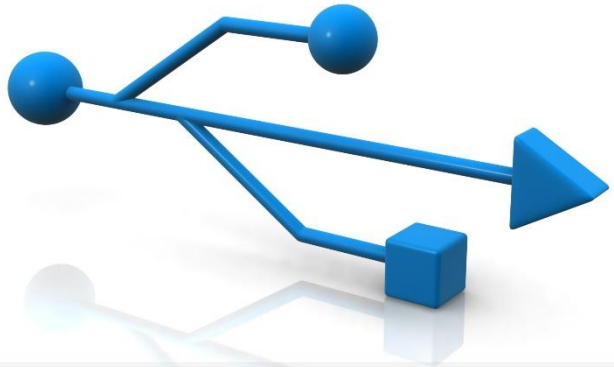
Configuration Descriptor Example

- **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.

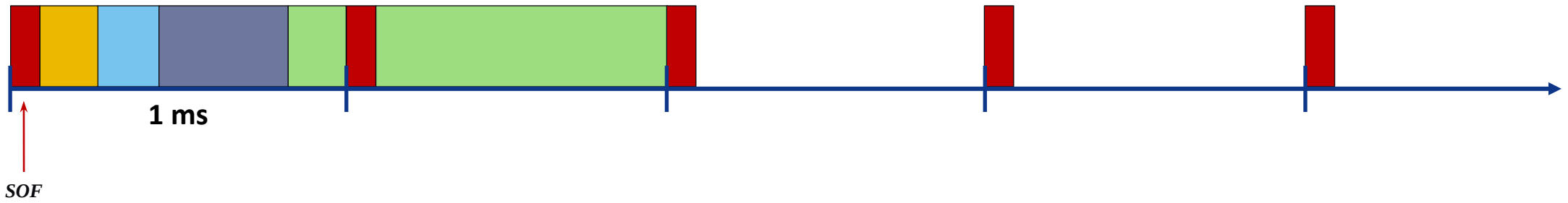


Transfer/ Endpoint Type	Polling Interval	% Reserved BW/Frame for all transfers of this type	Max. # Data Bytes/Frame/Endpoint (Max# transactions per frame @ Max Ep Size)*	Data Integrity
Interrupt	Fixed, Periodic	10	64 (1 x 64)	Yes
Isochronous	Fixed, Periodic	90	1,023 (1 x 1023)	No
Bulk	Variable, Uses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

MICROCHIP

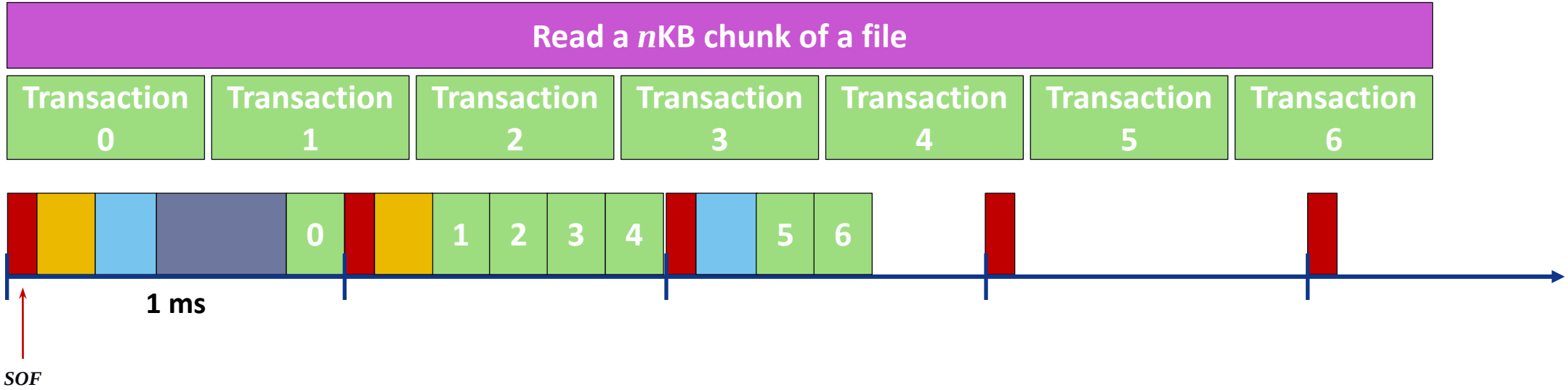
Data Transfer Scheduling

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

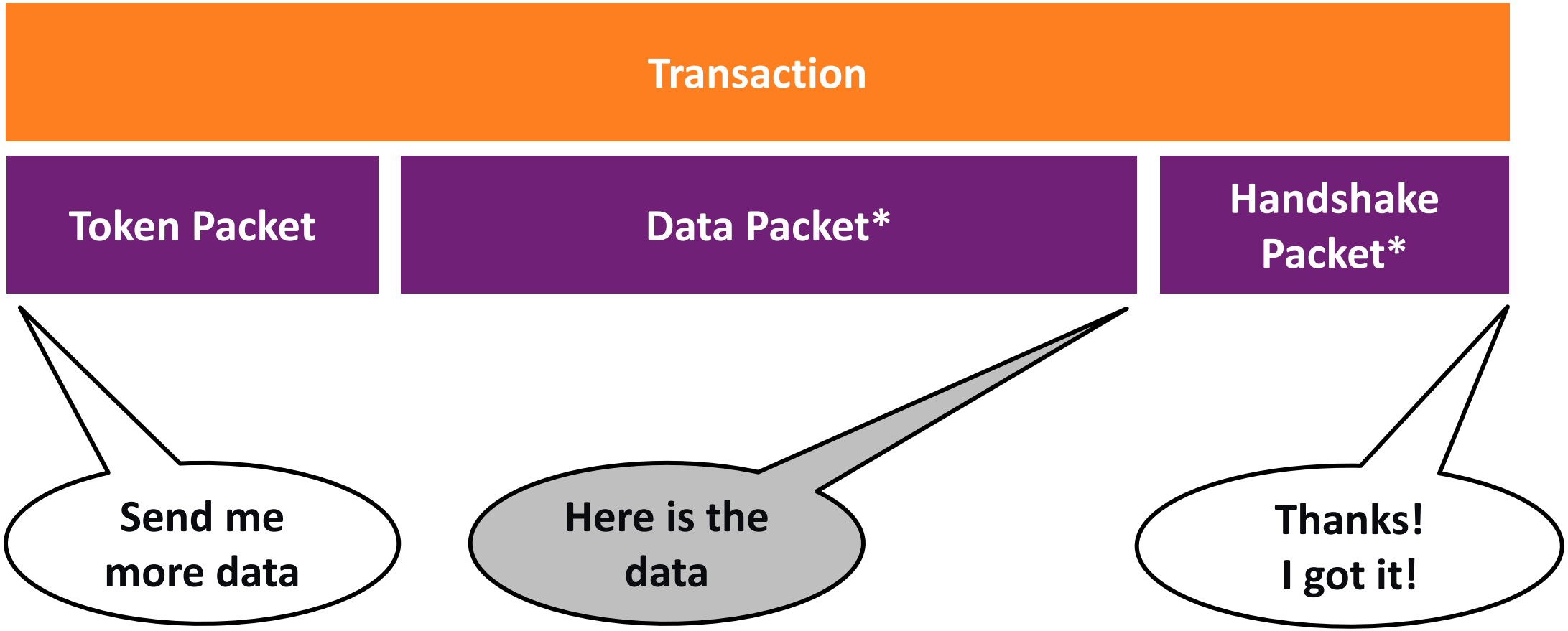


USB Transactions

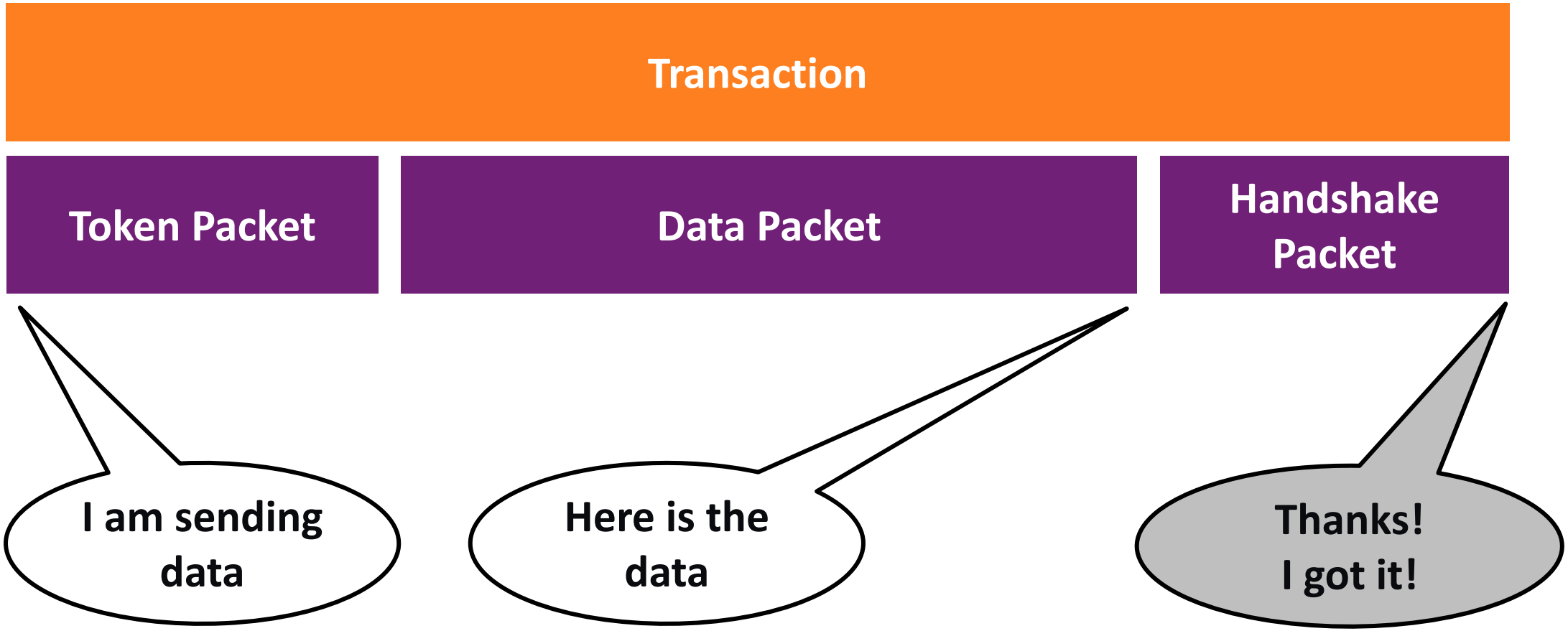
- **Specification defined maximum packet size for endpoint.**



USB Transactions - Require Data



USB Transactions - Send Data



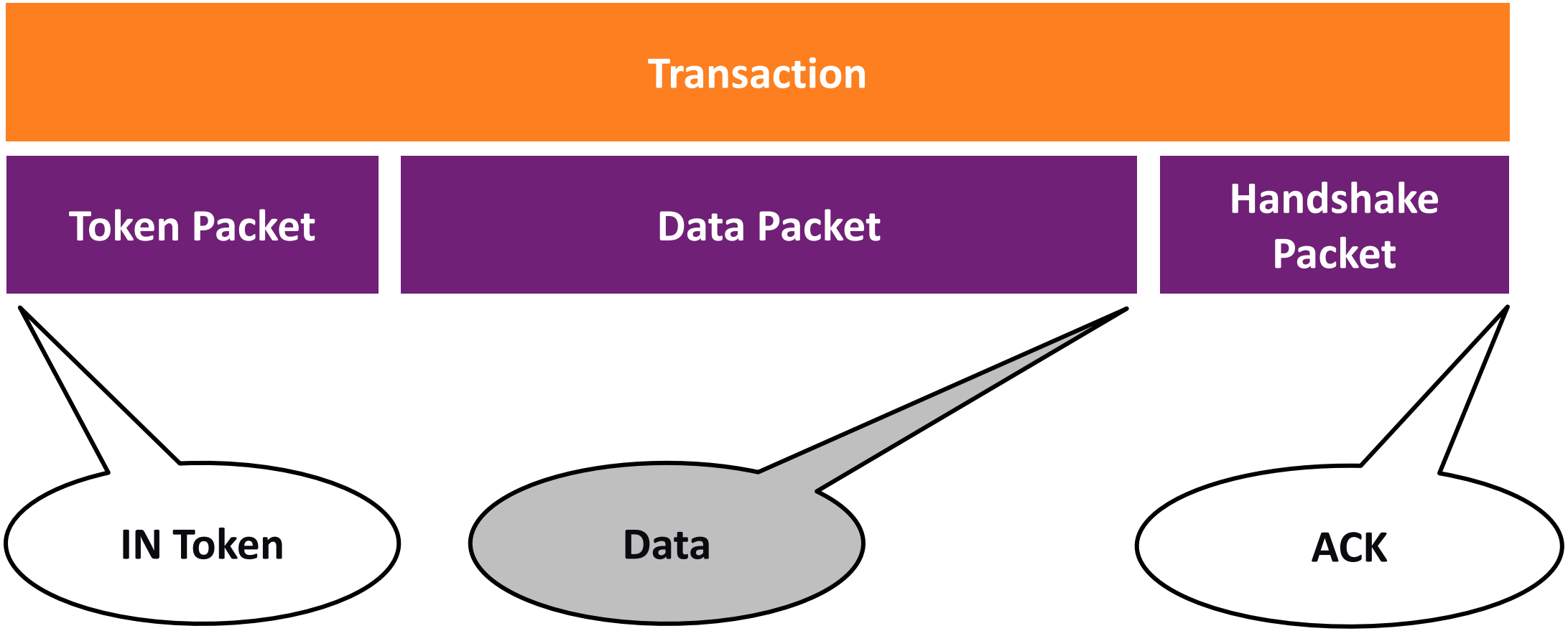
USB Transactions, Token & Handshake

Token Packet	Data Packet	Handshake Packet
IN		ACK
OUT		NACK
SETUP		STALL
PING		NYET
		ERR

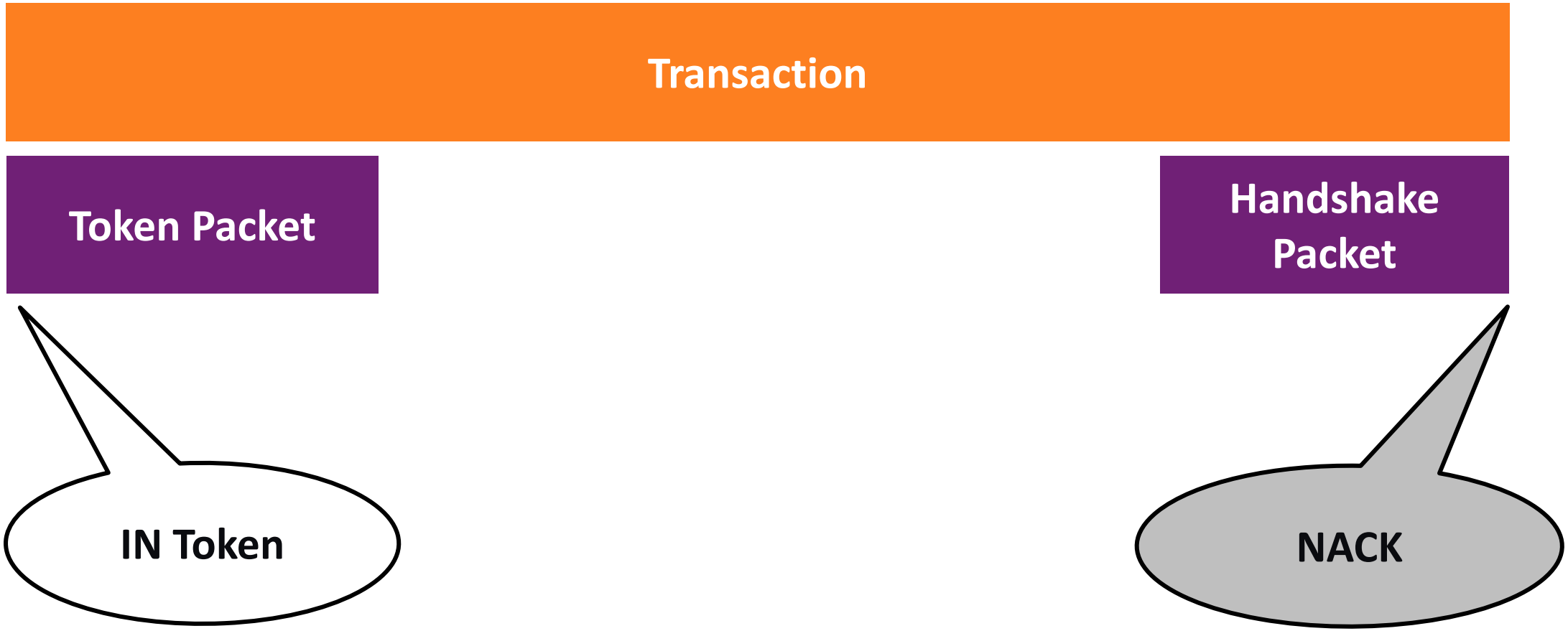
IN : Require Data
OUT : Send Data
SETUP : Control Operation
PING : Confirm Device Ready (2.0)

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

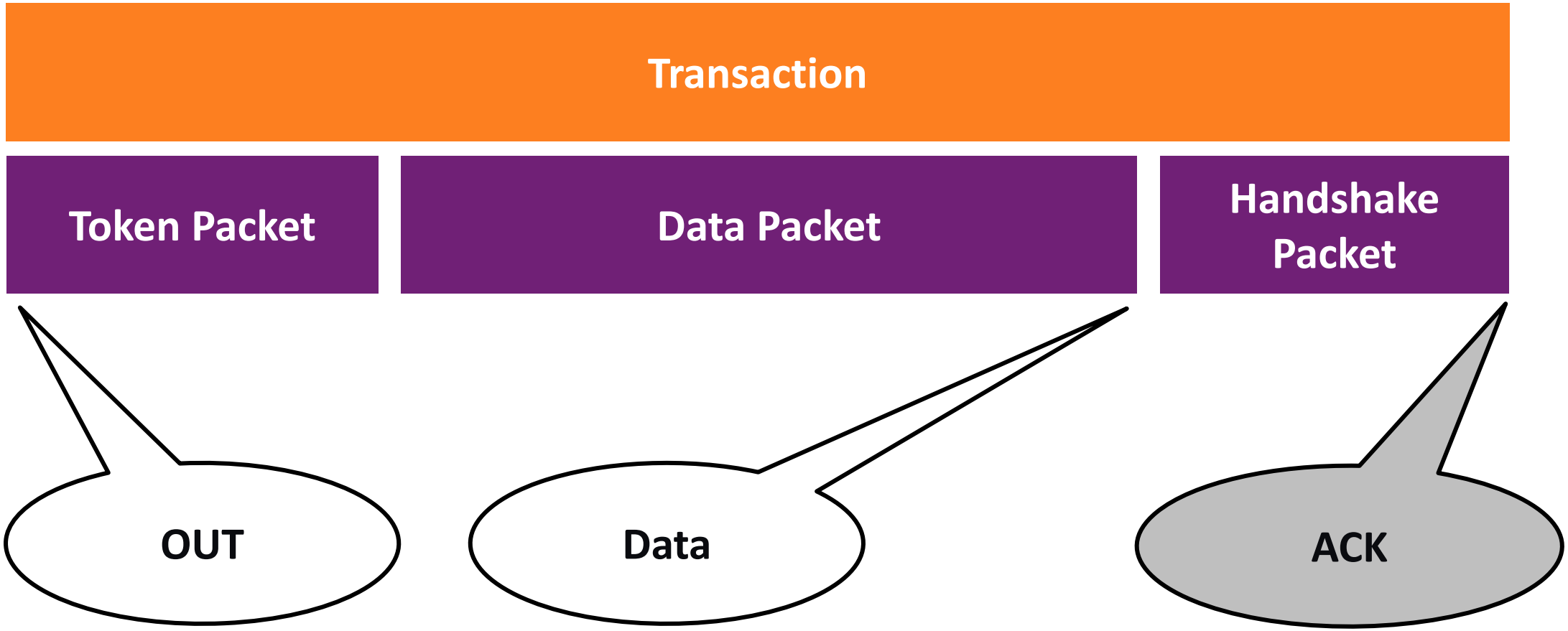
USB Transactions - Require Data, IN - ACK



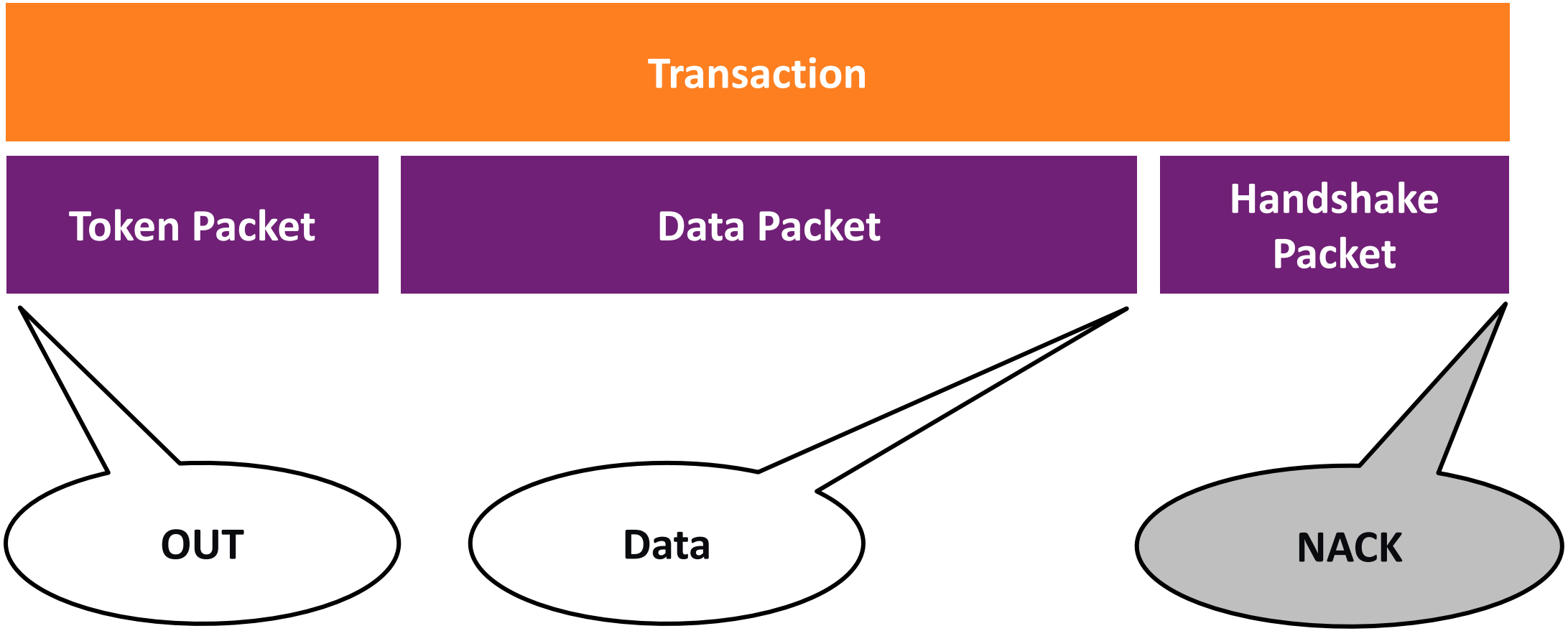
USB Transactions - Require Data, IN - NACK



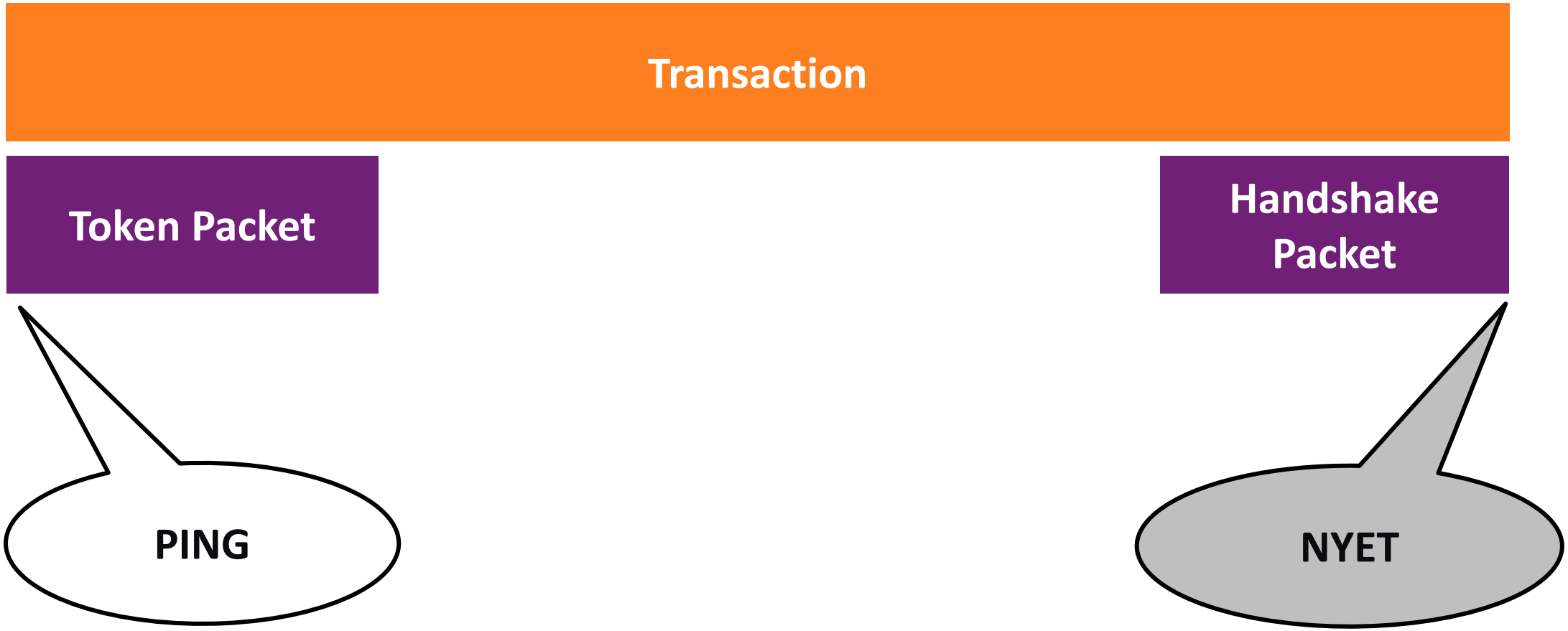
USB Transactions - Send Data, OUT - ACK



USB Transactions - Send Data, OUT - NACK

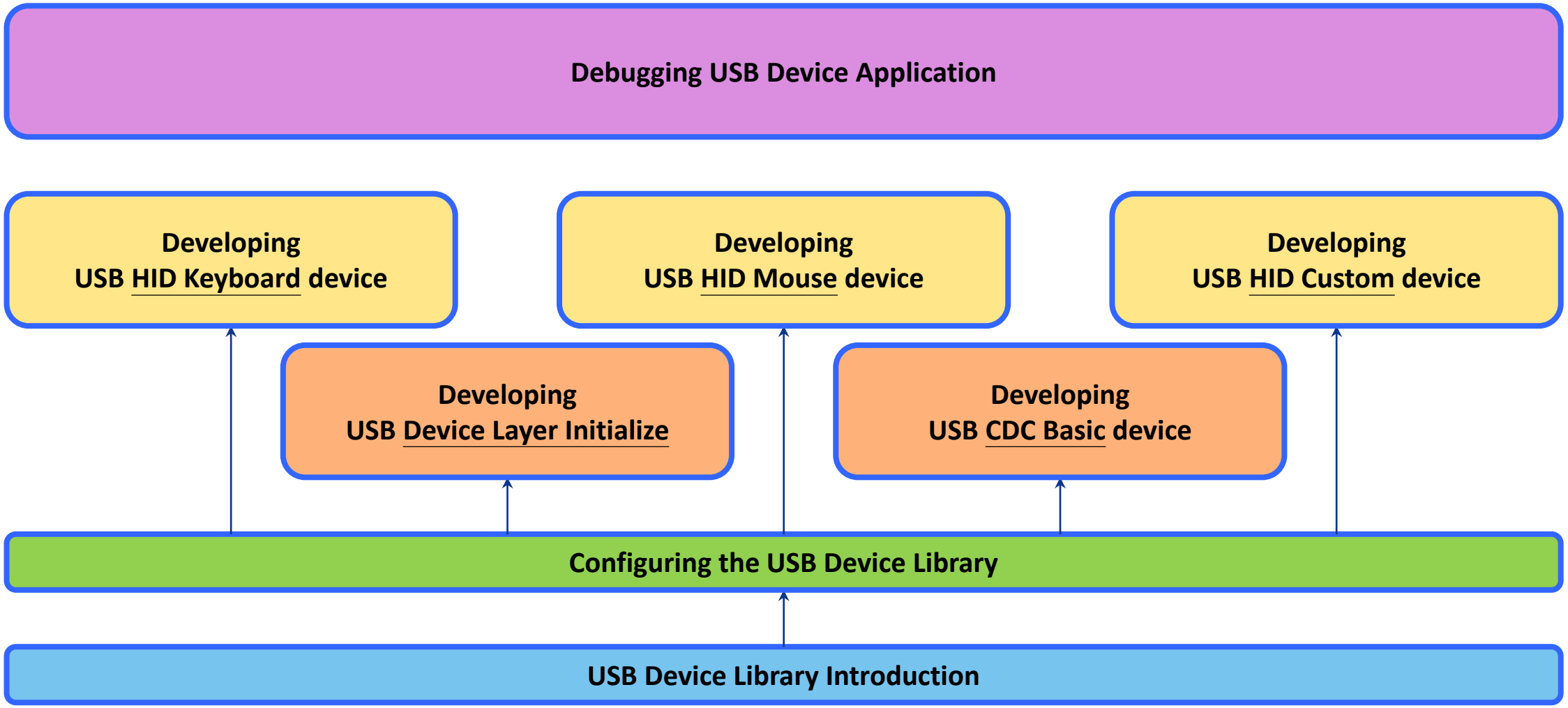


USB Transactions - Send Data, PING - NYET

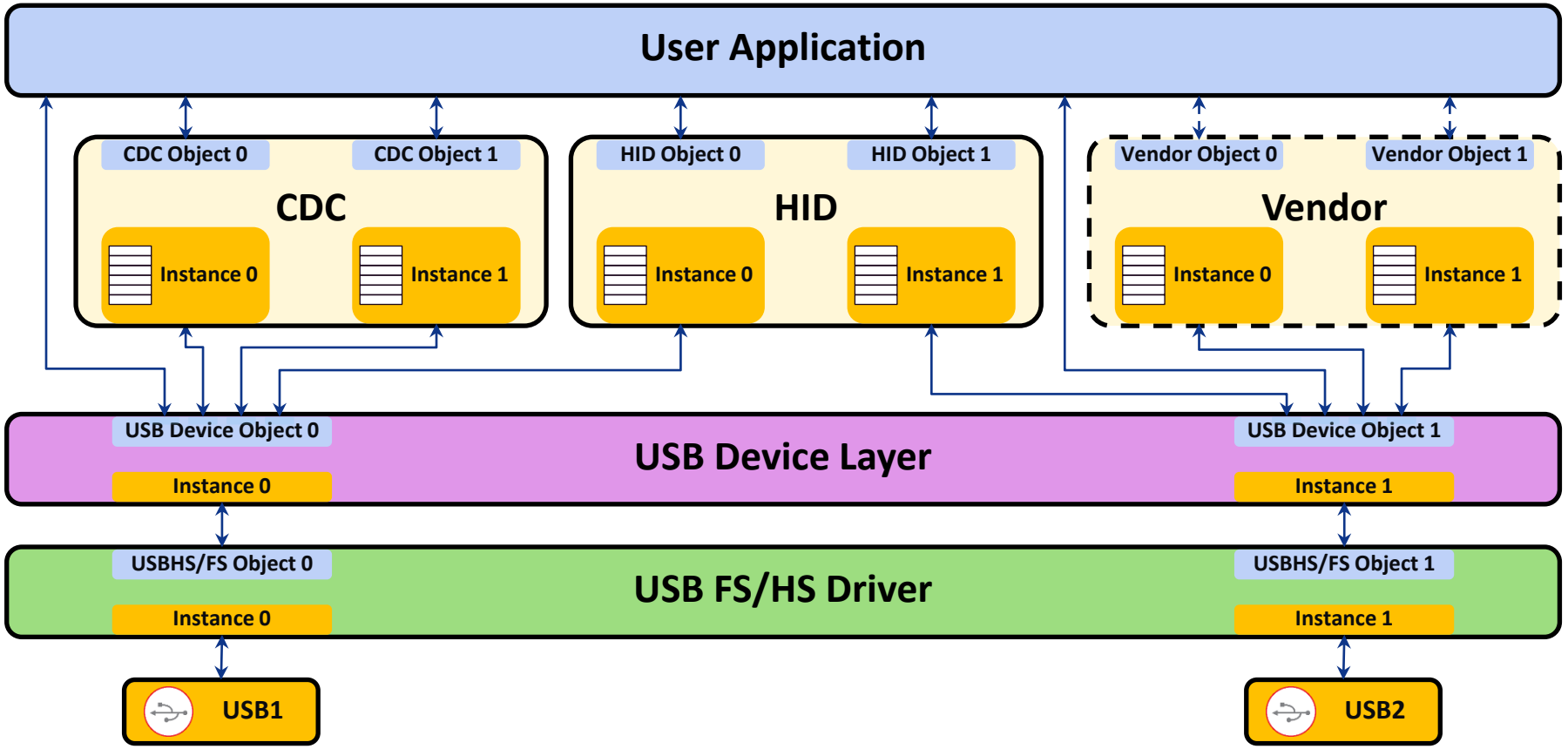


USB Device Library
(USB Device Architecture)

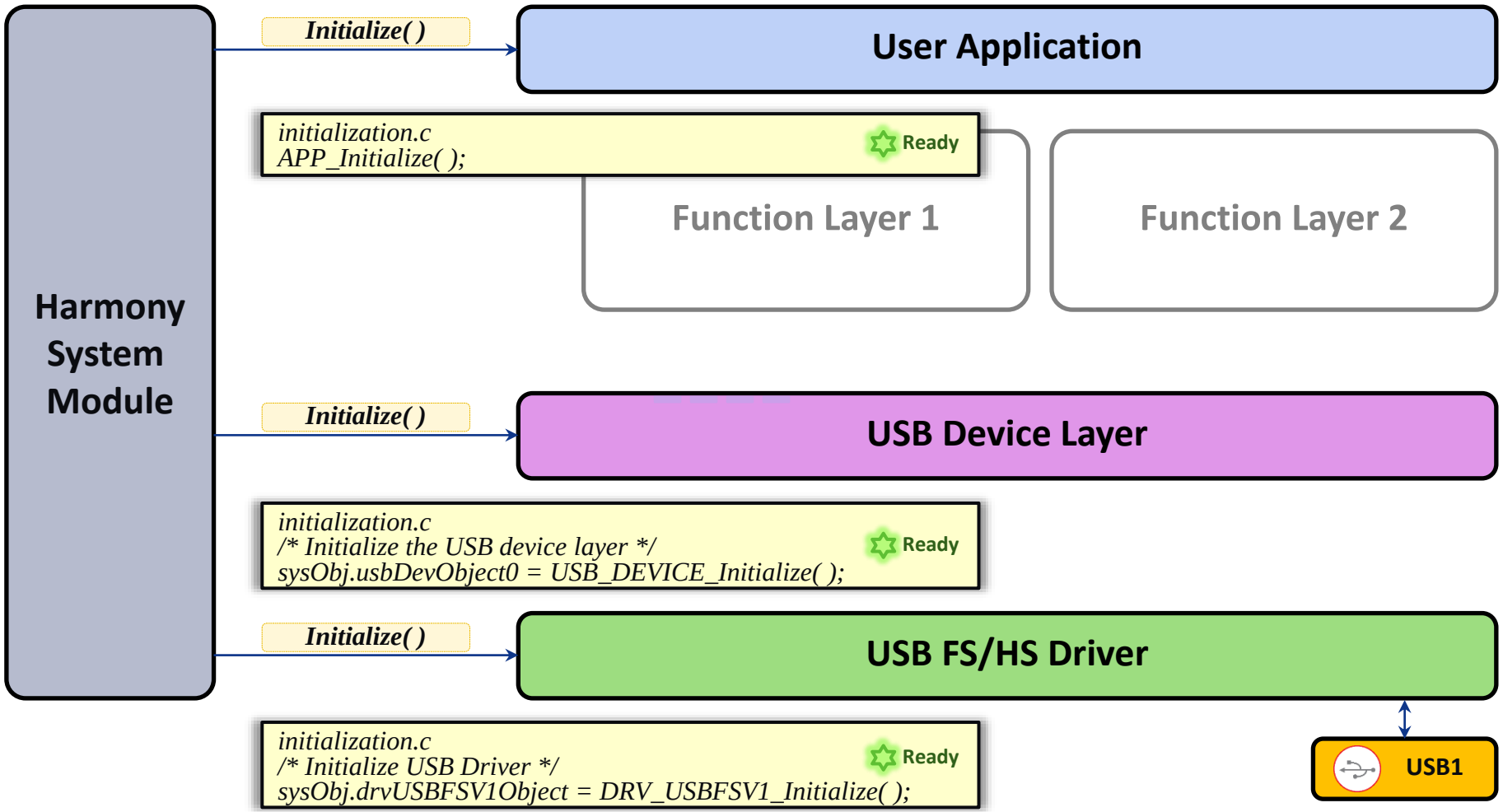
Course Structure for USB Device Library



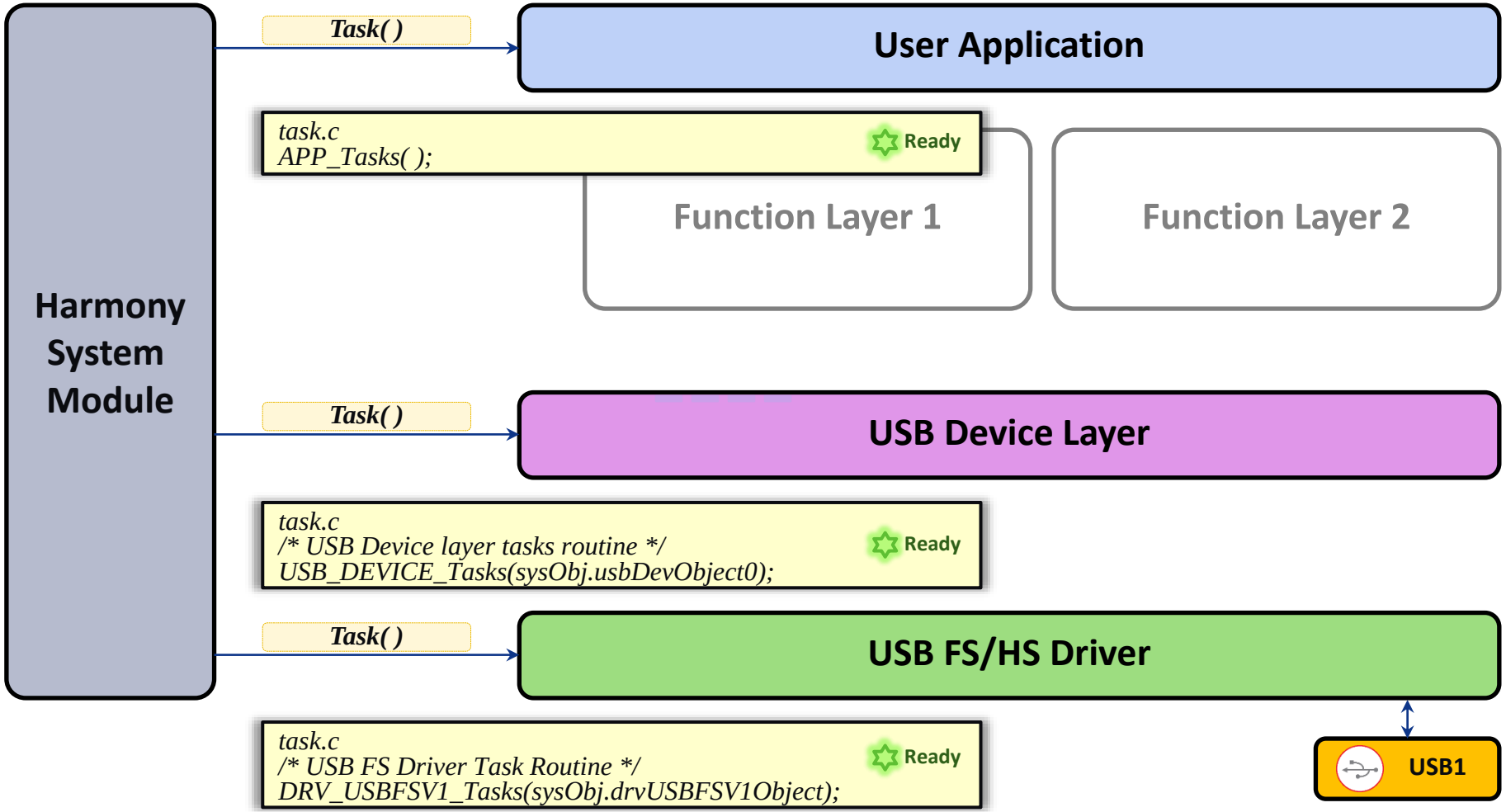
USB Device Library Architecture



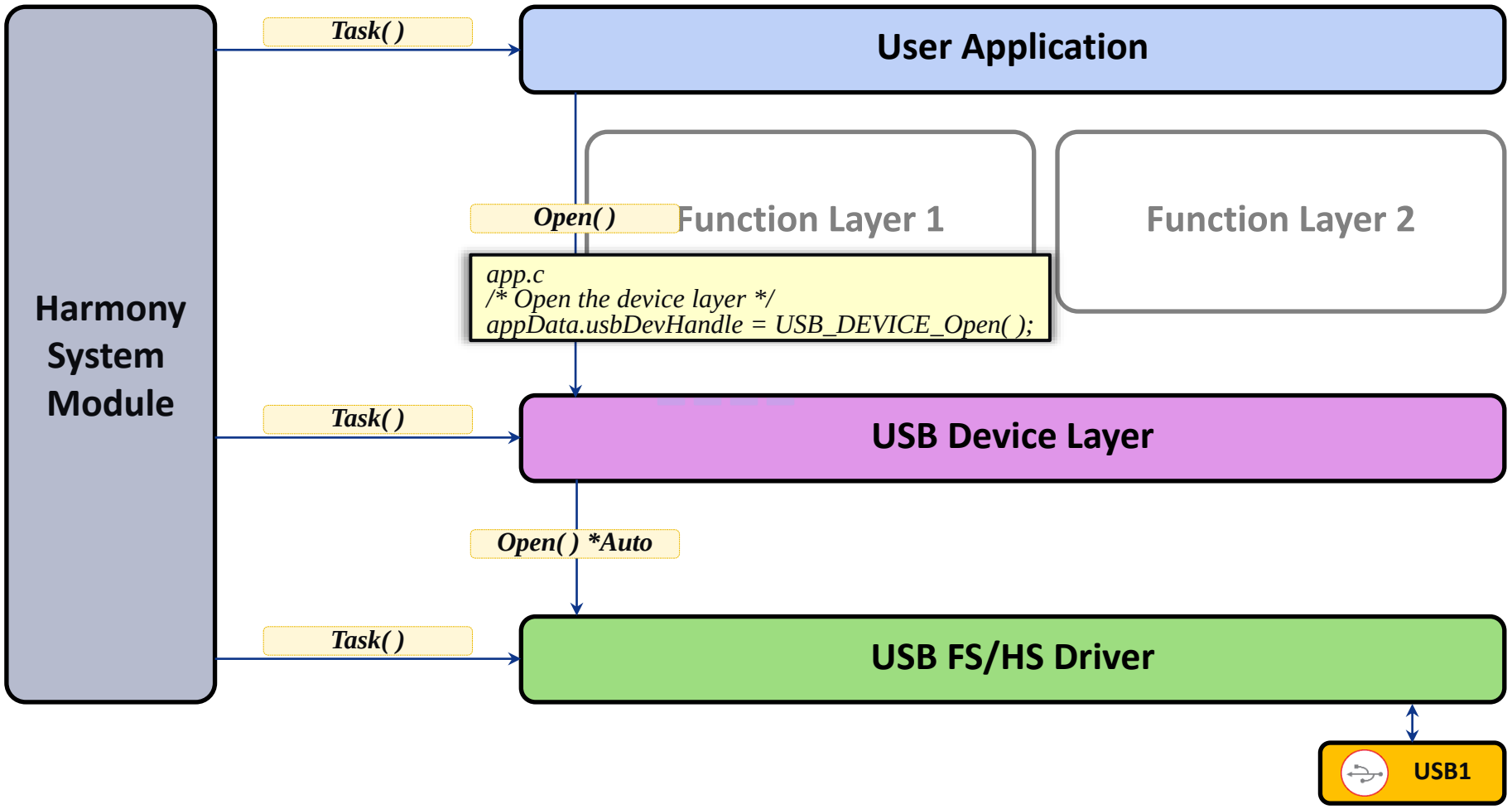
USB Stack Abstraction Model - Initialize & Open



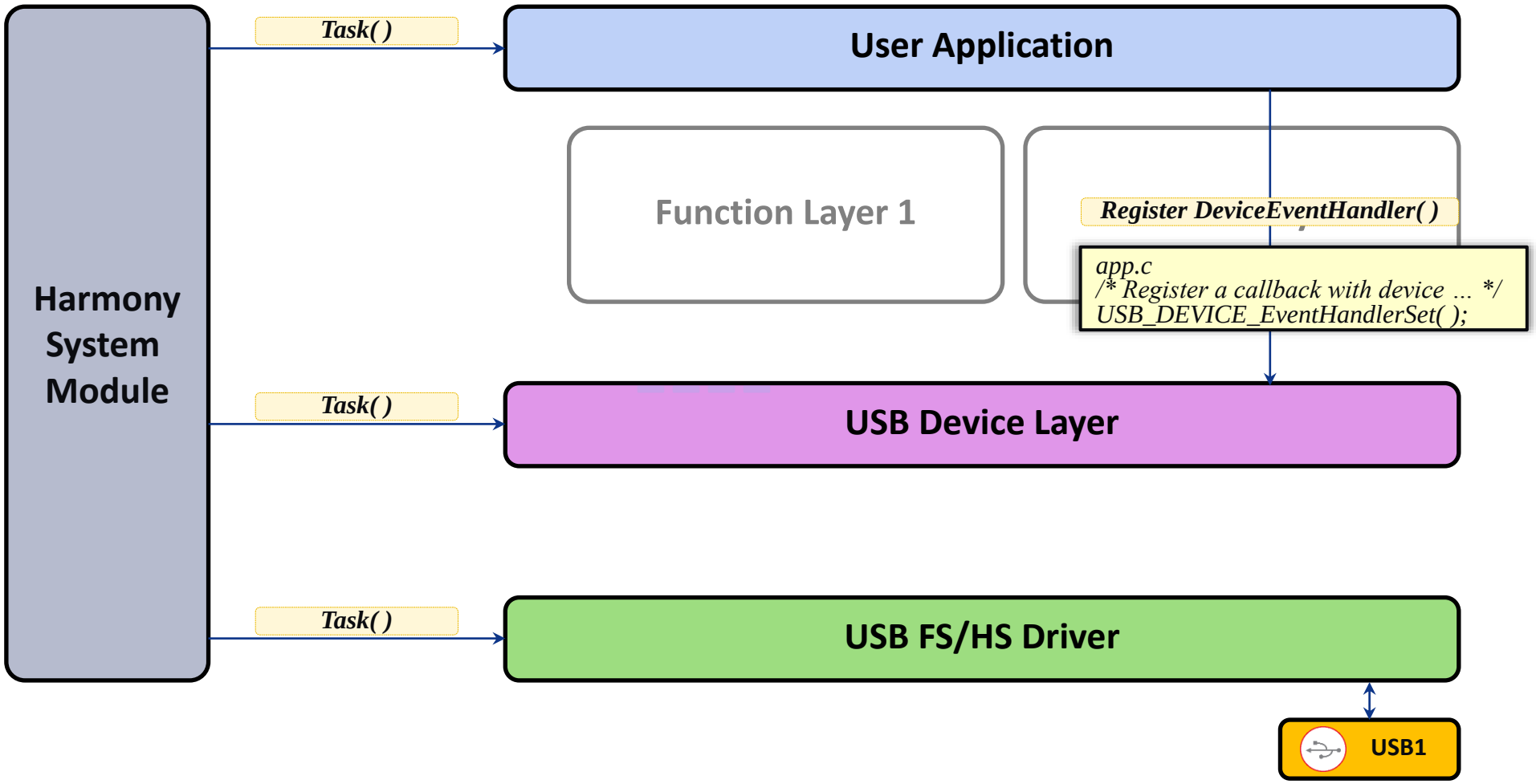
USB Stack Abstraction Model - Initialize & Open



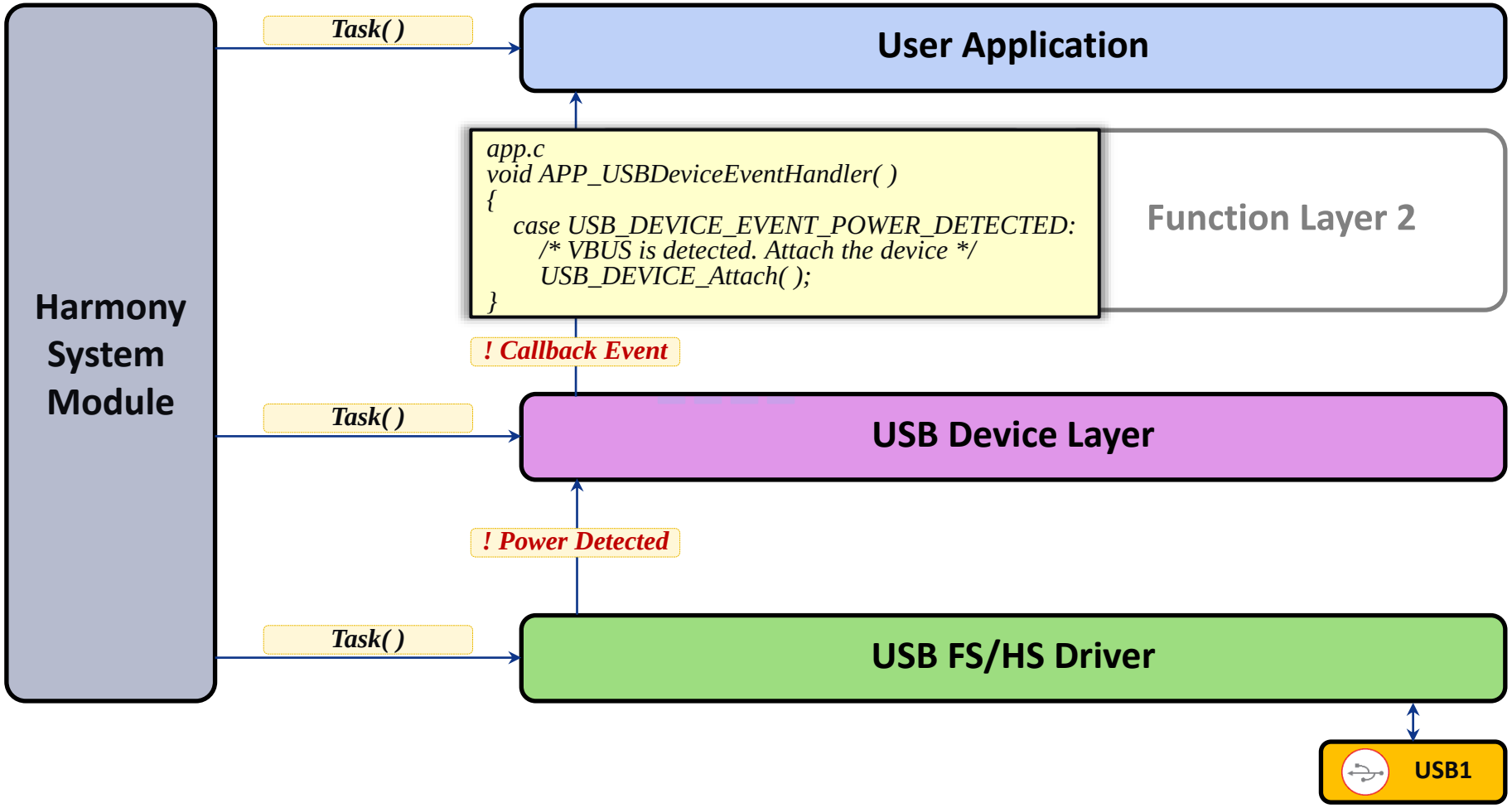
USB Stack Abstraction Model - Initialize & Open



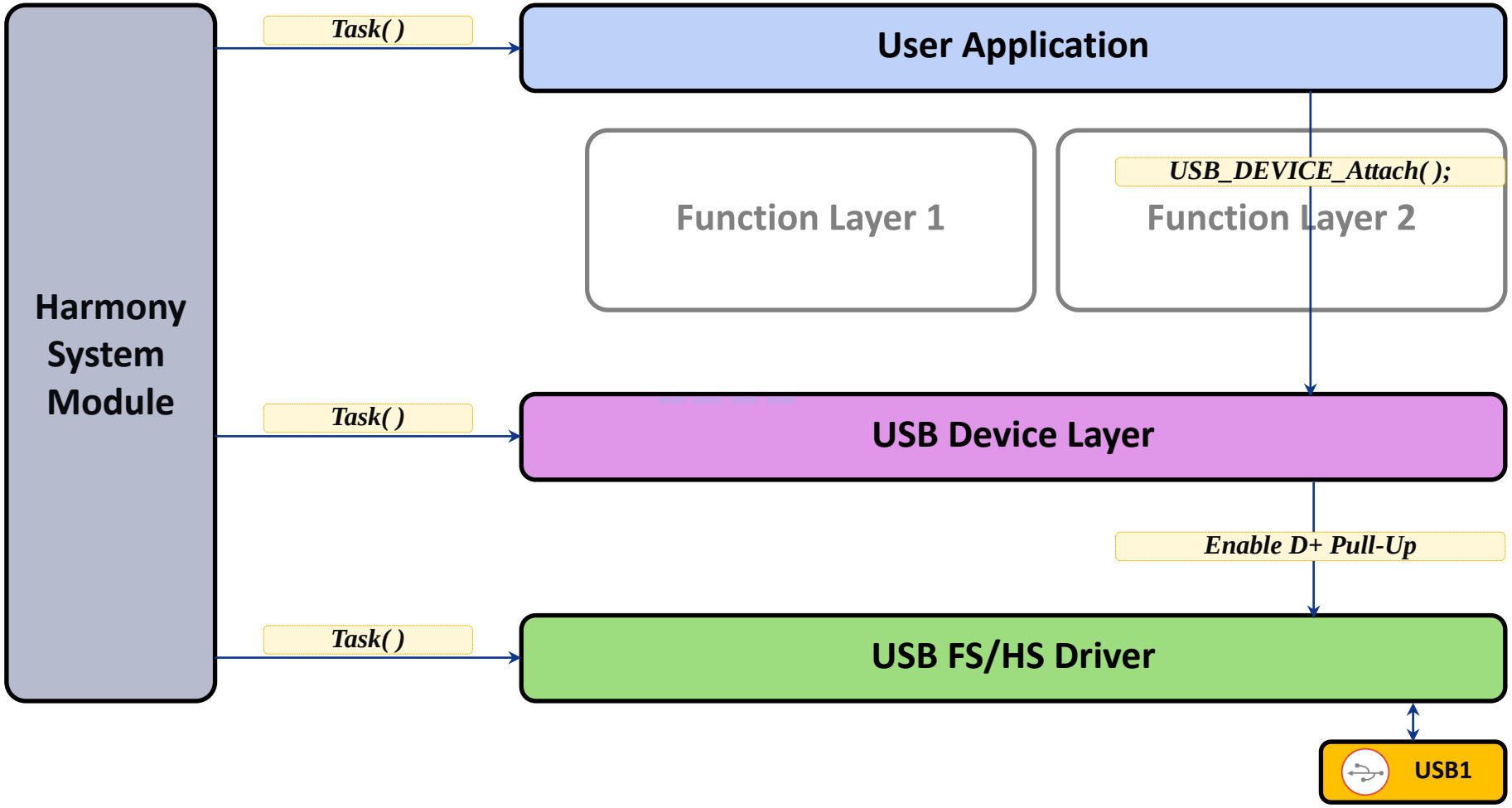
USB Stack Abstraction Model - Register Device Event Handler



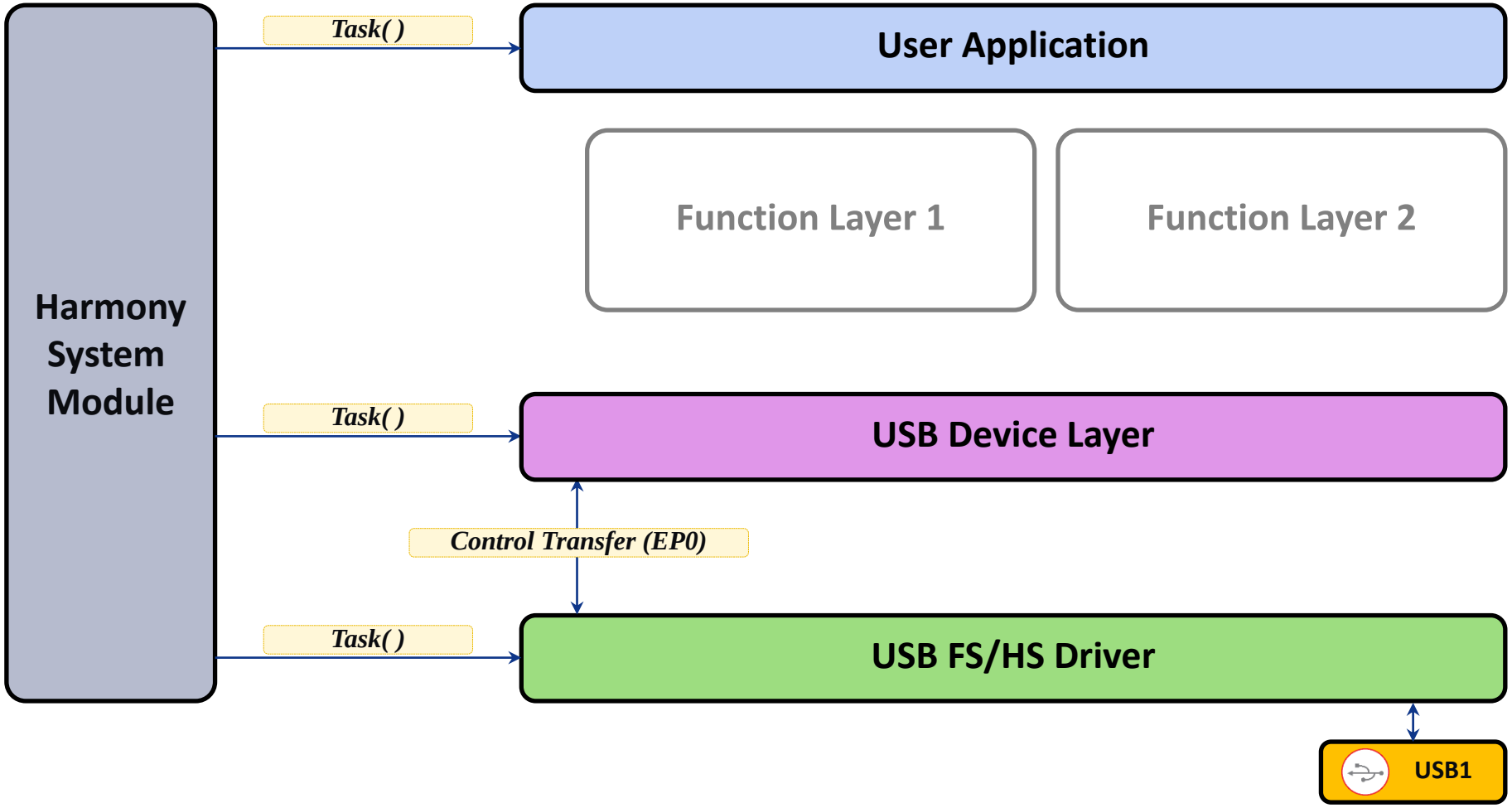
USB Stack Abstraction Model - Powered



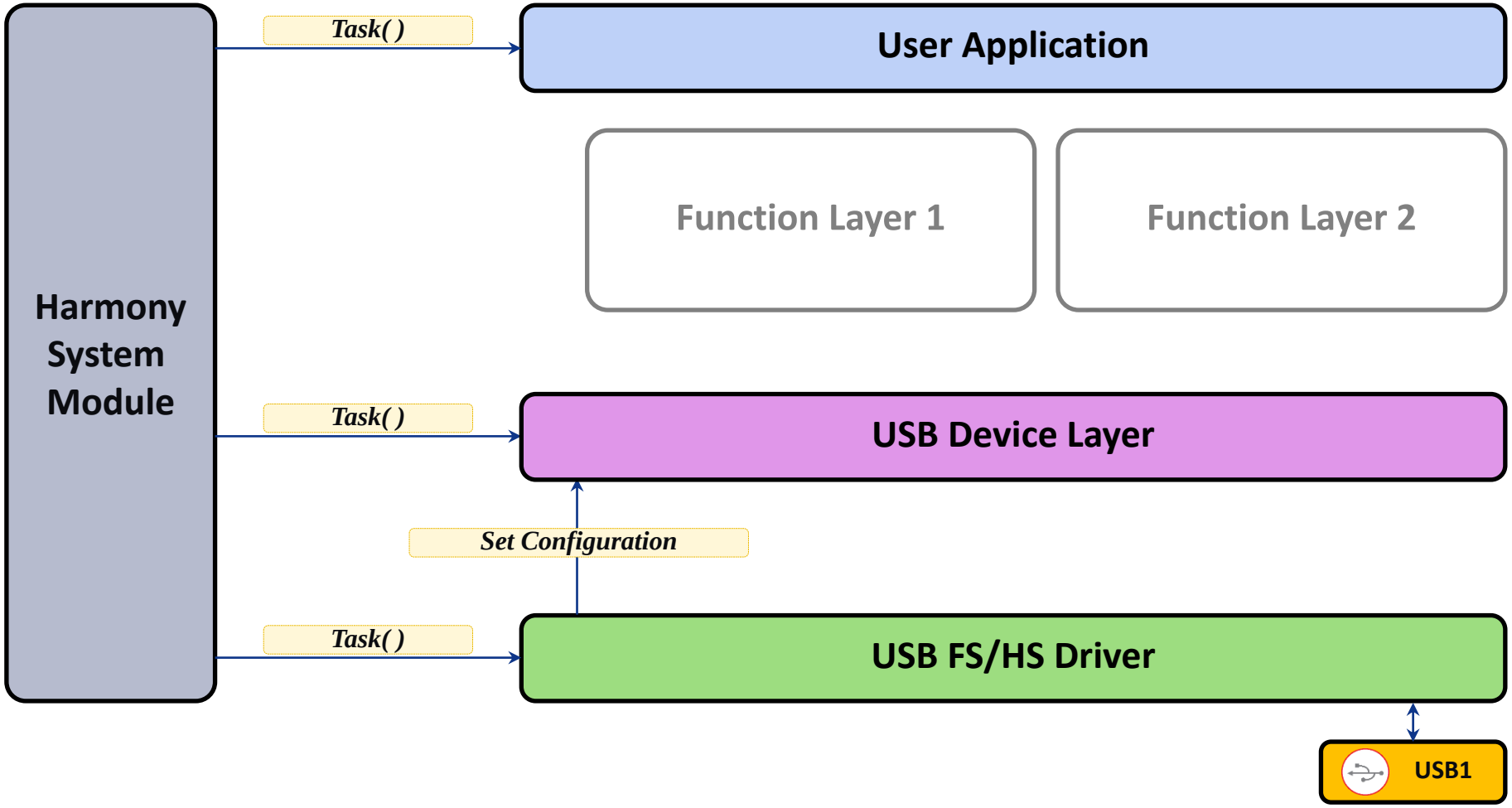
USB Stack Abstraction Model - Attach



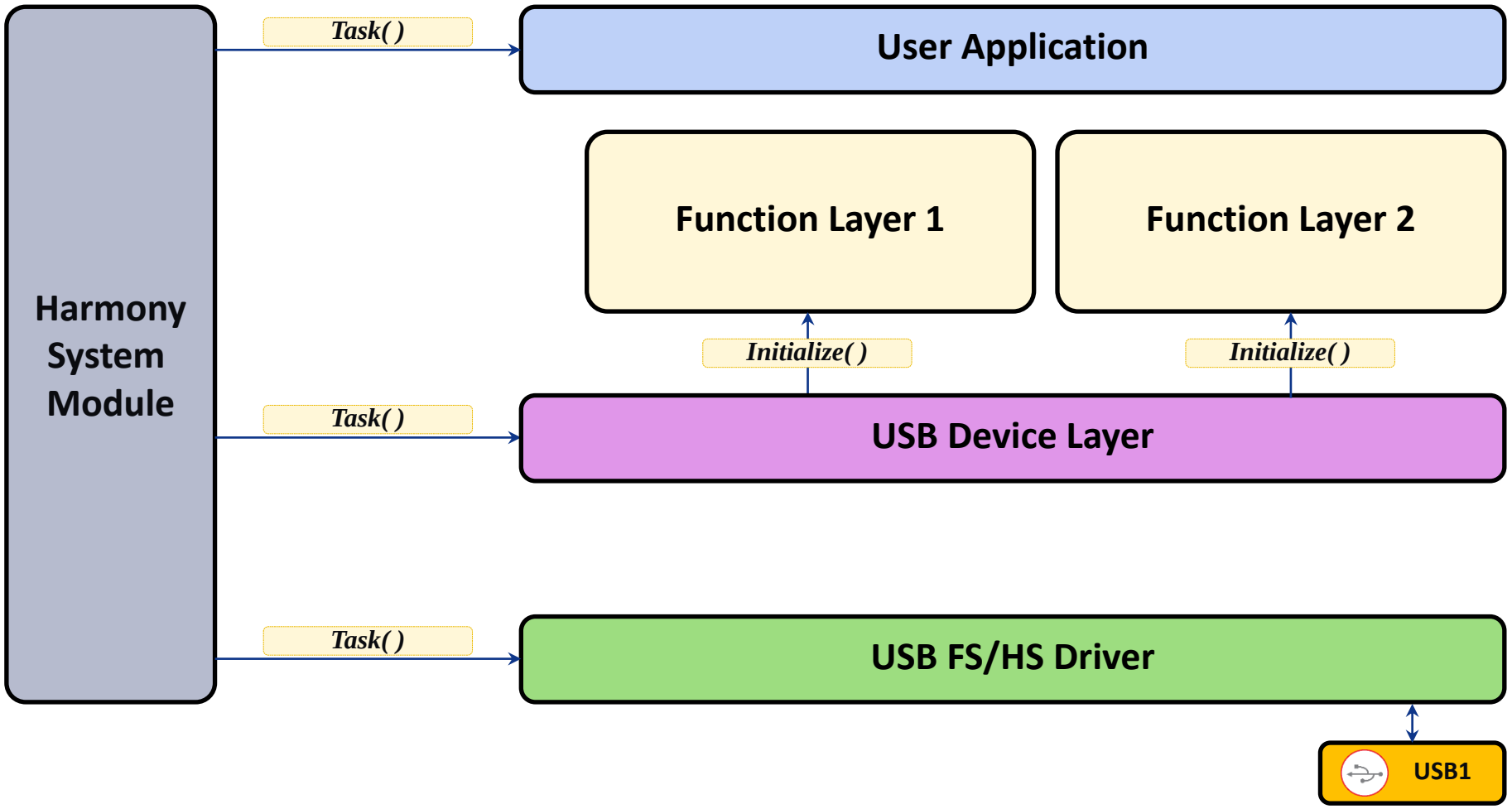
USB Stack Abstraction Model – Control Transfer



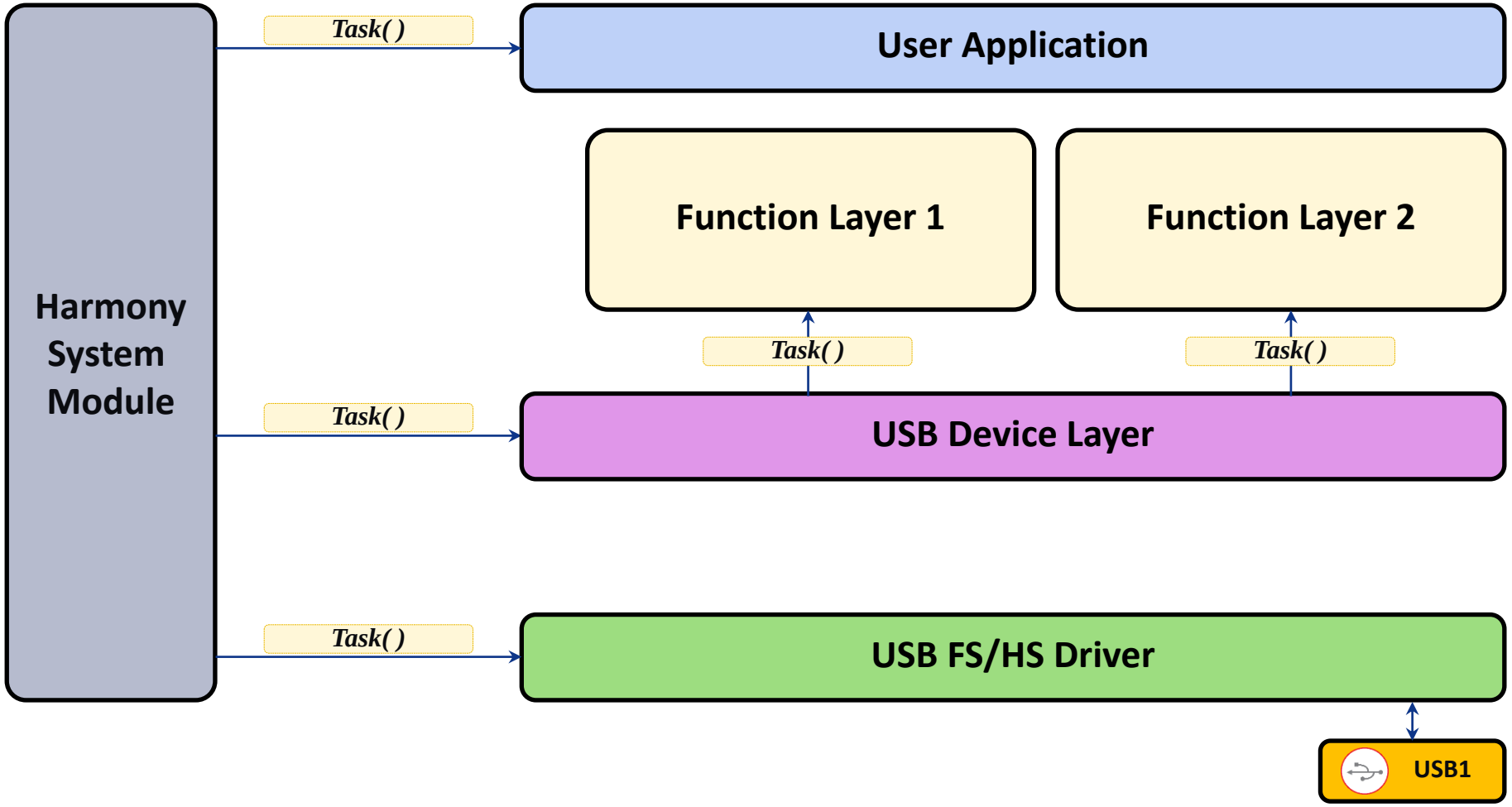
USB Stack Abstraction Model - Set Configuration



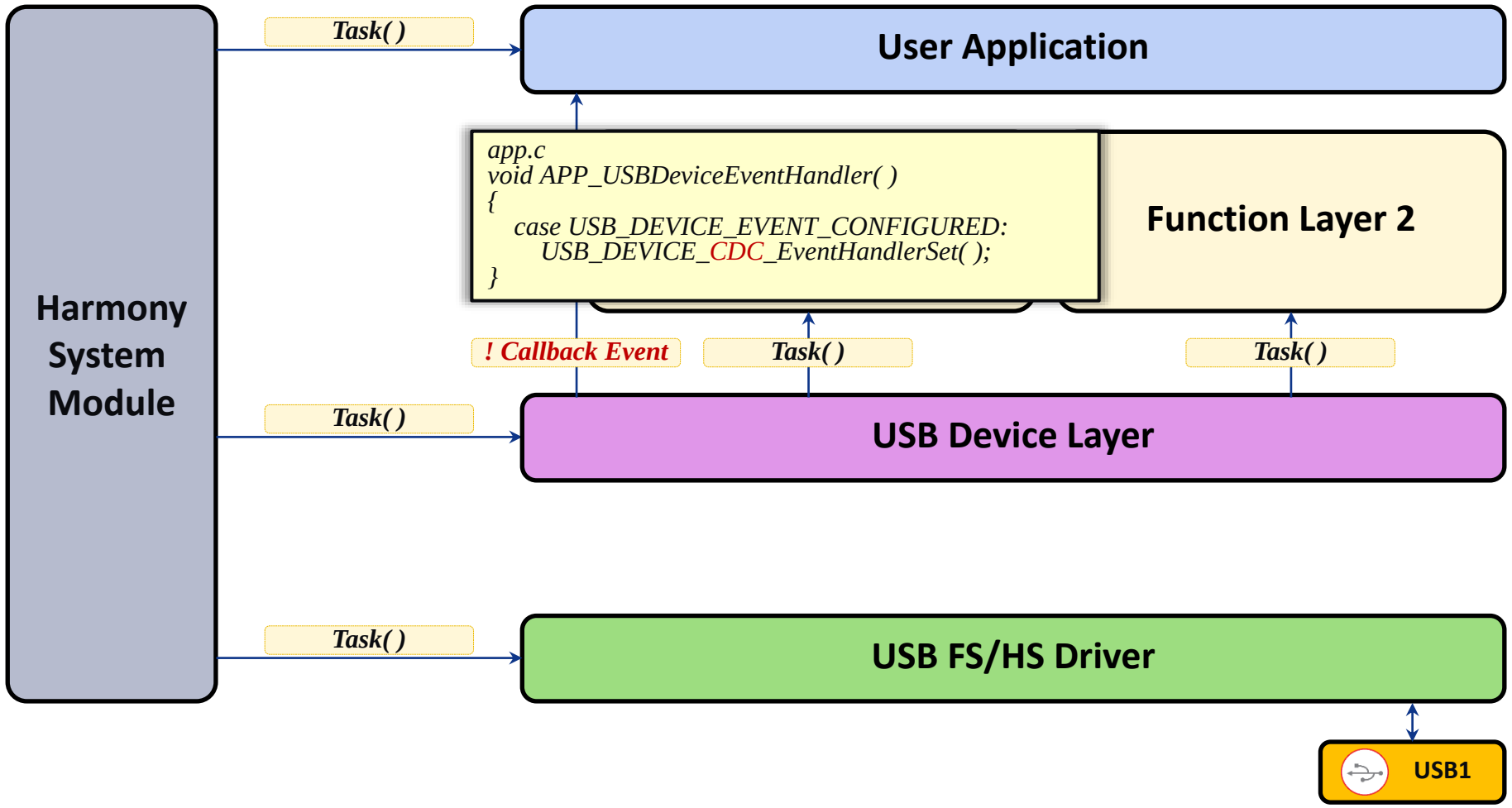
USB Stack Abstraction Model - Function Initialize & Open



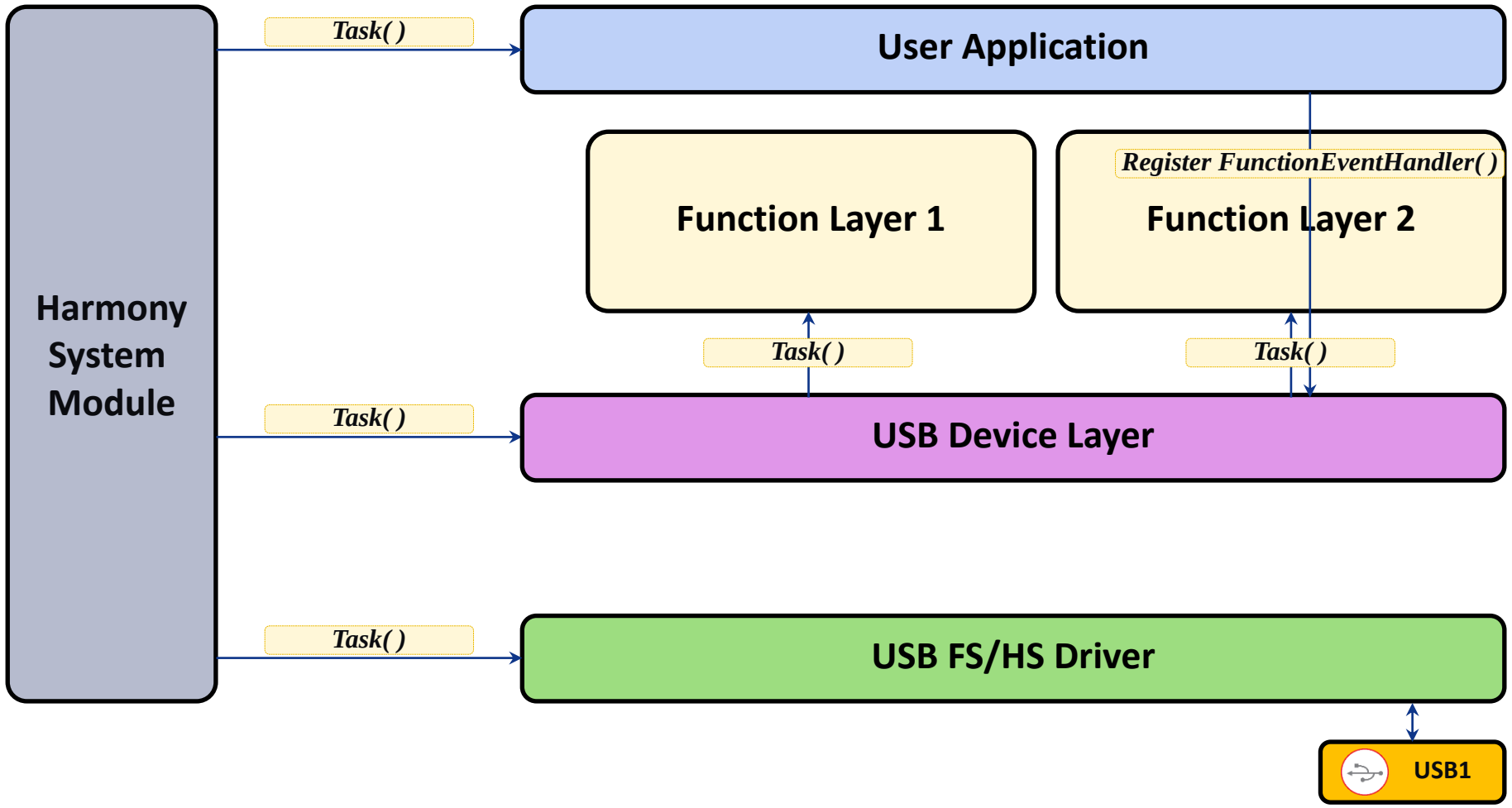
USB Stack Abstraction Model - Function Initialize & Open



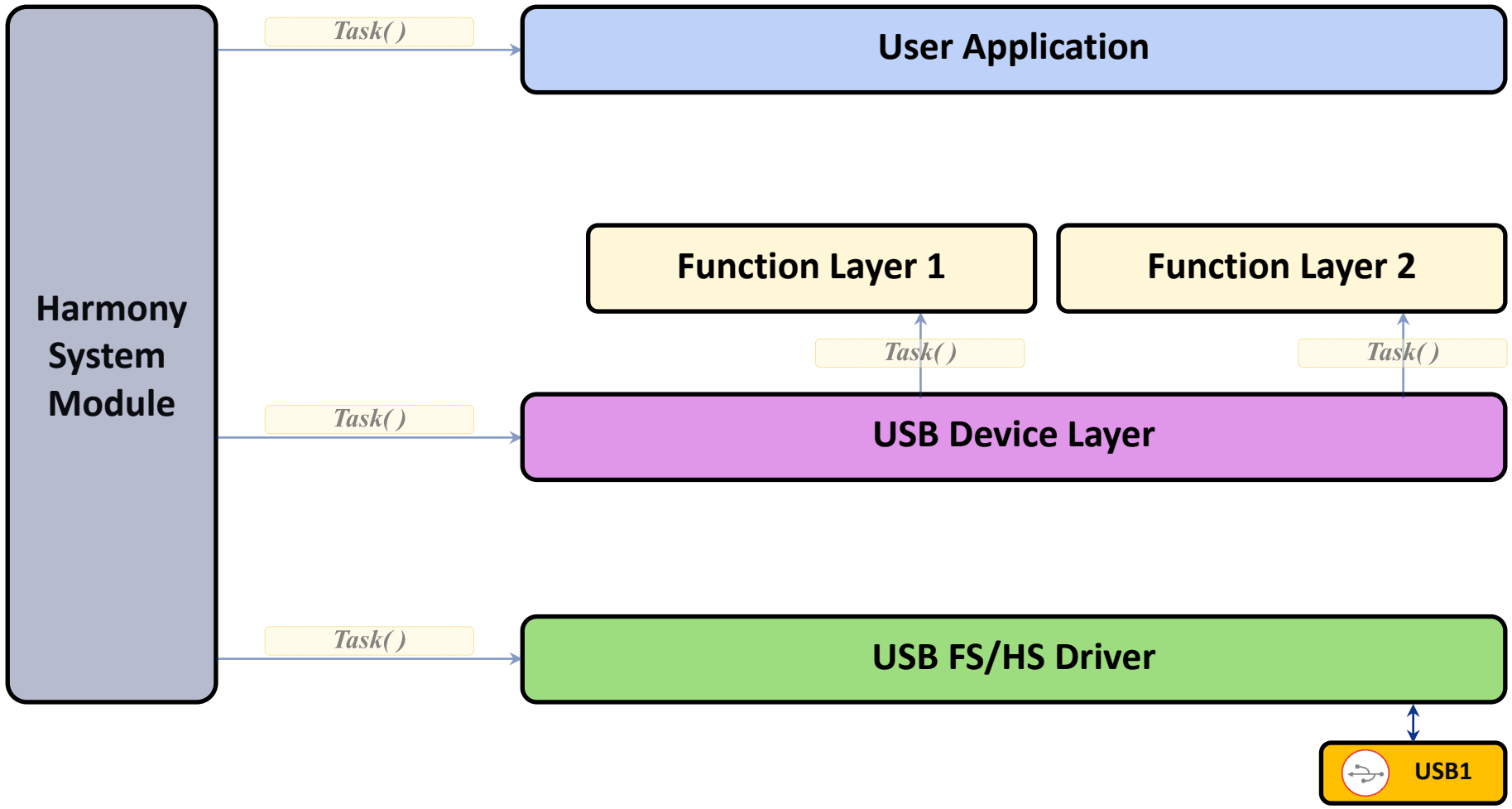
USB Stack Abstraction Model - Register Function Event Handler



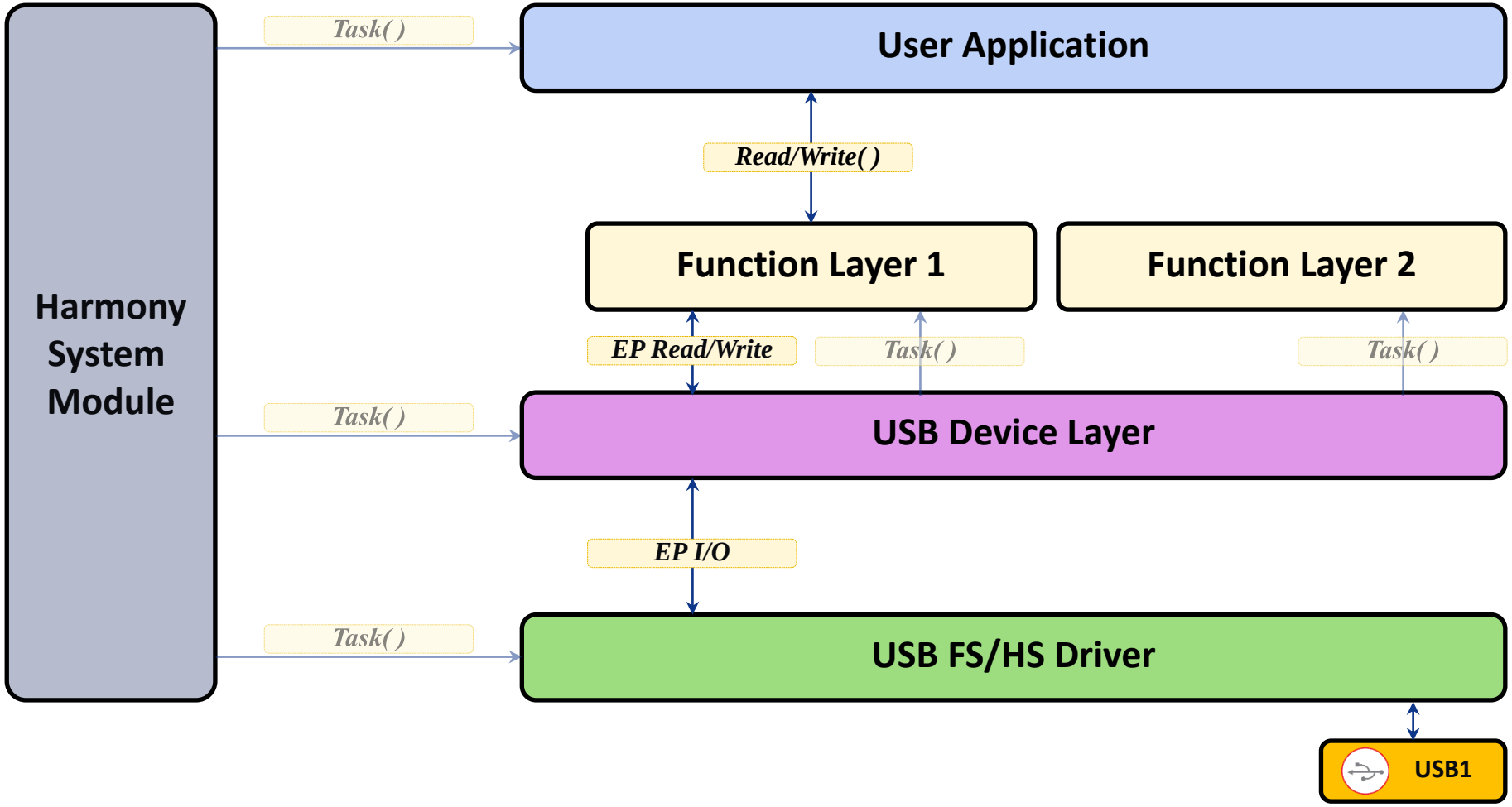
USB Stack Abstraction Model - Register Function Event Handler



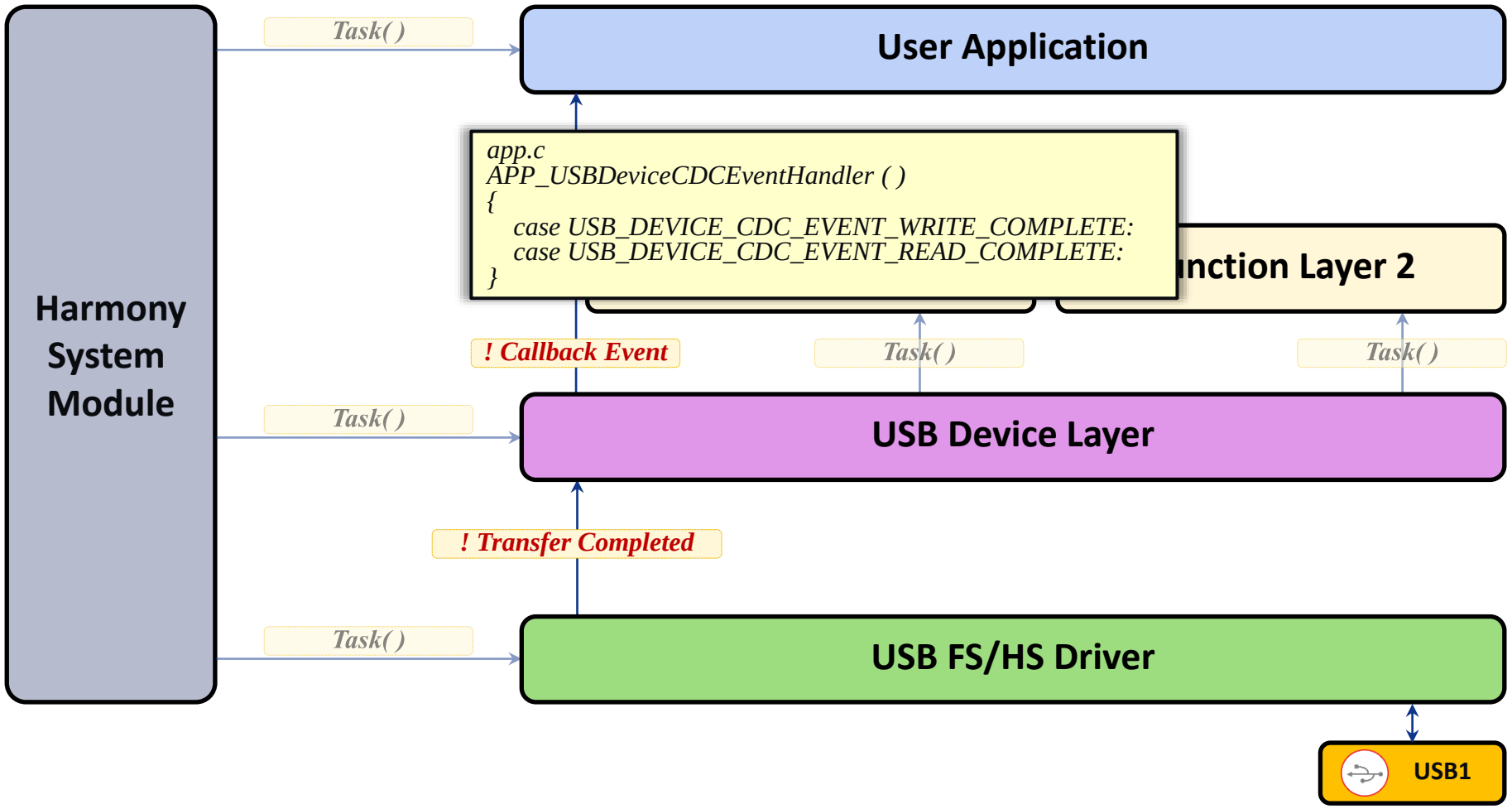
USB Stack Abstraction Model - Function Service Ready



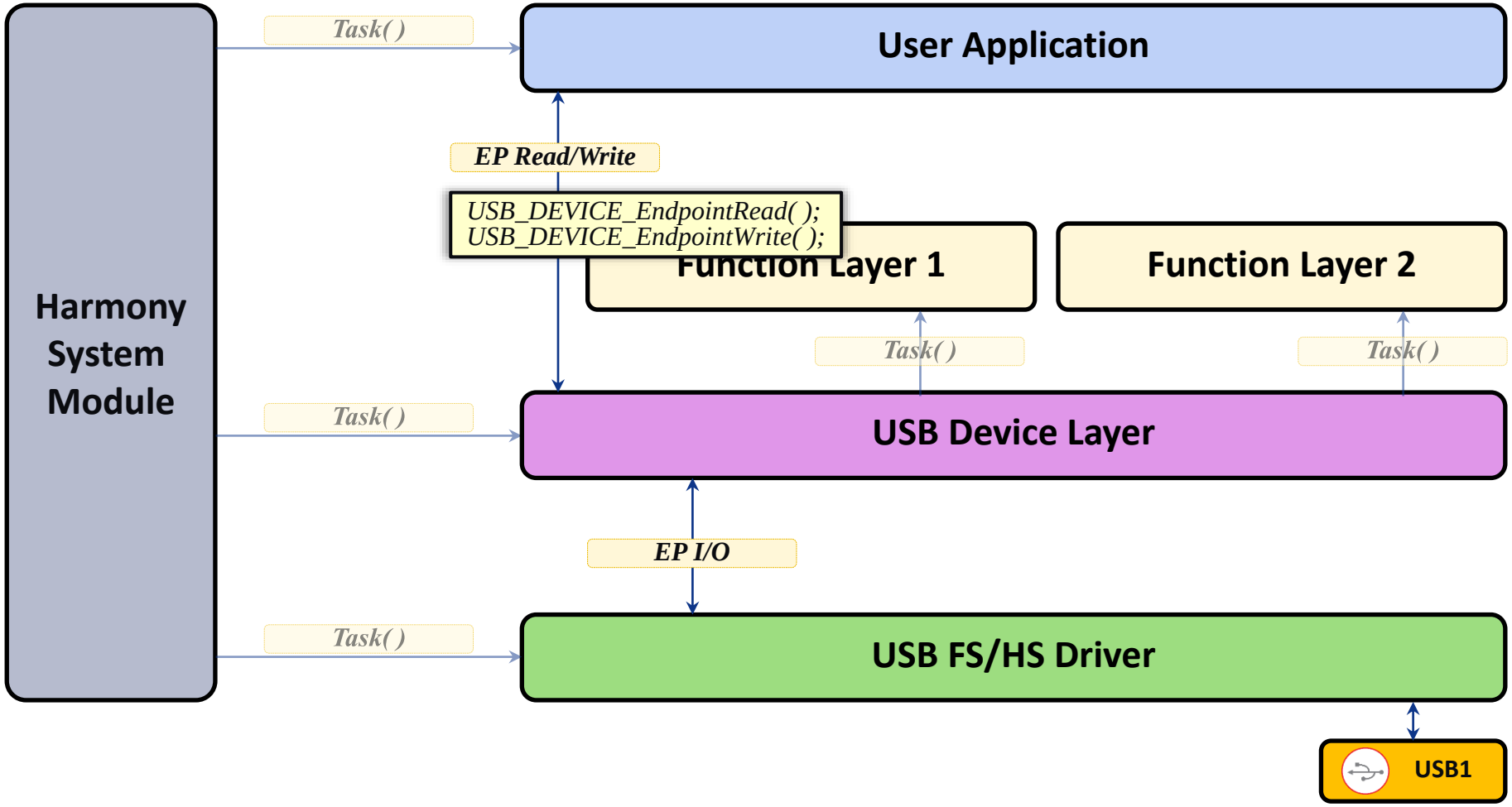
USB Stack Abstraction Model - Functionally Read/Write



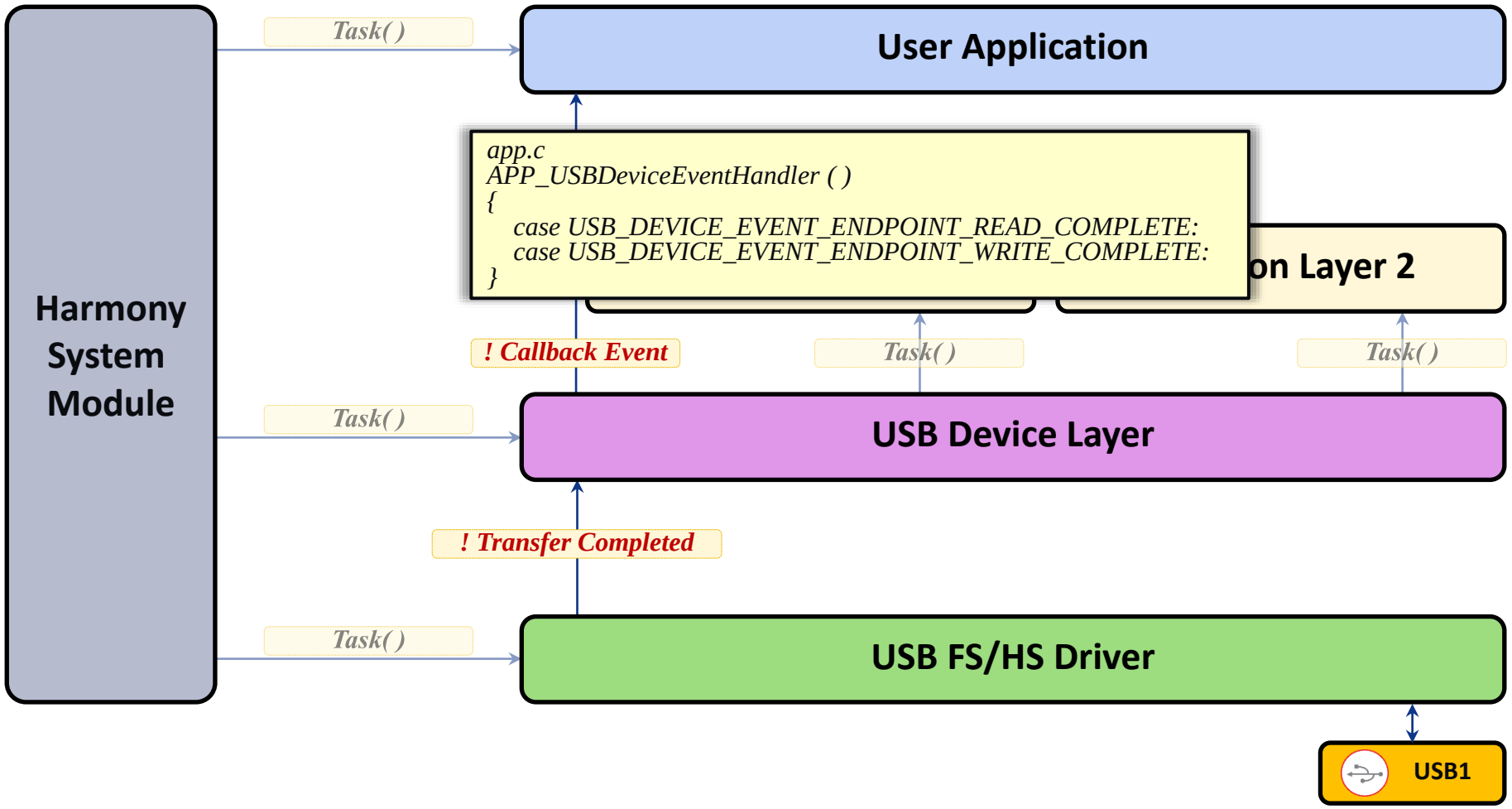
USB Stack Abstraction Model - Functionally Read/Write



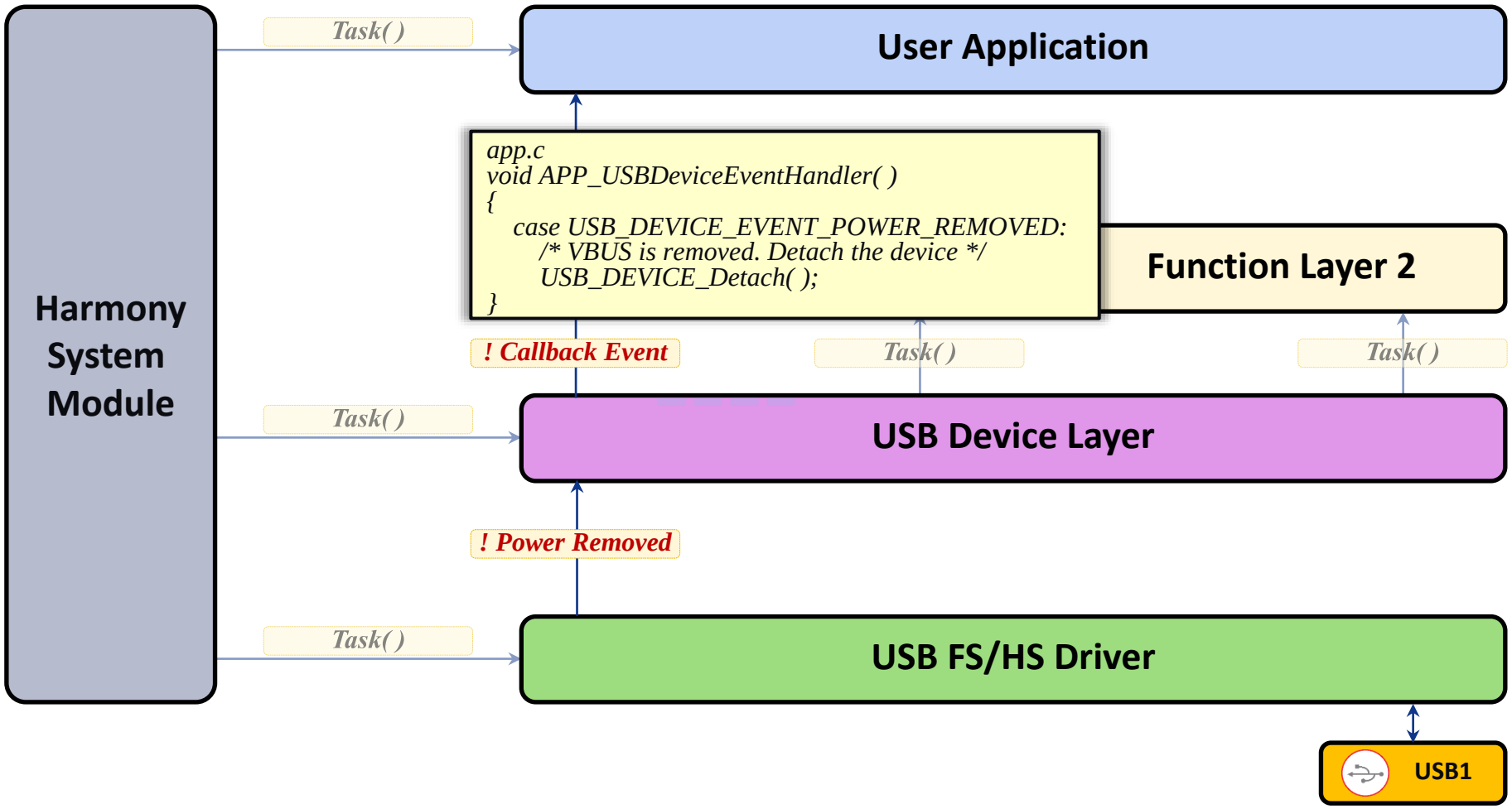
USB Stack Abstraction Model - Device Layer Read/Write



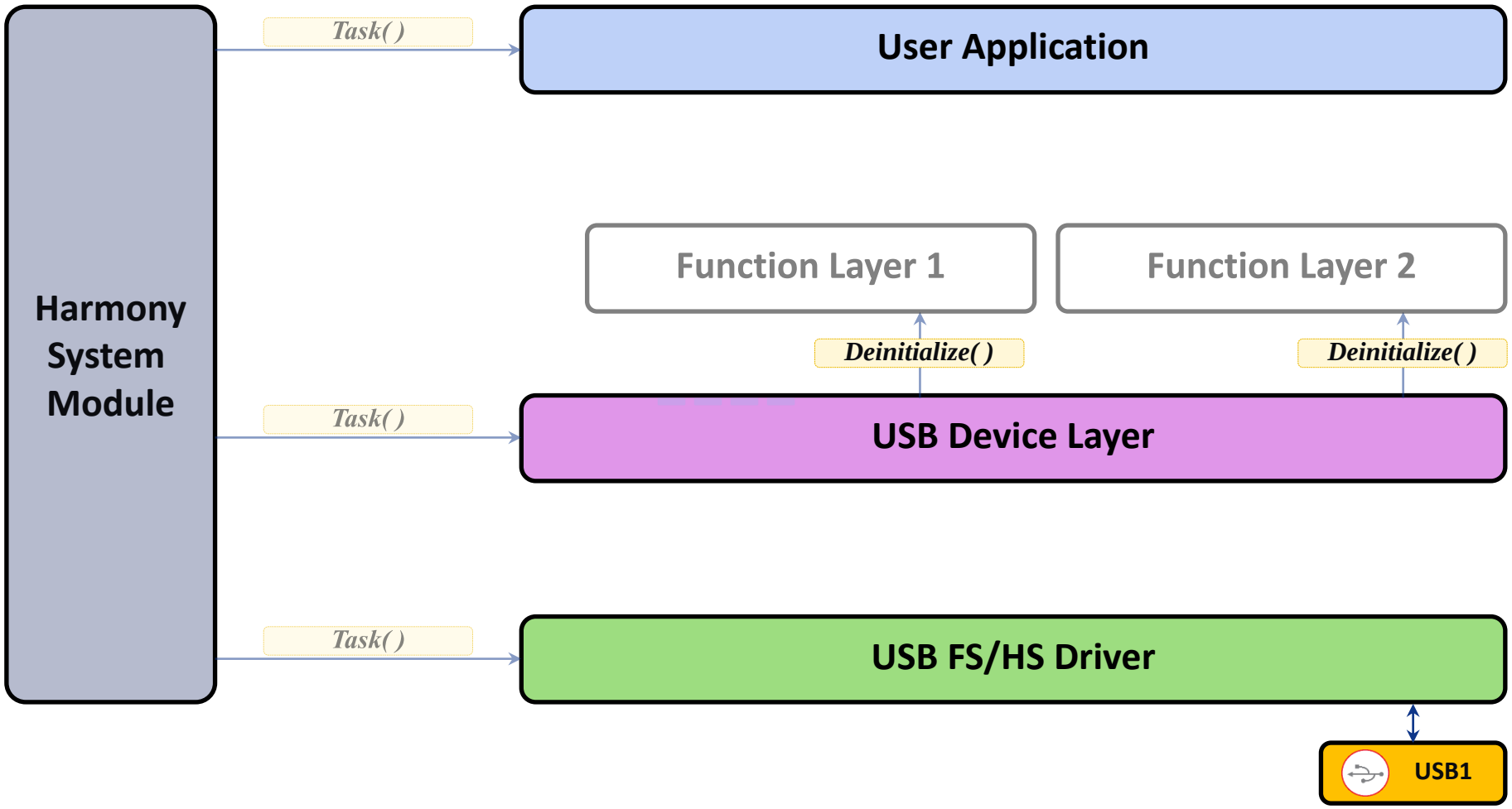
USB Stack Abstraction Model - Device Layer Read/Write



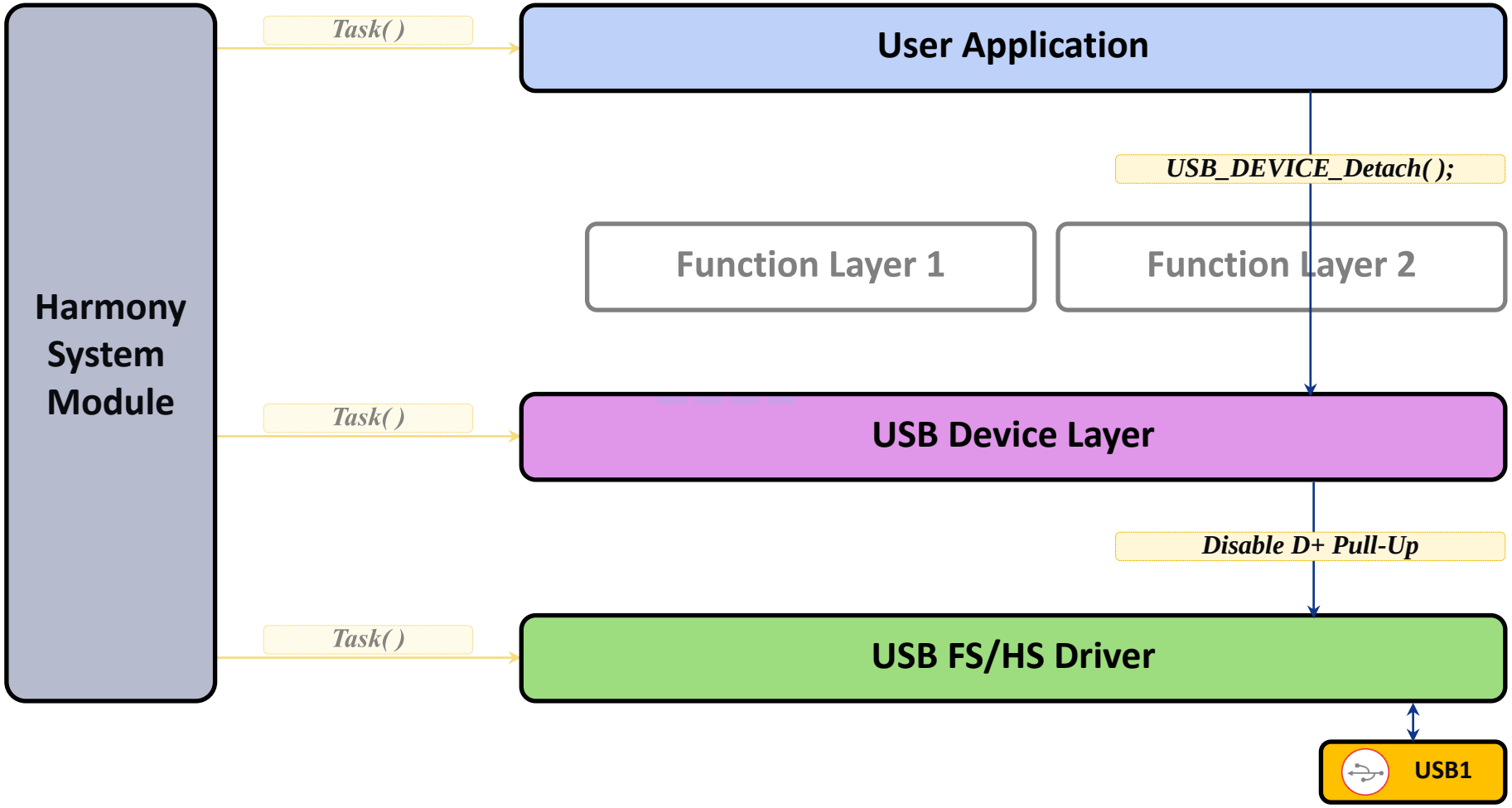
USB Stack Abstraction Model - Power Removed



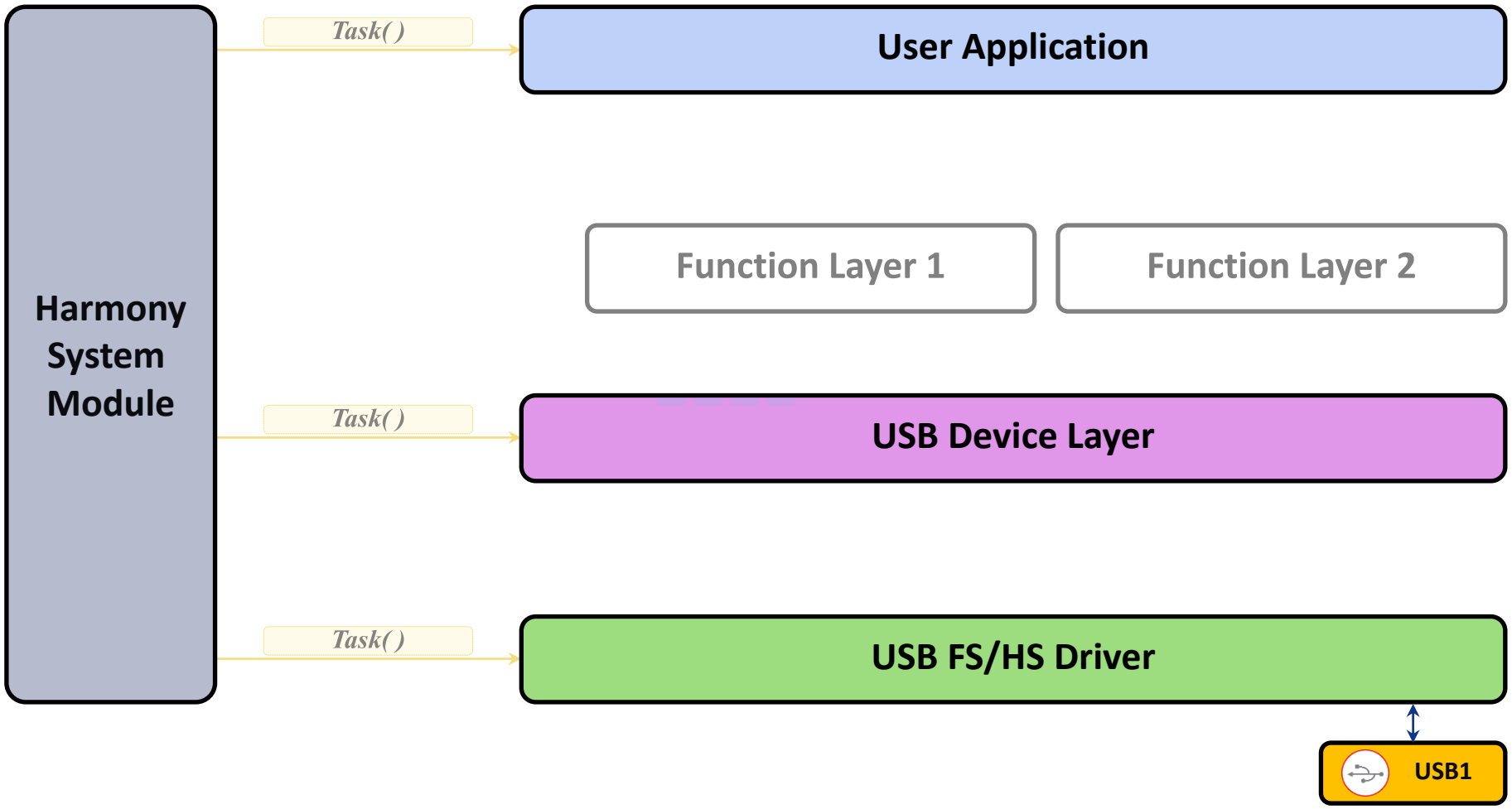
USB Stack Abstraction Model - Function Deinitialize



USB Stack Abstraction Model - Detach



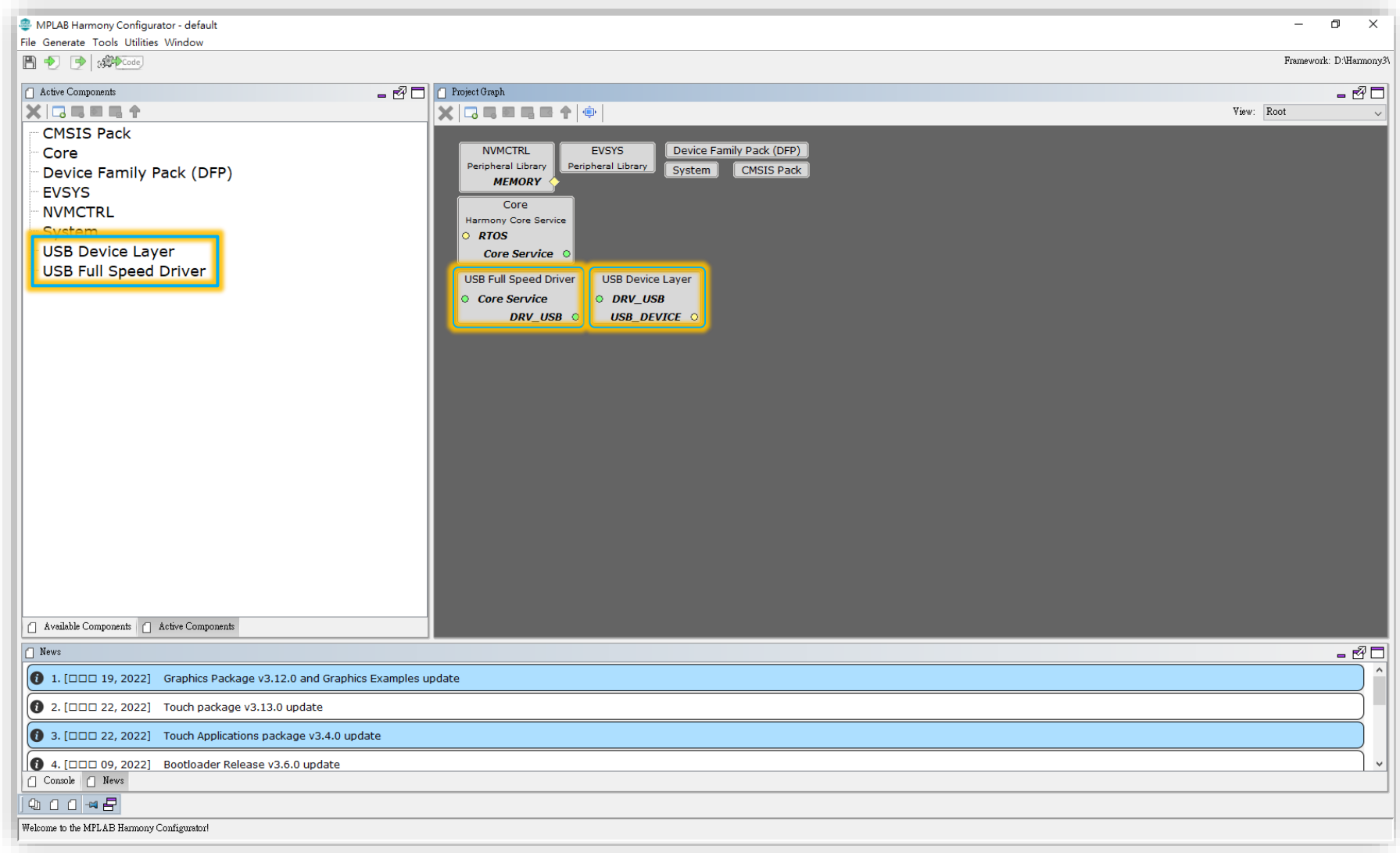
USB Stack Abstraction Model - Disconnection



Harmony USB Device
Device Layer Configurator

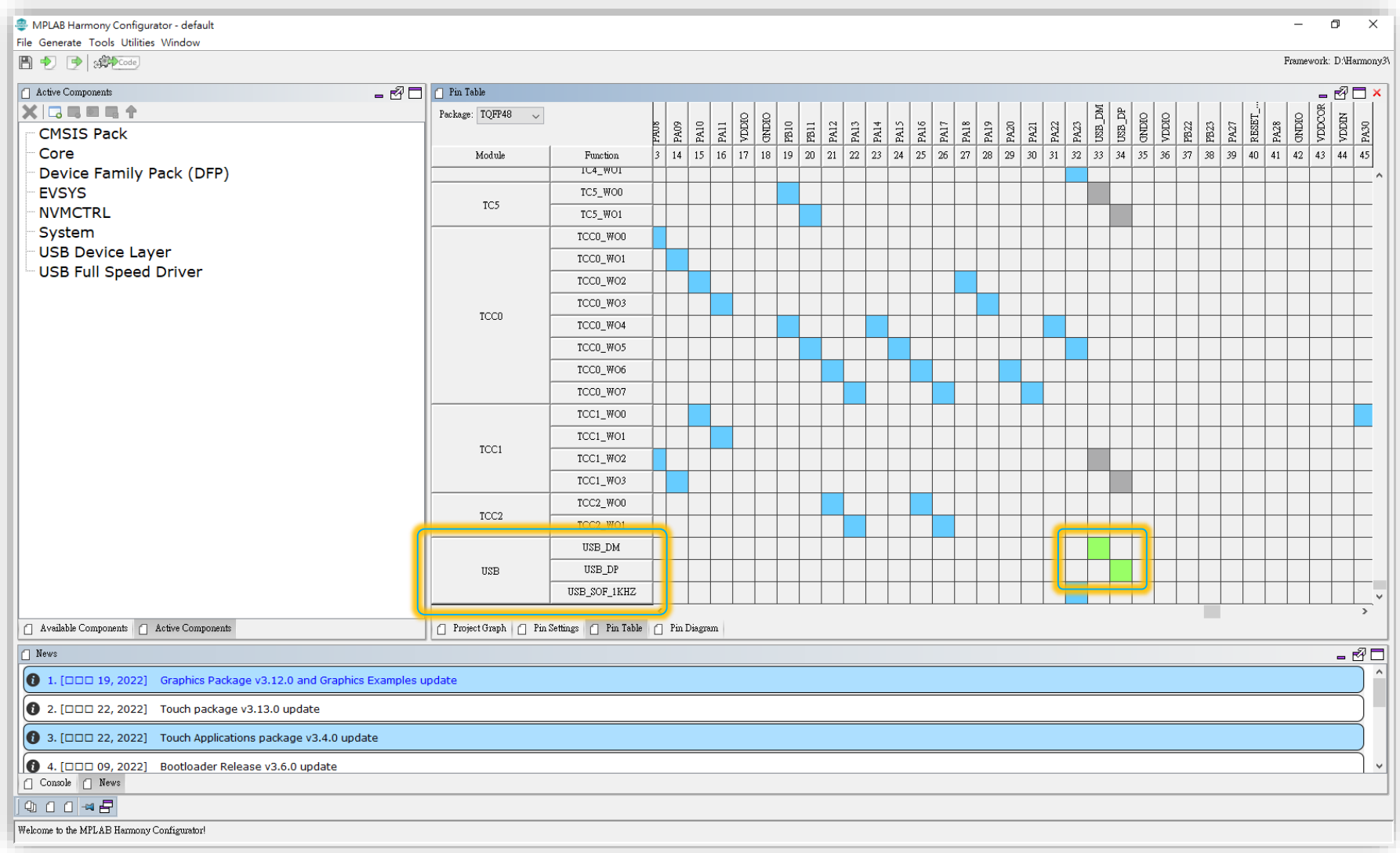
Harmony USB Device - Device Layer Configurator

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



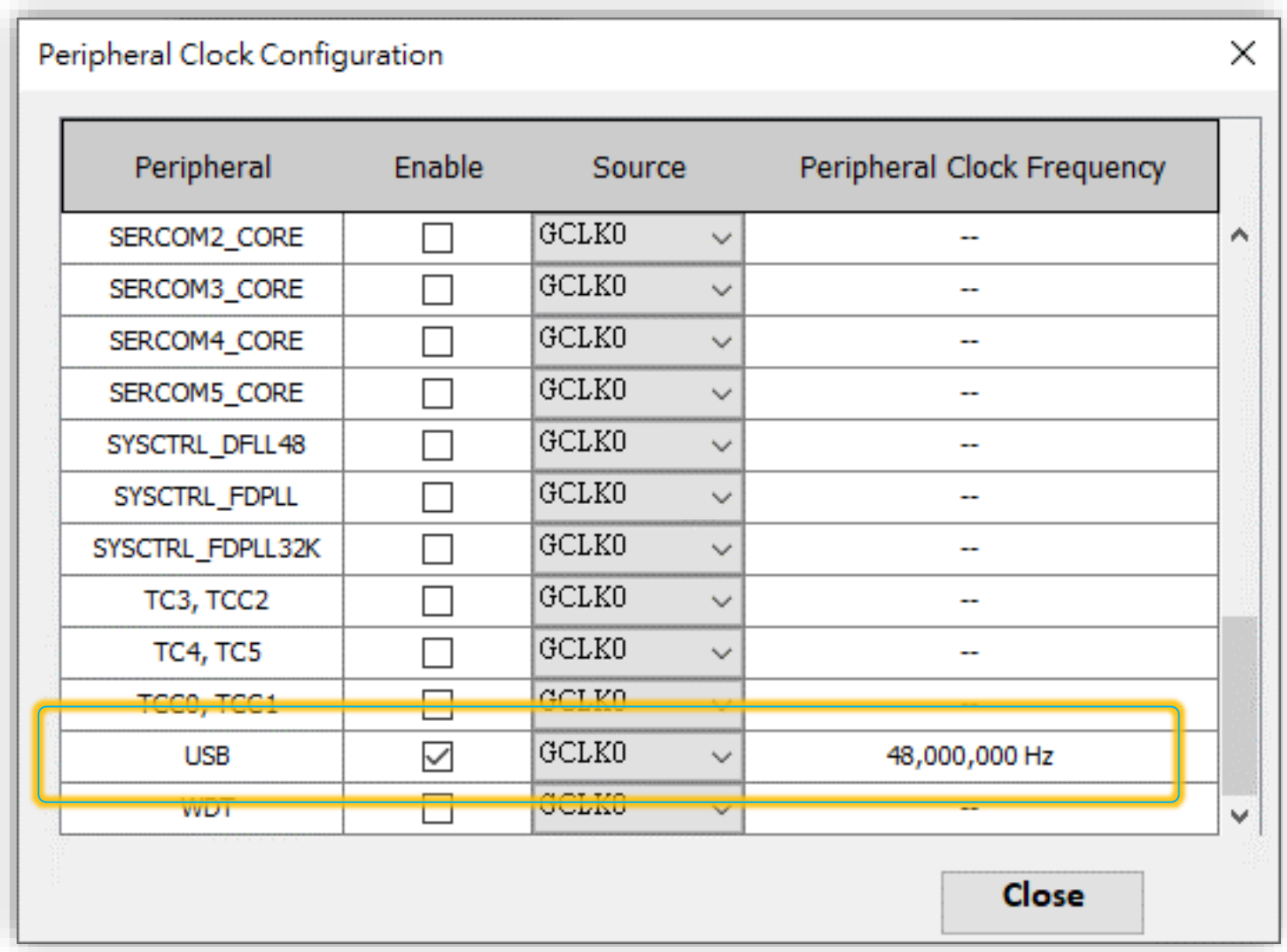
Harmony USB Device - Device Layer Configurator

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



Harmony USB Device - Device Layer Configurator

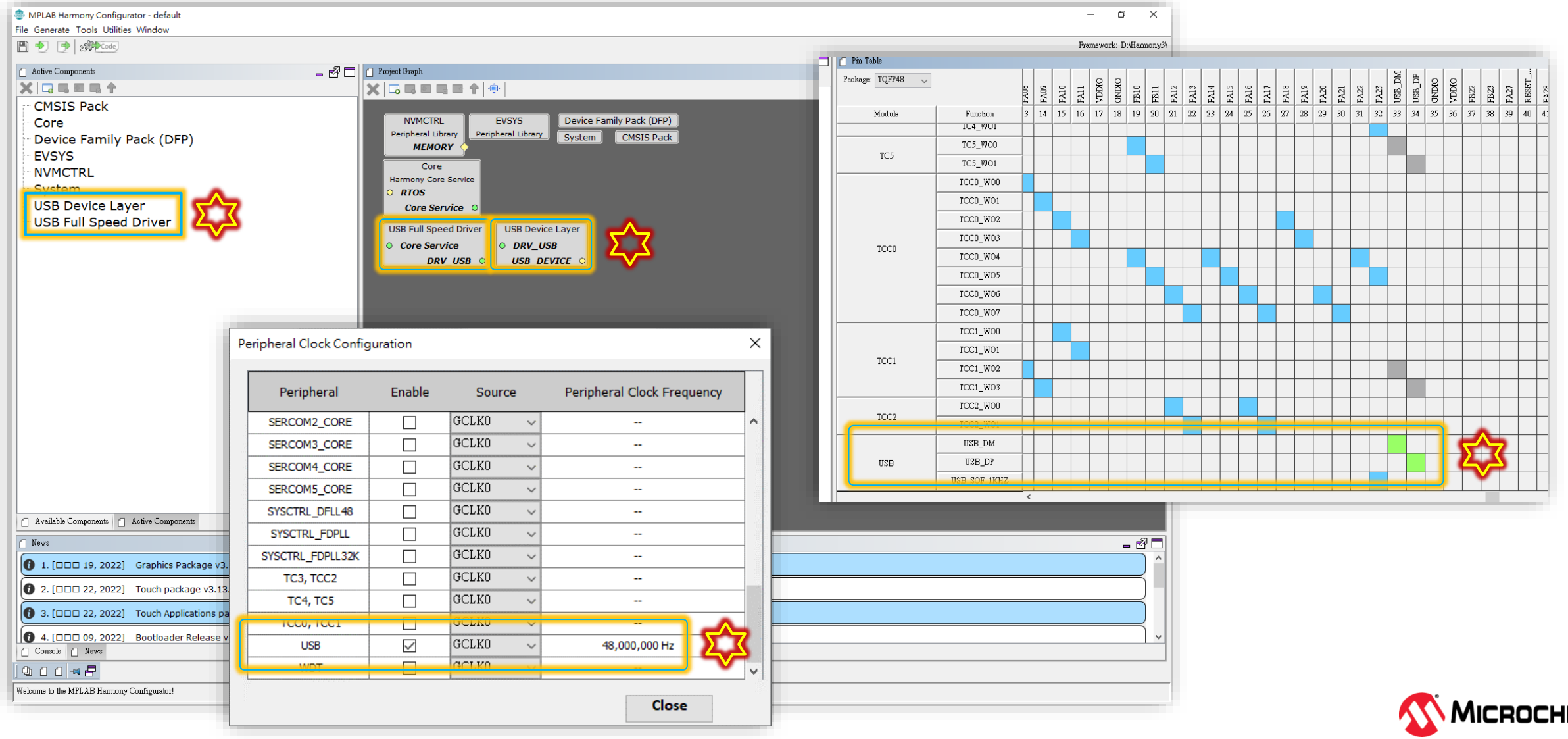
Confirm & Enable GCLK for USB Module



🛠 Lab4 USB Device
Device Layer Initialize

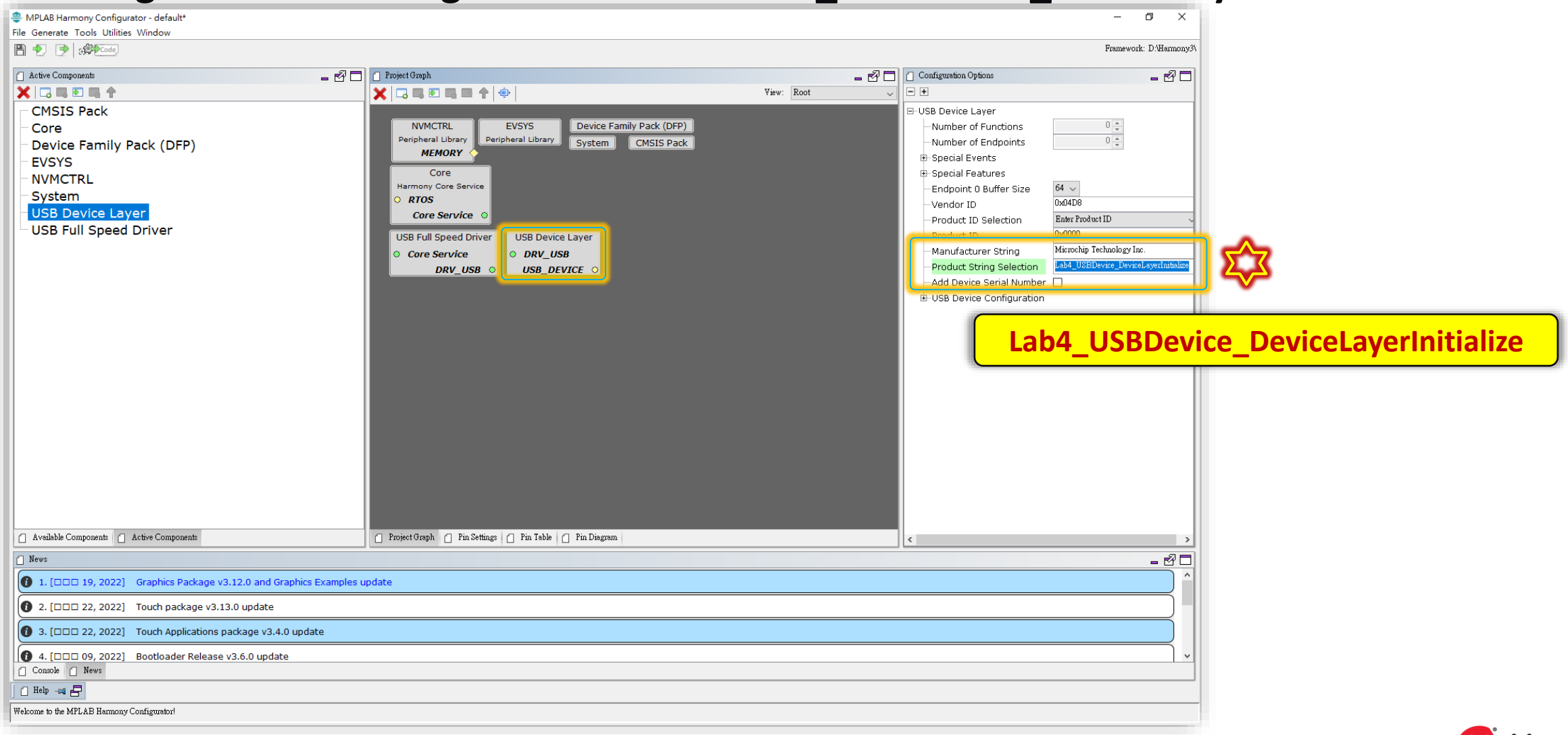
🛠 Lab4 USB Device Device Layer Initialize

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



🛠 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;
}
```

MICROCHIP

🛠 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;
}
```

MICROCHIP

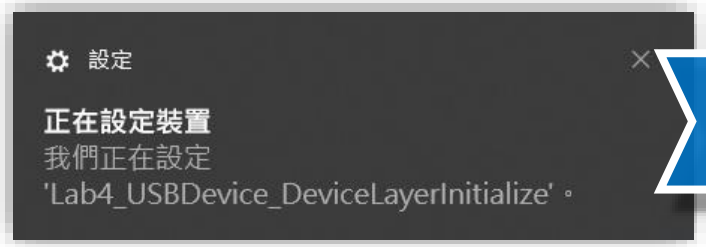
🛠 Lab4 USB Device Device Layer Initialize

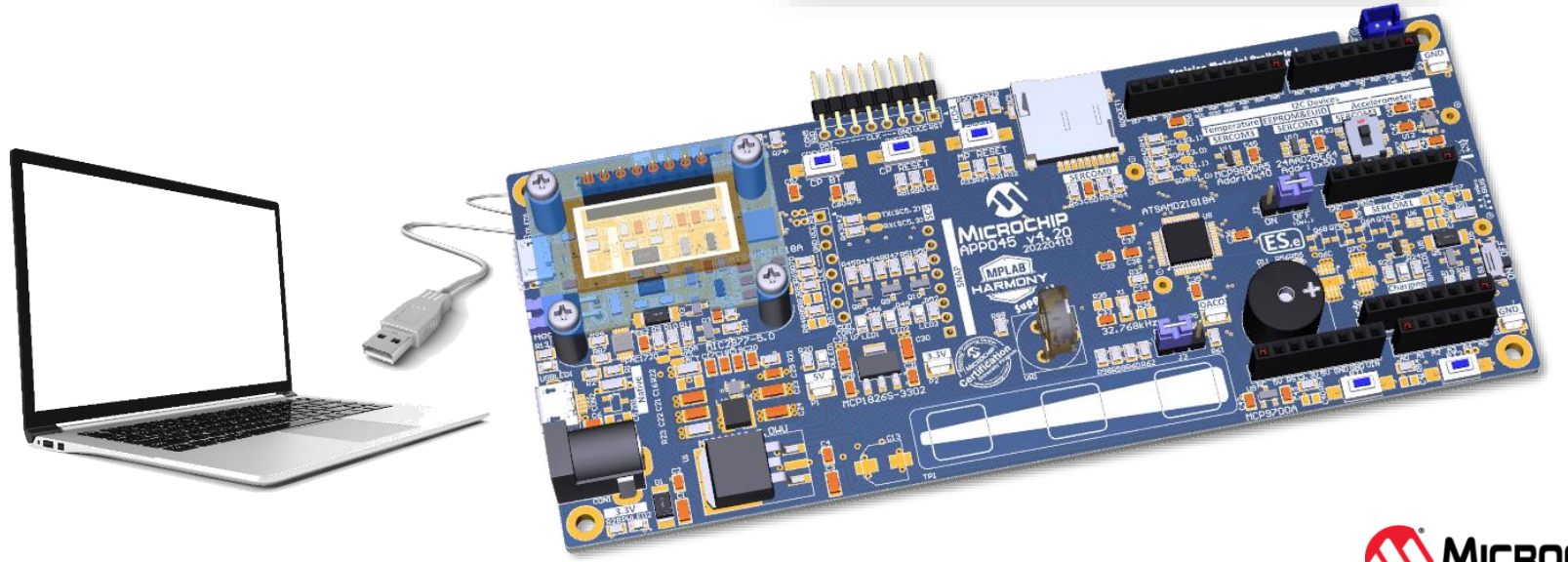
- **Implement below function at app.c.**

```
// 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;
}
```

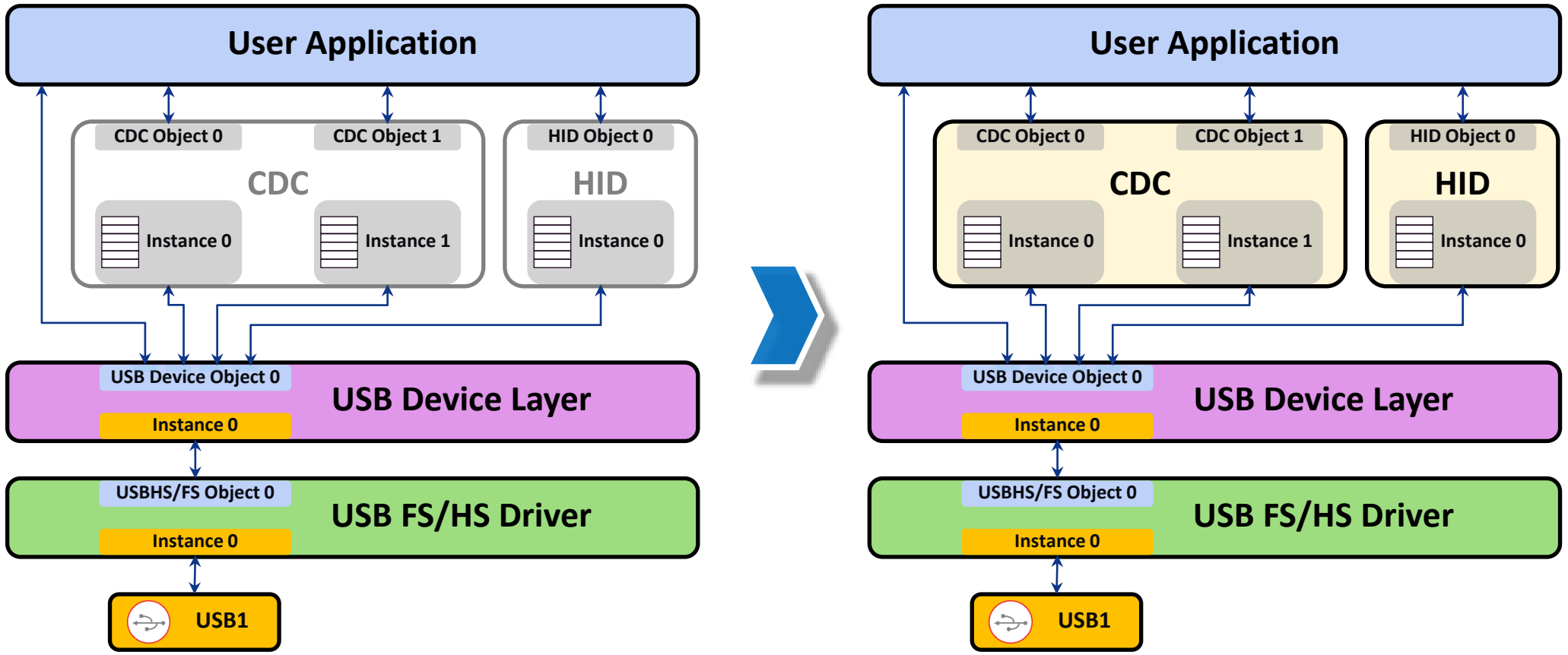
🛠 Lab4 USB Device Device Layer Initialize

- **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)**

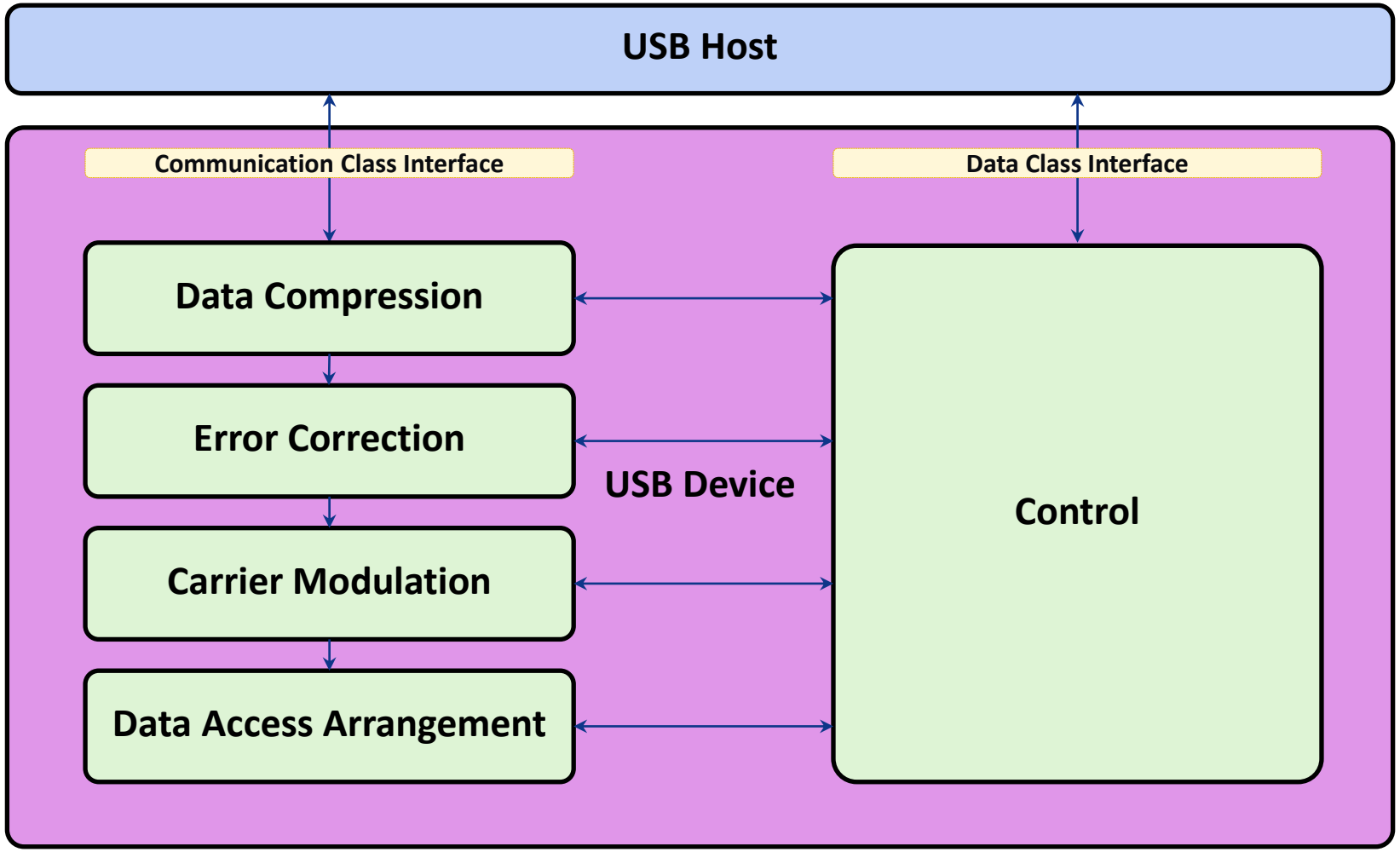
Communications Class Subclass Codes

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	[USBEEM1.0]
0Dh	Network Control Model	[USBNCM1.0]
0Dh-7Fh	RESERVED (future use)	
80-FEh	RESERVED (vendor specific)	

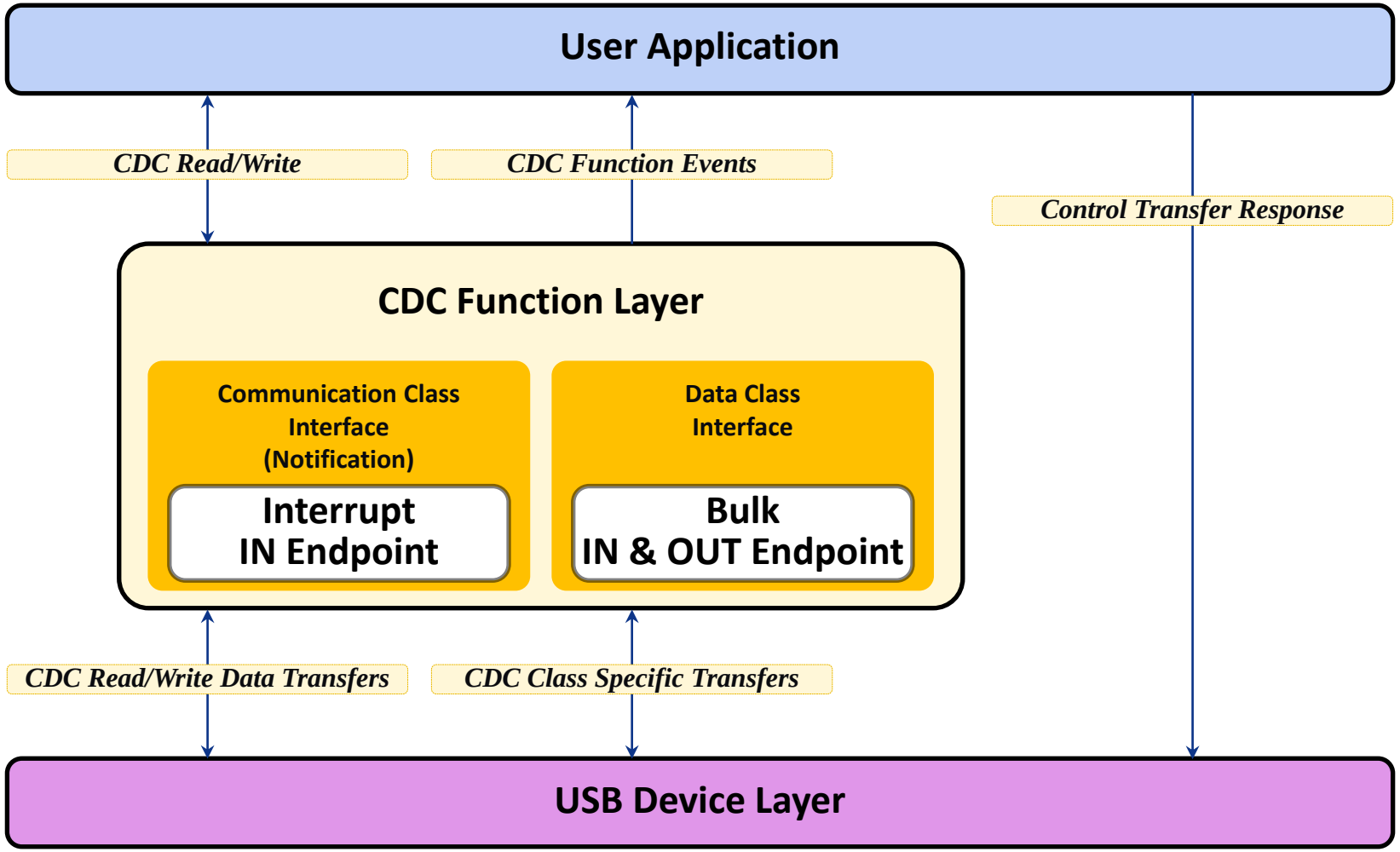
Communications Class Protocol Codes

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
    ...
}
```

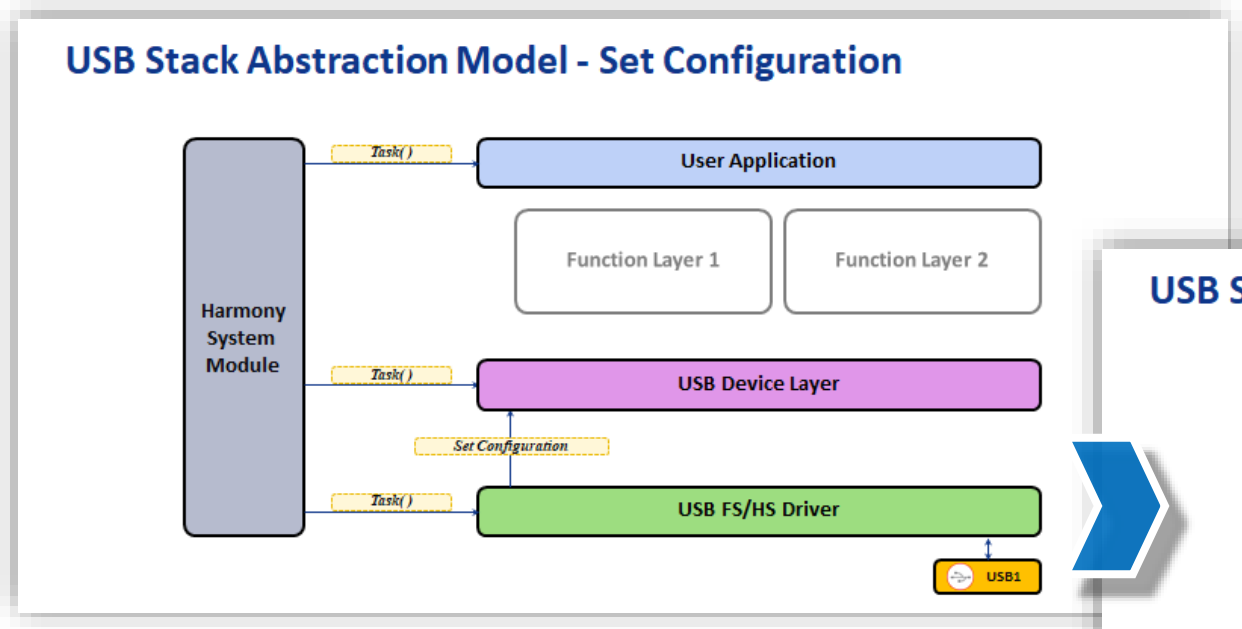
```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // Descriptor Type is Int
    0, // Interface Number
    0x00, // Alternate Setting Number
    0x01, // Number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

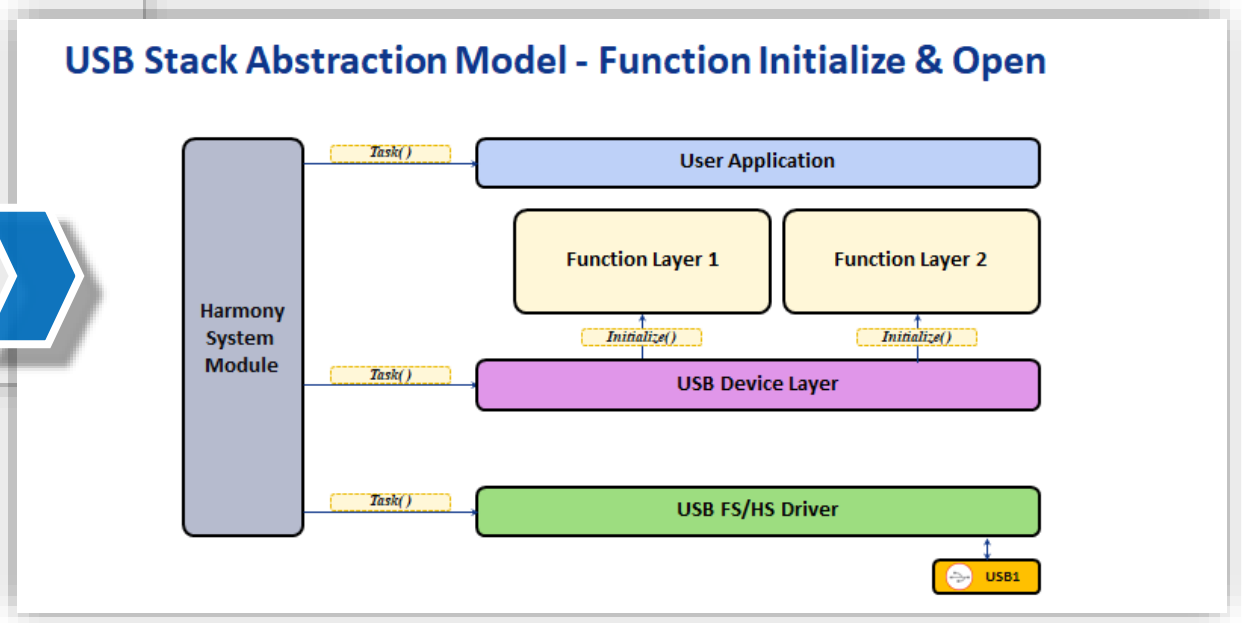
```
usb_device_init_data.c
const uint8_t fullSpeedConfigurationDescriptor[] =
{
    ...
    /* Interface Descriptor */
    0x09, // Size of this descriptor in bytes
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    1, // Interface Number
    0x00, // Alternate Setting Number
    0x02, // 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
    ...
}
```

MICROCHIP

Review USB Stack Abstraction Model

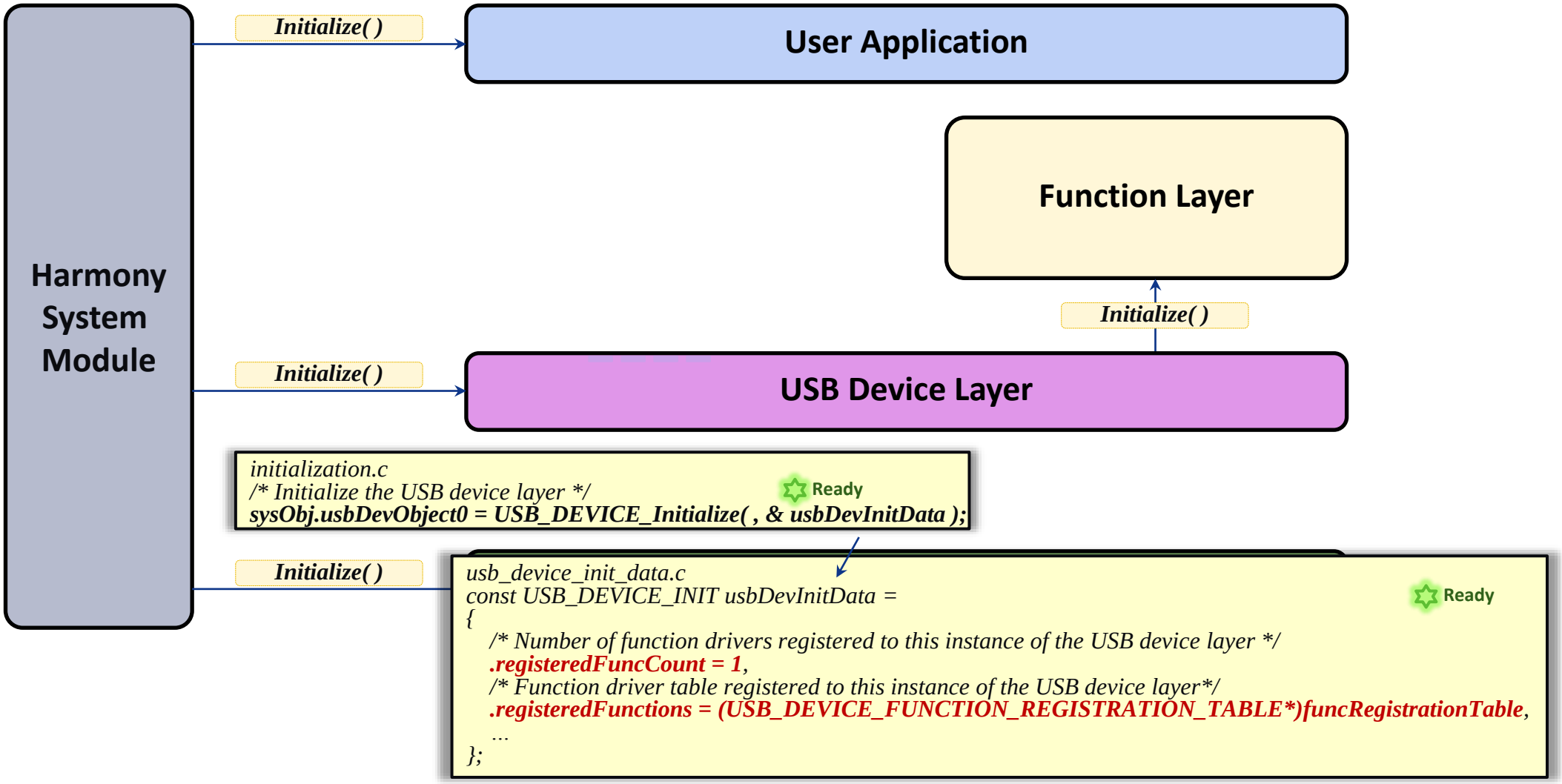
- Ideally, Function Layer be initialized after “Set Configuration” state.
- So, function layer needs initialize and set function event handler at there.





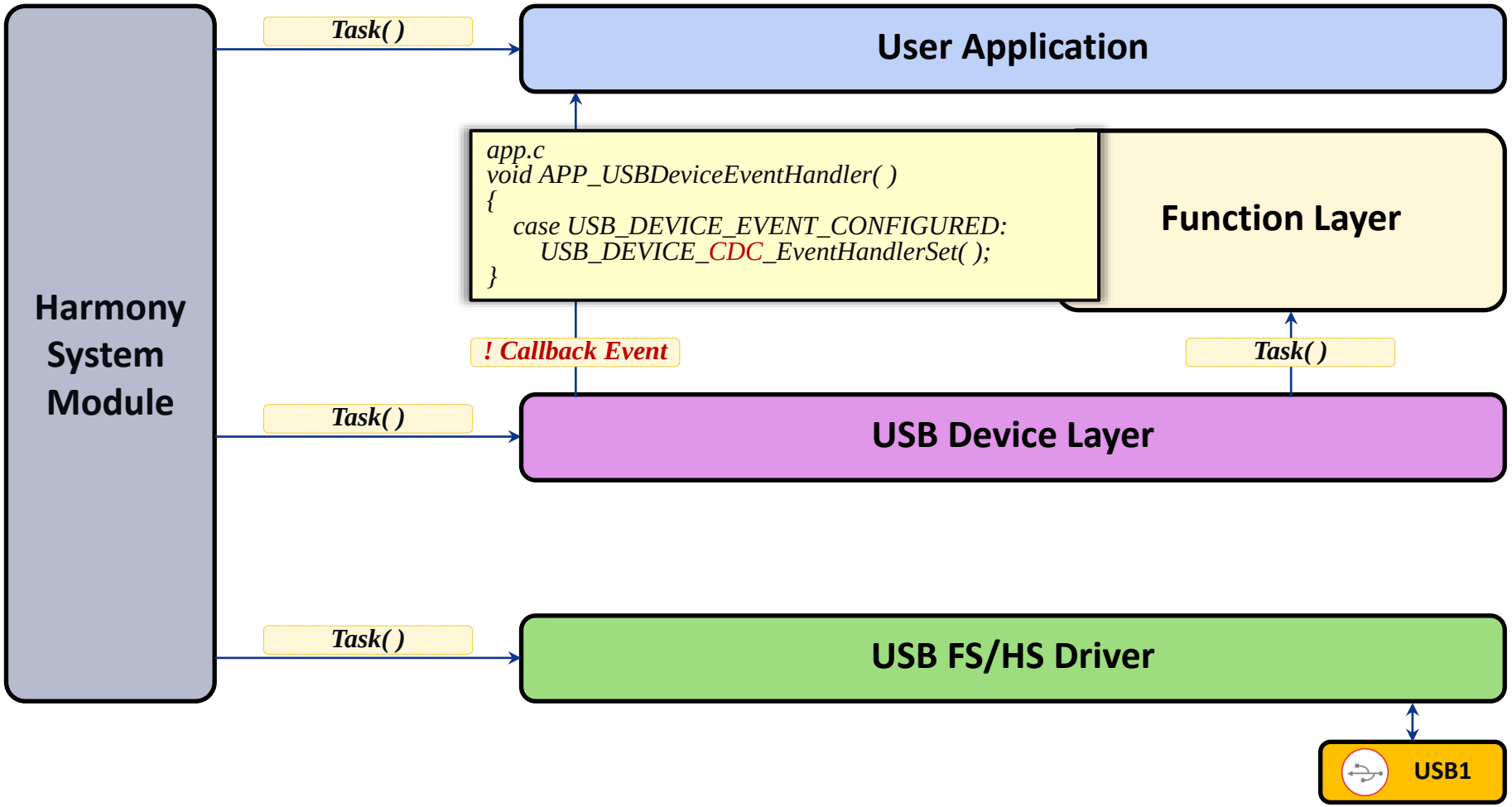
Review USB Stack Abstraction Model

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

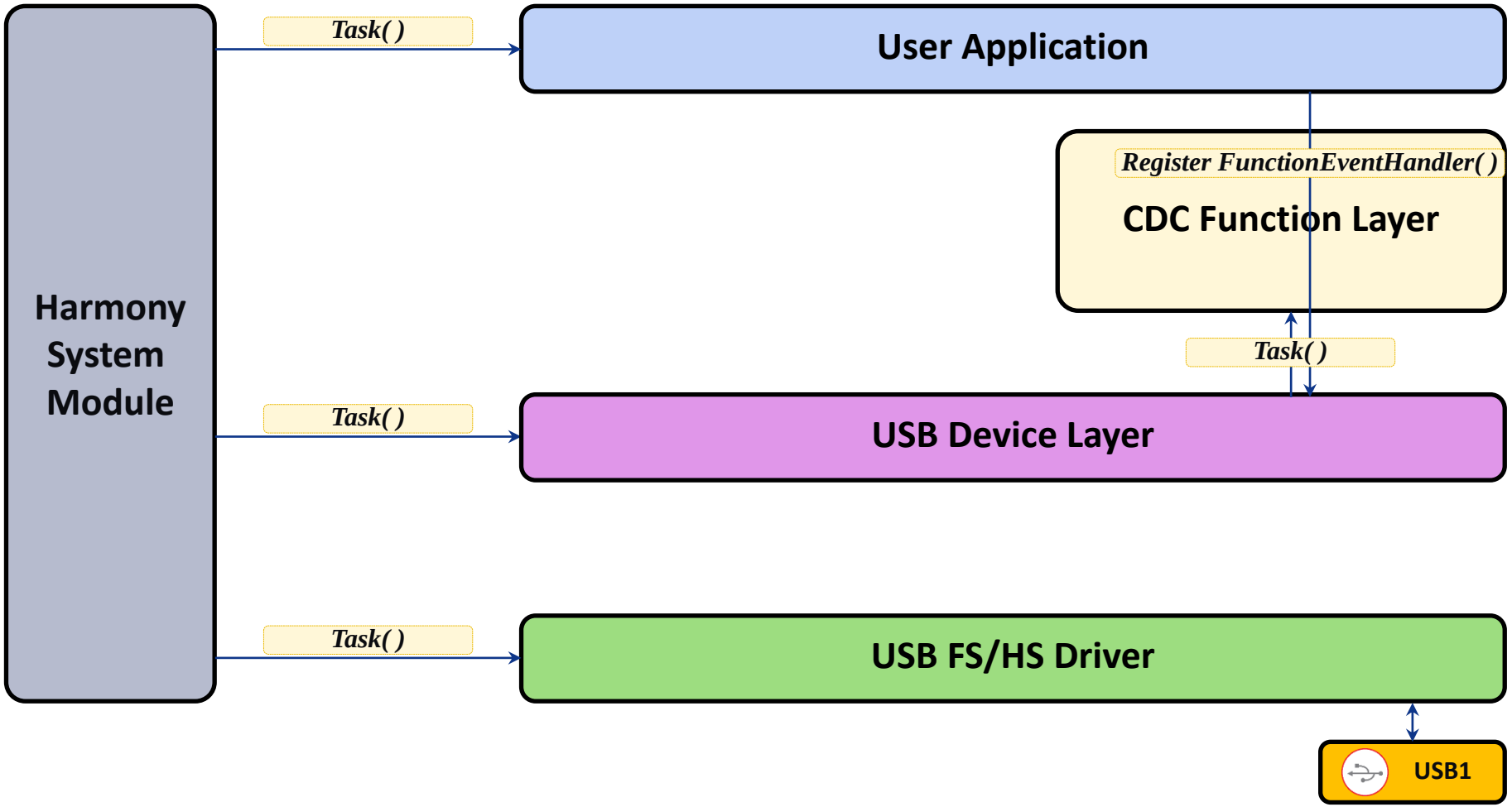


USB Stack Abstraction Model - Register Function Event Handler

- 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( )	Called by the device layer during Set Configuration Processing.
_USB_DEVICE_CDC_Initialization( )	Called by the device layer during Set Configuration Processing.
_USB_DEVICE_CDC_Deinitialization( )	Deinitializes the function driver instance.
_USB_DEVICE_CDC_EndpointDisable( )	Disabled USB Device CDC endpoints.
_USB_DEVICE_CDC_ControlTransferHandler( )	Control Transfer Handler for class specific control transfer.
_USB_DEVICE_CDC_ReadIRPCallback( )	for IRPs submitted through the USB_DEVICE_CDC_Read().
_USB_DEVICE_CDC_WriteIRPCallback( )	for IRPs submitted through the USB_DEVICE_CDC_Write().
_USB_DEVICE_CDC_SerialStateSendIRPCallback( )	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( )	Requests a data read from the CDC Function Driver Layer.
USB_DEVICE_CDC_Write( )	Requests a data write to the USB Device CDC Function Driver Layer.
USB_DEVICE_CDC_ReadPacketSizeGet( )	Return read packet size.
USB_DEVICE_CDC_WritePacketSizeGet( )	Return write packet size.
USB_DEVICE_CDC_SerialStateNotificationSend( )	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;
}
```

MICROCHIP

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
);
```

MICROCHIP

🛠 Lab5 USB Device CDC Class - Basic TX and RX

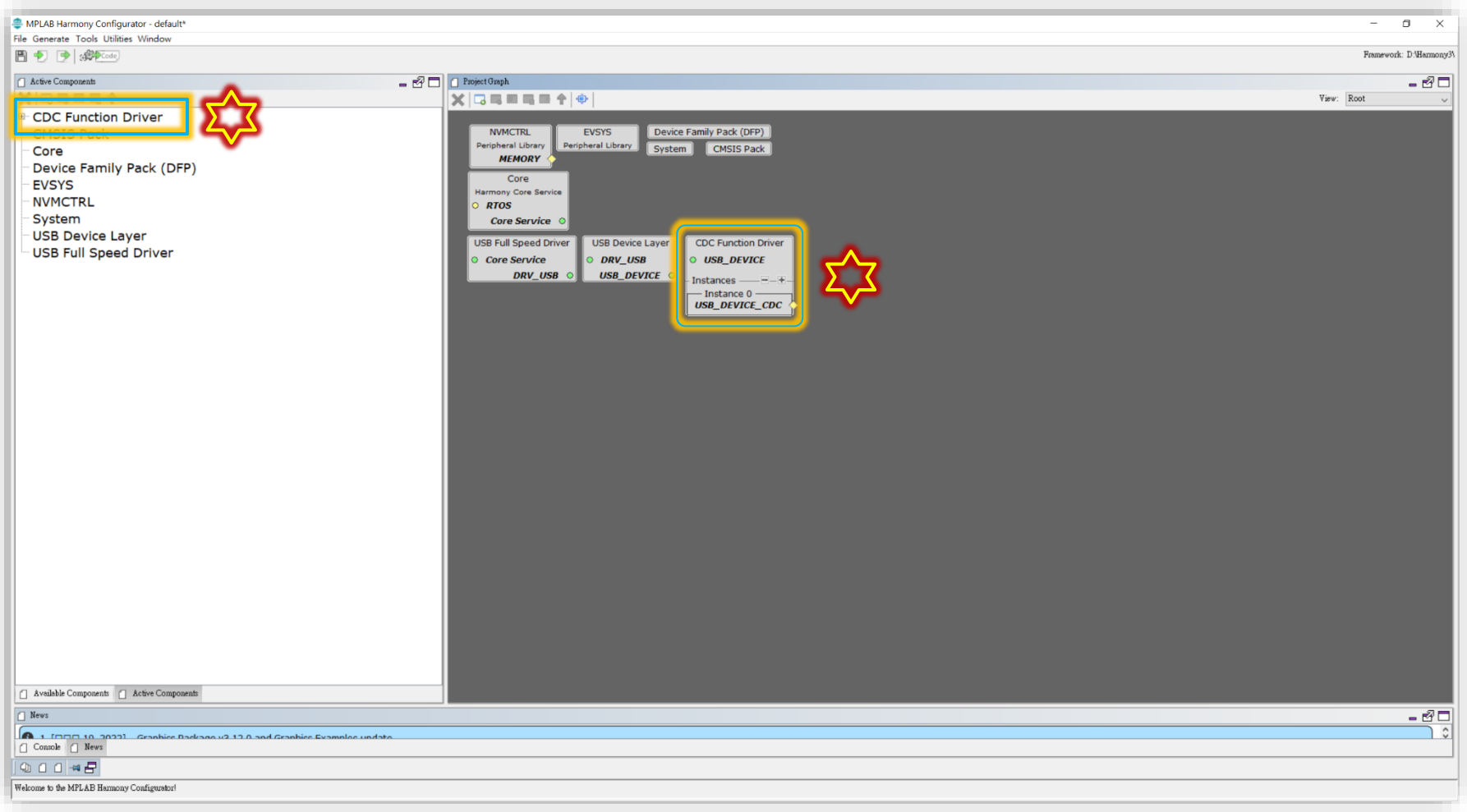
🛠 Lab5 USB Device CDC Class - Basic TX and RX

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

- **Let’s go !**

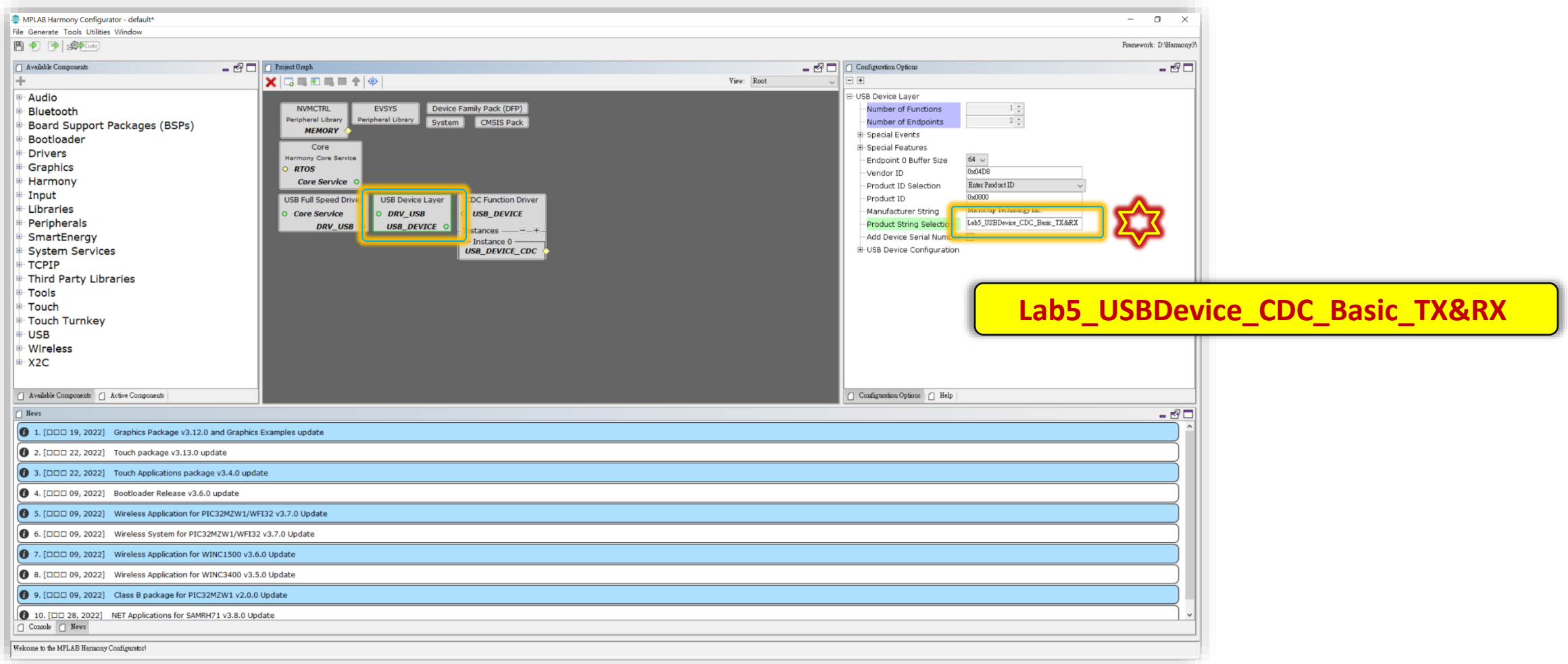
🛠 Lab5 USB Device CDC Class - Basic TX and RX

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



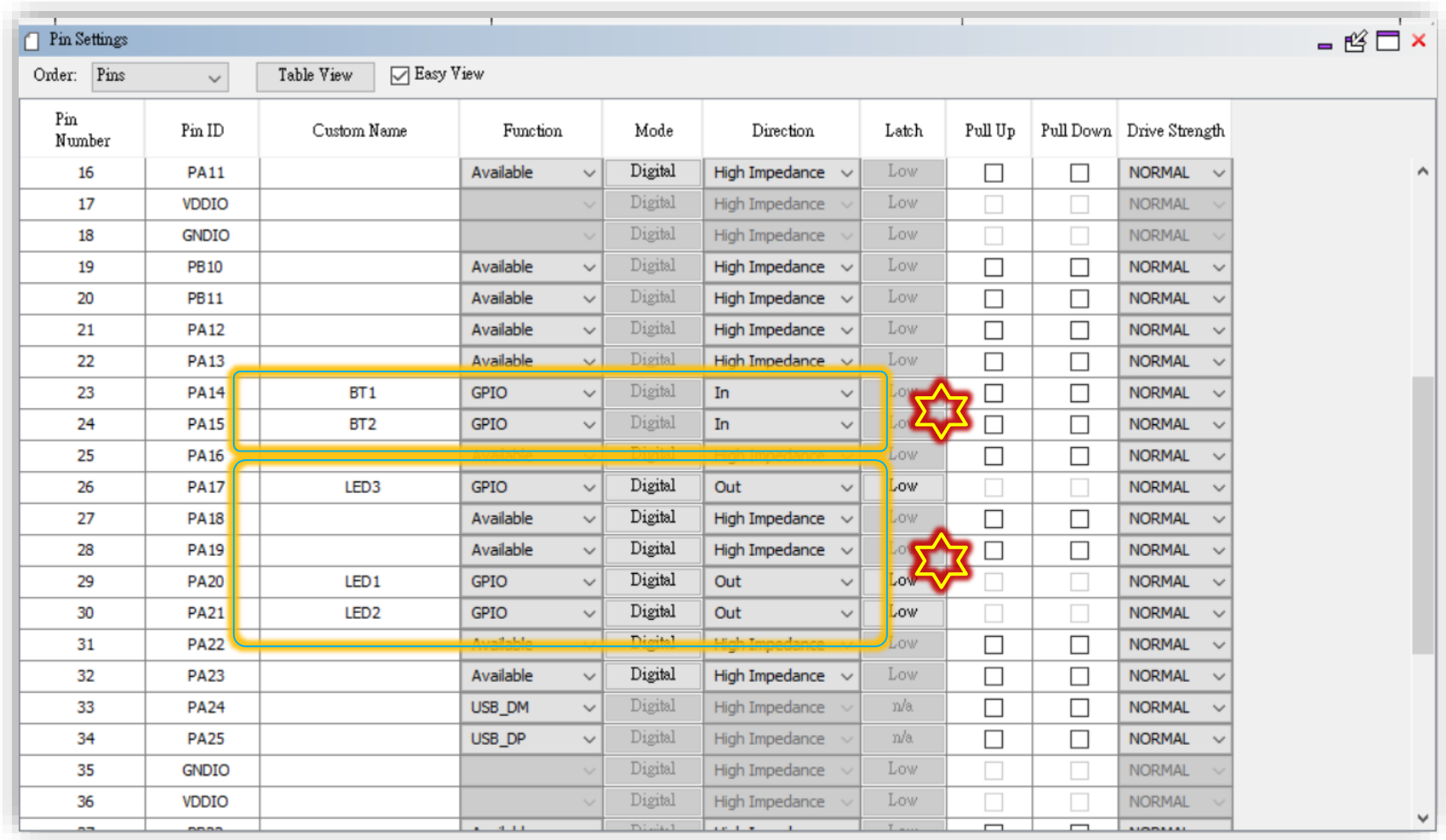
🛠 Lab5 USB Device CDC Class - Basic TX RX

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



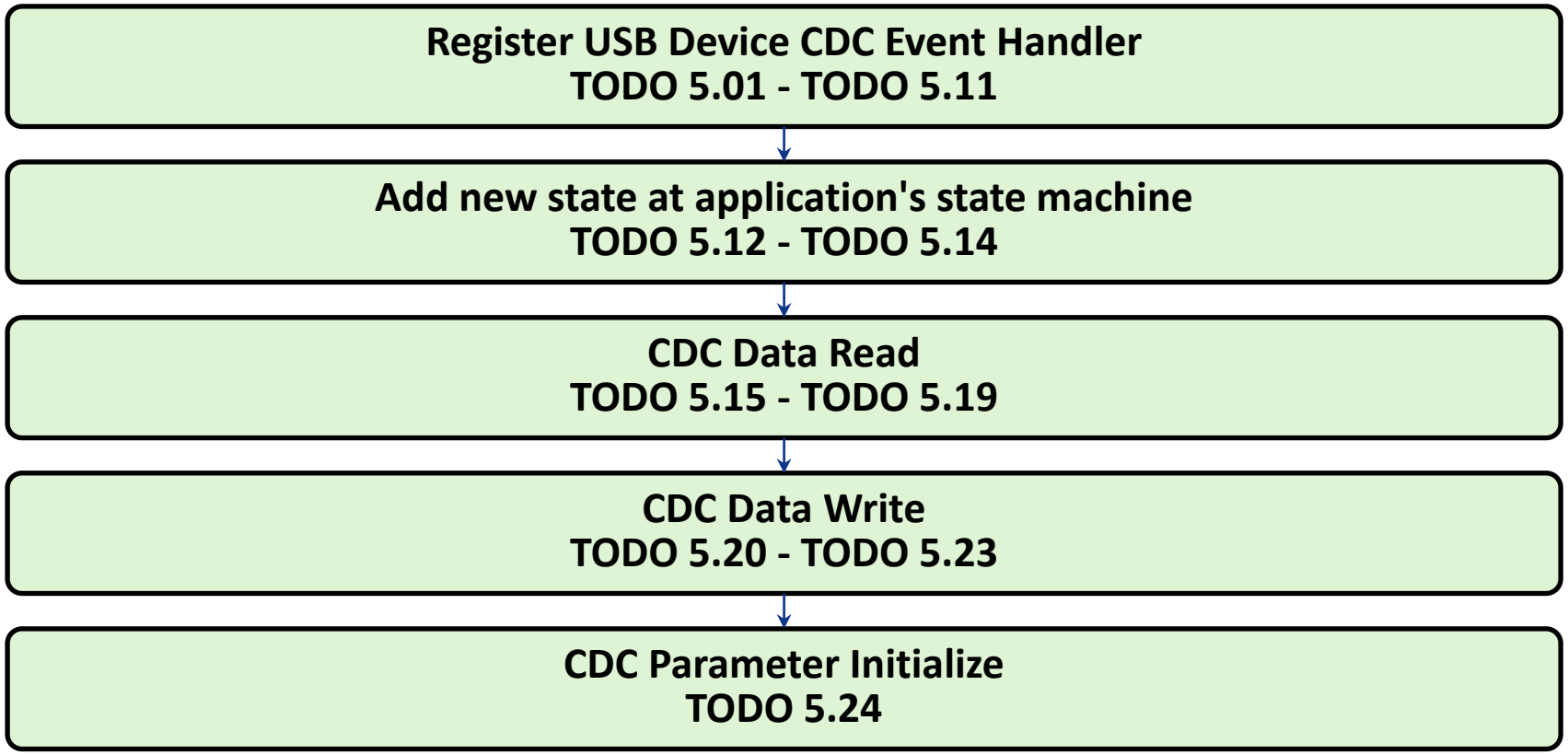
🛠 Lab5 USB Device CDC Class - Basic TX RX

- **Set LEDs and BTs in Pin Settings**



🛠 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 * controlLineStateData;

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 * controlLineStateData;
    } 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.controlLineStateData = (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;
}
```

MICROCHIP

🛠 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;
```

```
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_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;
    }
    }
}
```

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

MICROCHIP

🛠 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;
```

```
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;
}
```

```
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;
    }
    }
}
```

```
//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);
            }
        }
        ...
    }
    }
}
```

MICROCHIP

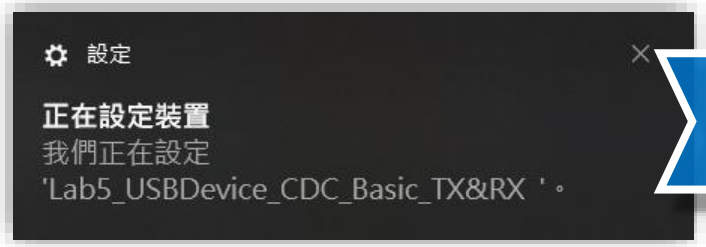
🛠 Lab5 USB Device CDC Class - Basic TX RX

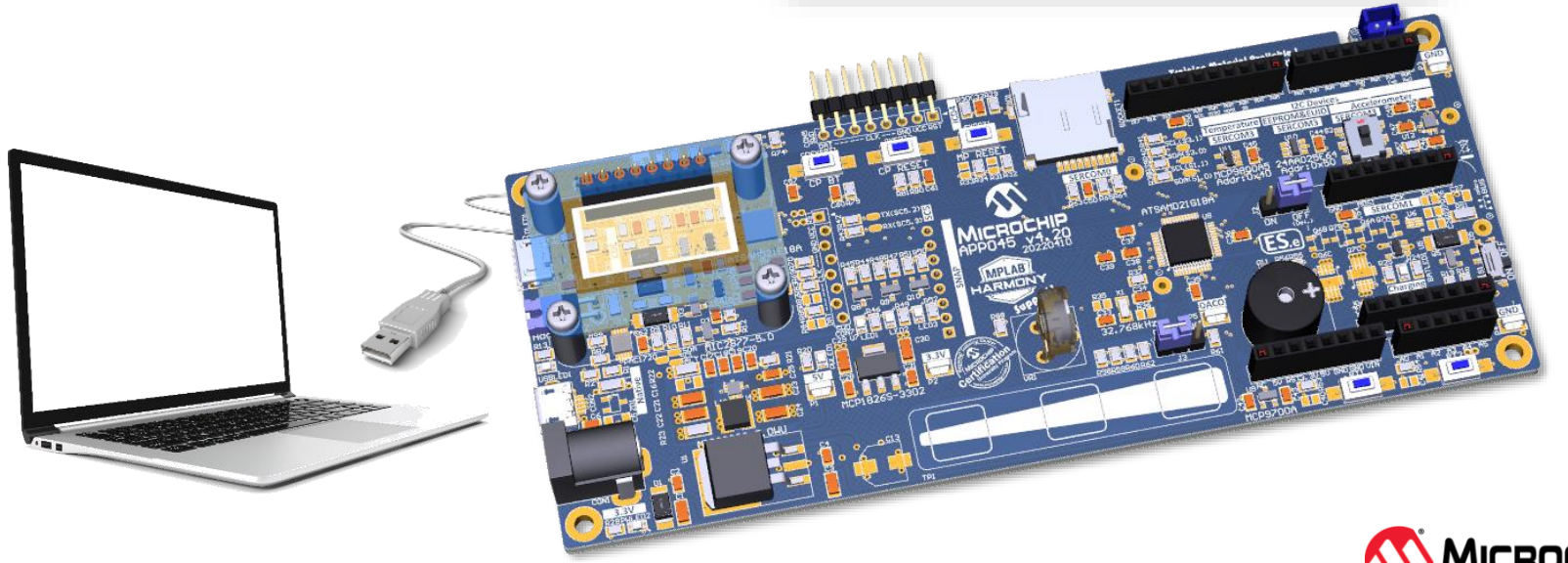
- **CDC Parameter Initialize**

```
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

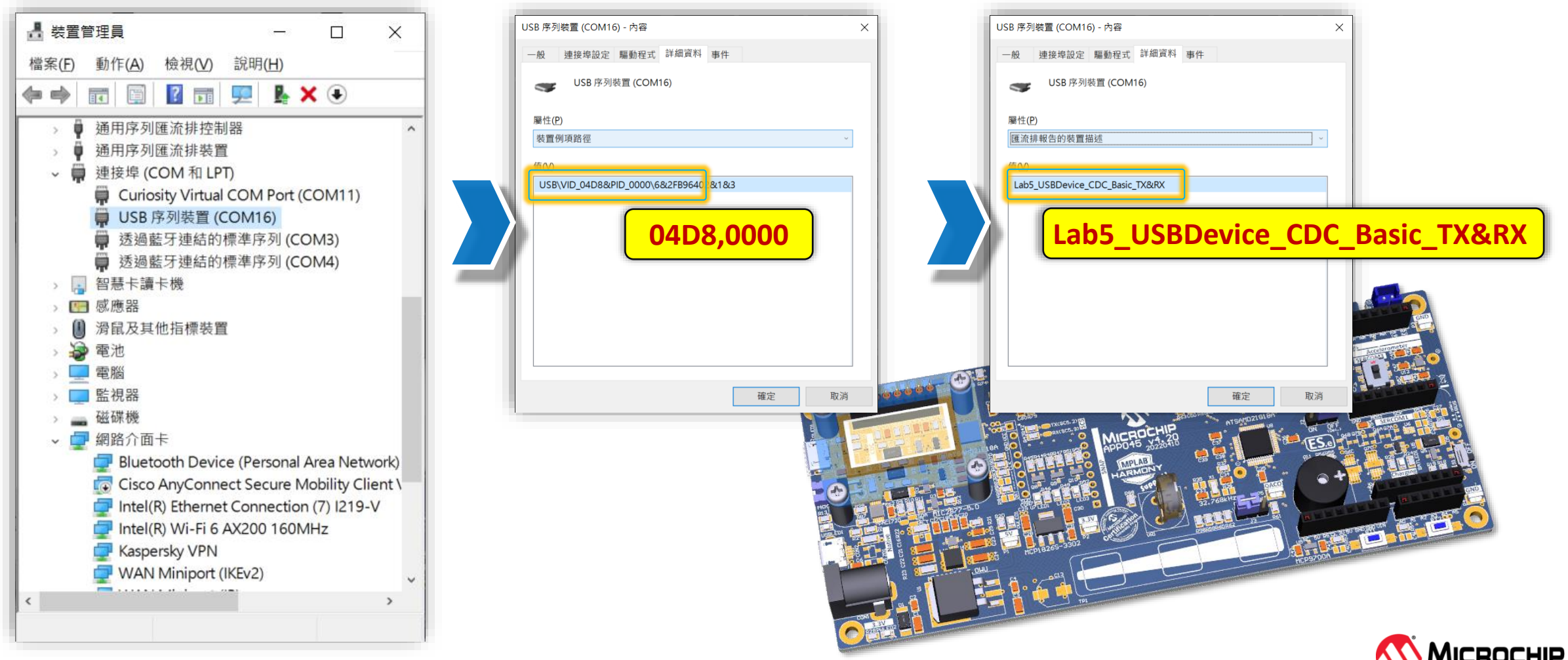
- **The device list at Device Manager, Named “USB 序列裝置(COM*n*)” After configured.**





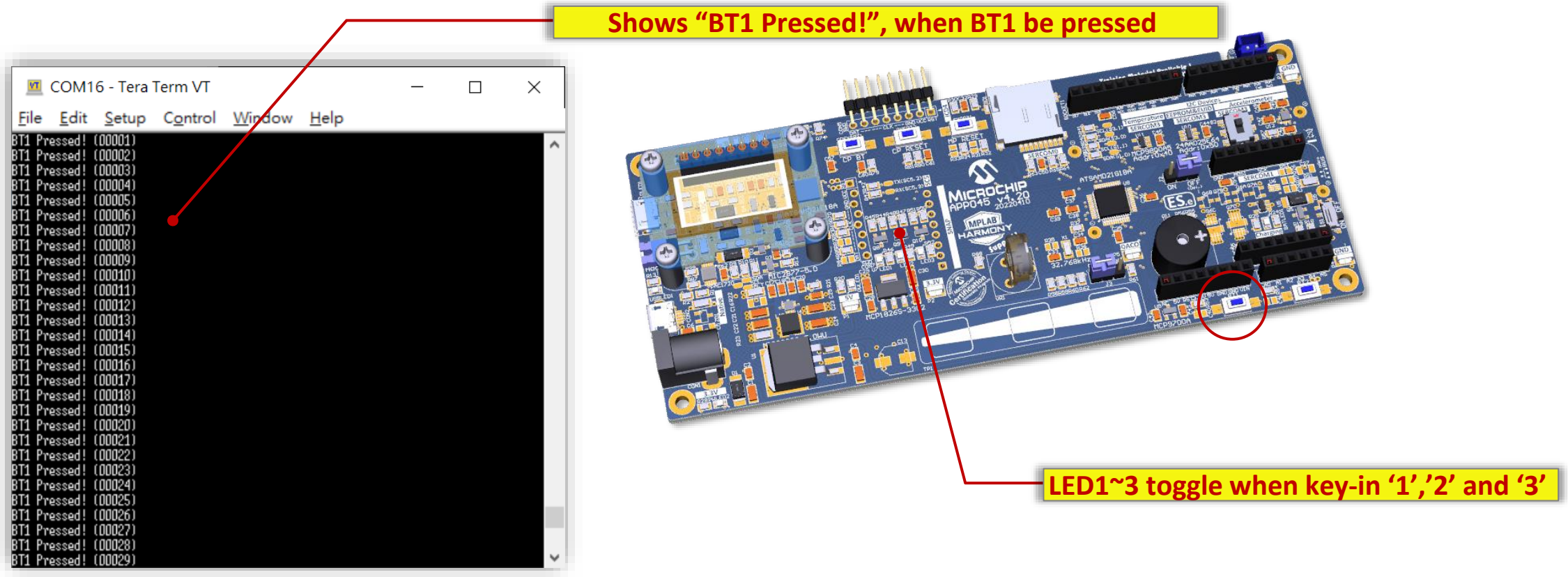
🛠 Lab5 USB Device CDC Class - Basic TX RX

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



🛠 Lab5 USB Device CDC Class - Basic TX RX

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



MICROCHIP

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
   ...
}
```

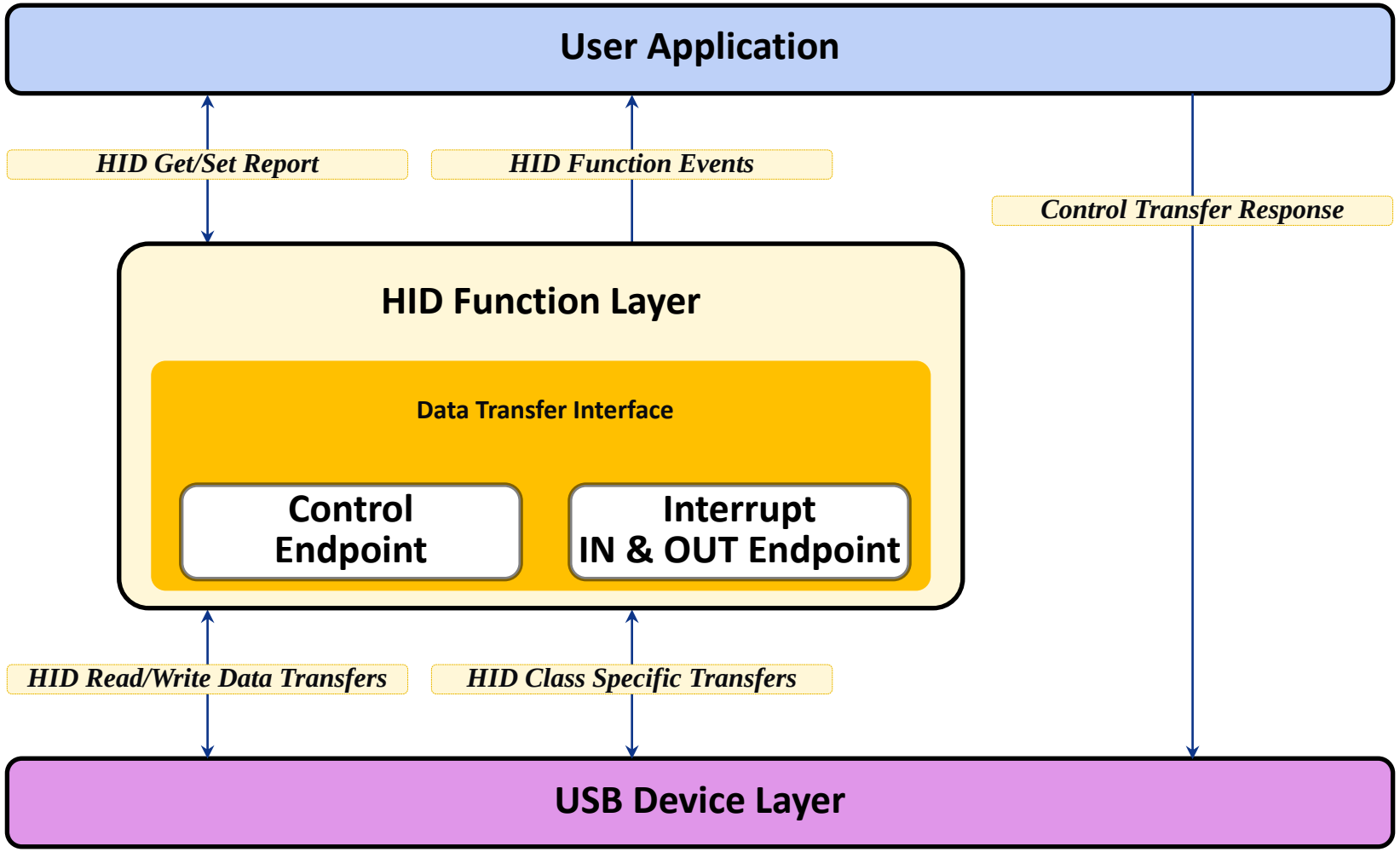
HID Usage Tables

- **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

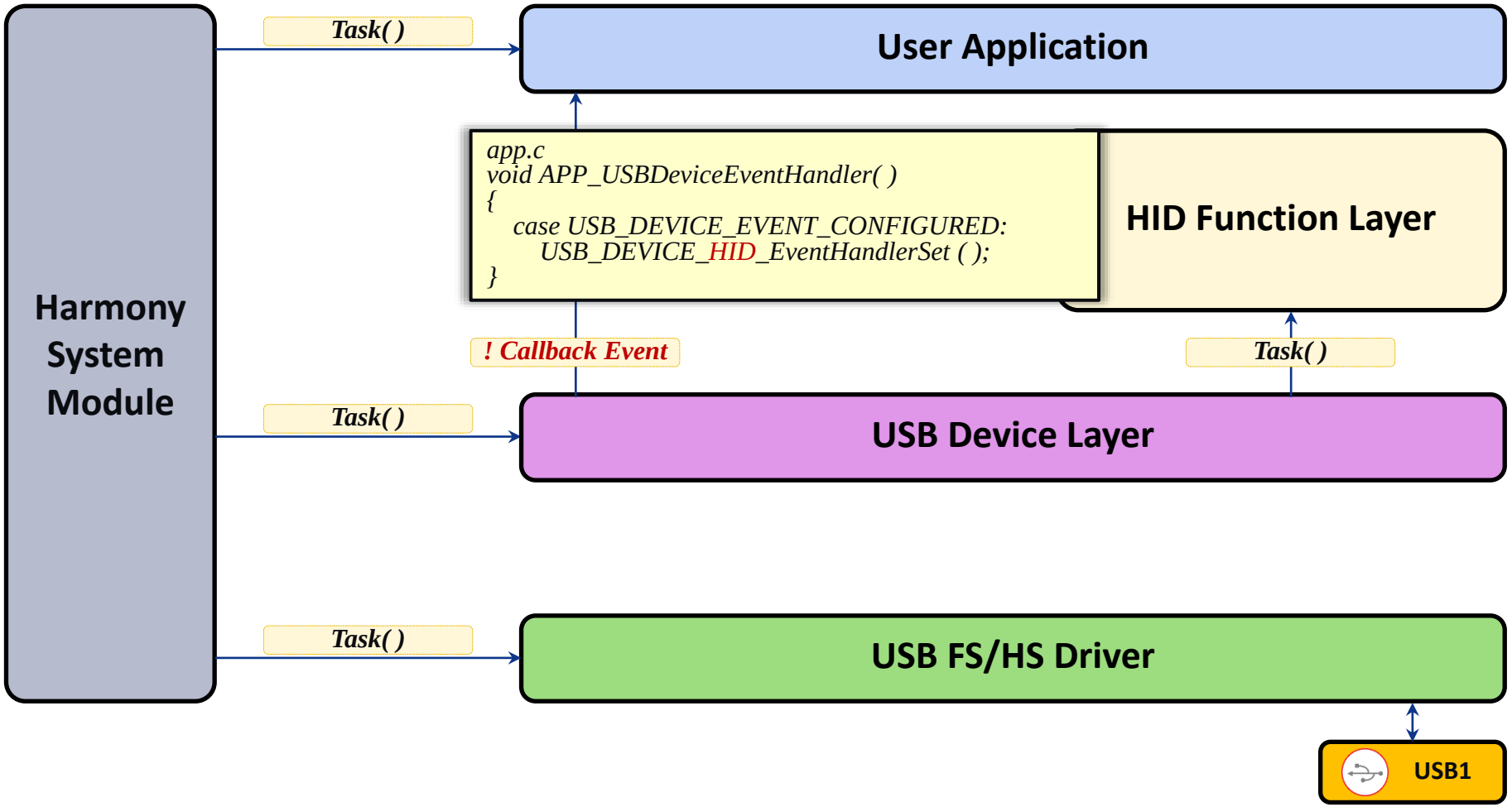
```
usb_device_init_data.c
const uint8_t hid_rpt0[] =
{
    0x06, 0x00, 0xFF,  // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,        // Usage (Vendor Usage 1)
    0xA1, 0x01, // Colsslection (Application)
    0x19, 0x01, // Usage Minimum
    0x29, 0x40, // Usage Maximum    //64 input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

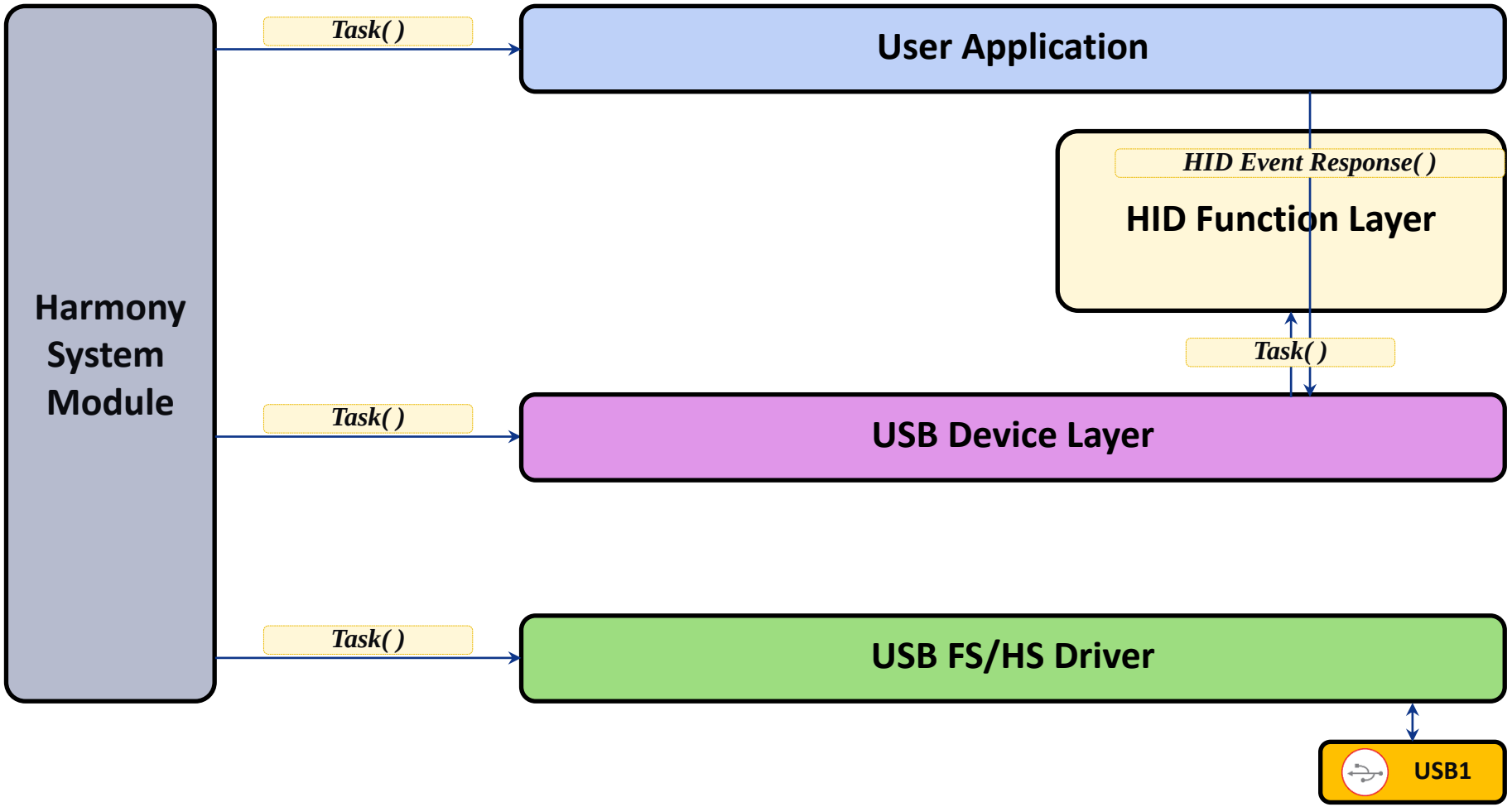


USB Stack Abstraction Model - Register Function Event Handler

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

MICROCHIP

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
);
```

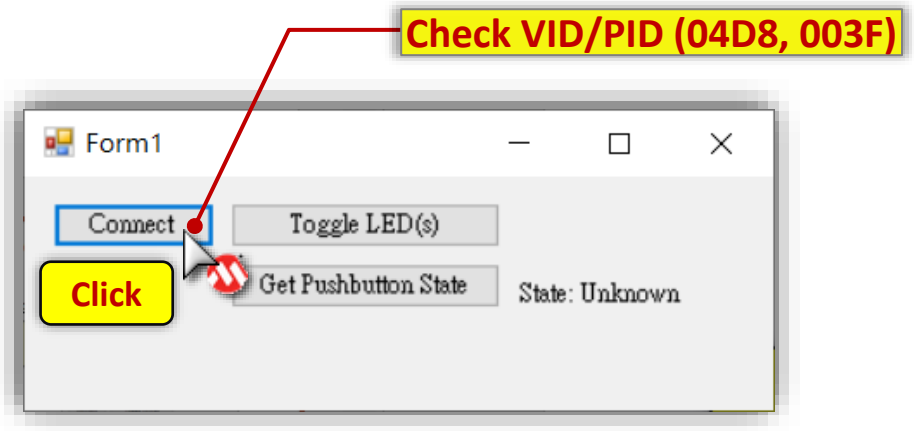
🛠 Lab6 USB Device HID Class - Basic TX and RX

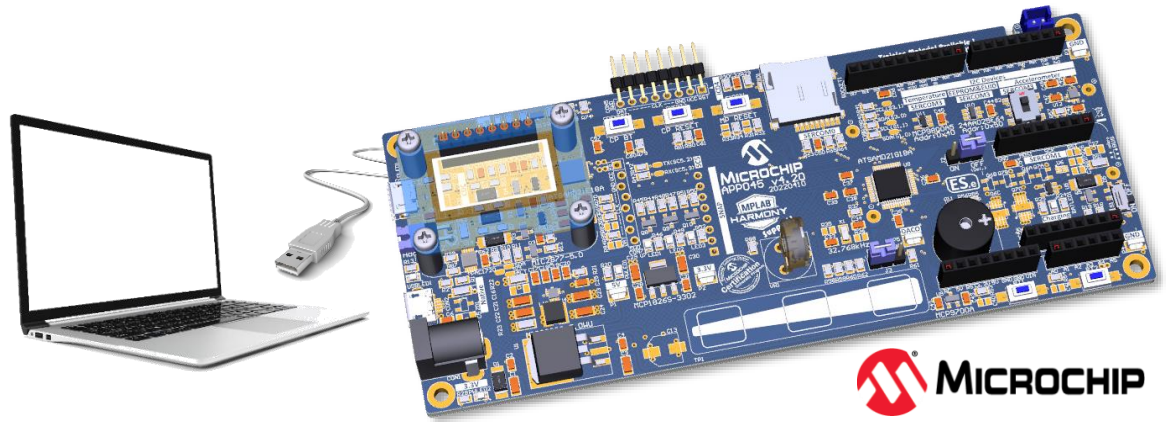
🛠 Lab6 USB Device HID Class - Basic TX and RX

- 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.

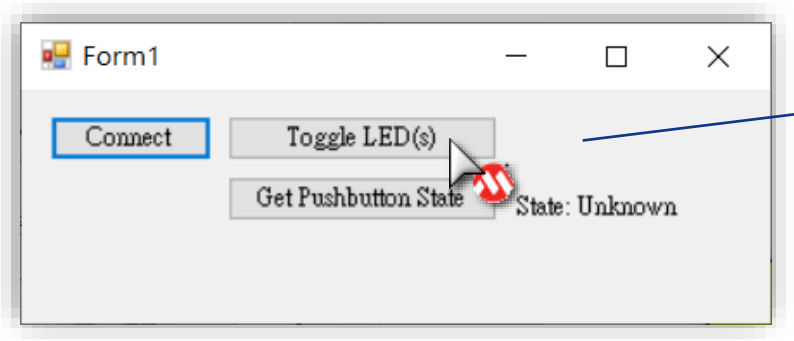
- **Let’s go !**

PC Application - Generic HID Simple Demo.exe



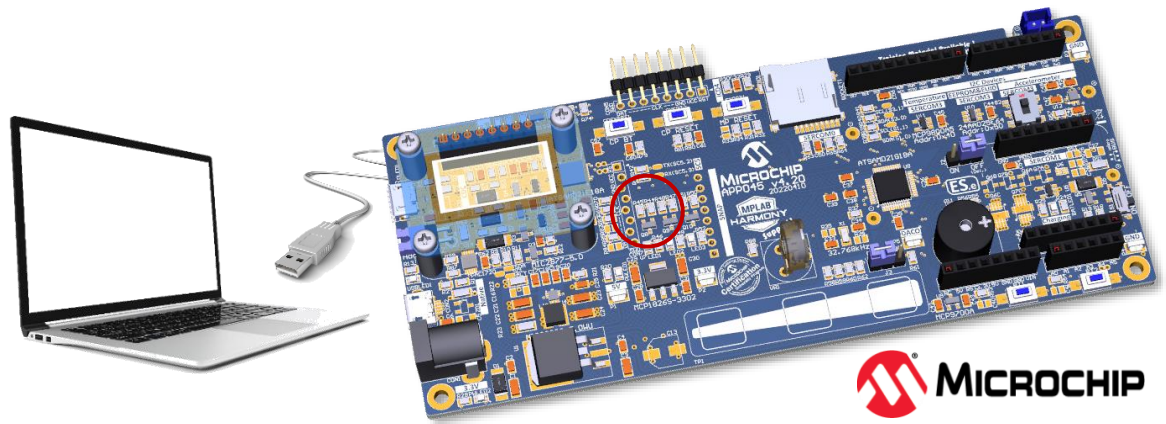


PC Application - Generic HID Simple Demo.exe

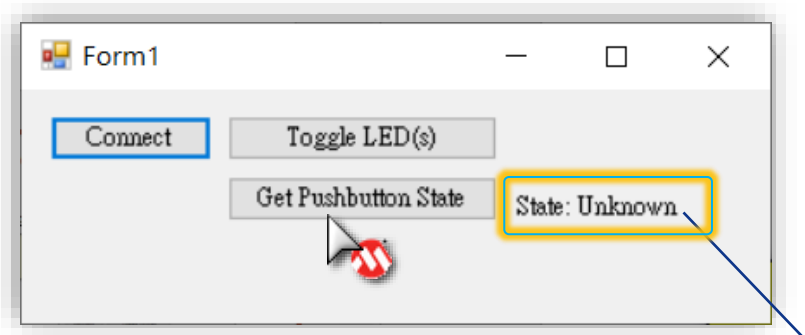


Toggle LED, (OUT Report)

0x80	0x00	...	0x00



PC Application - Generic HID Simple Demo.exe

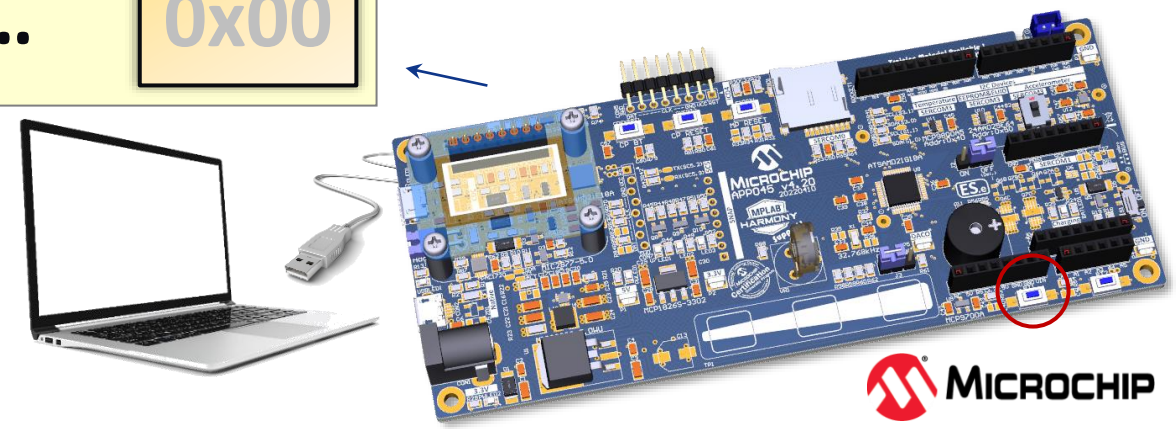


Get Push Button State (OUT Report)

0x81	0x00	...	0x00

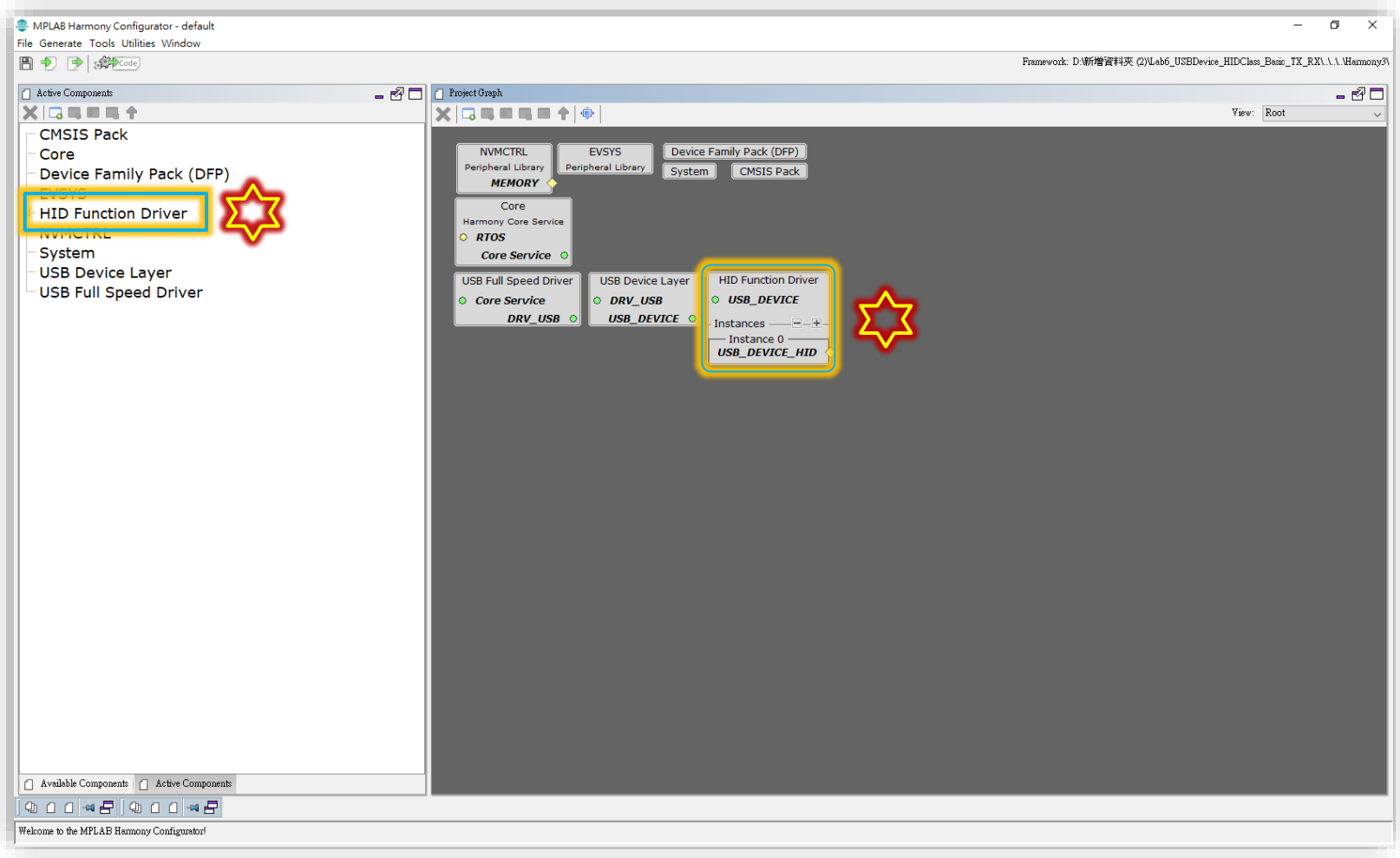
(IN Report)

0x81	BT	...	0x00



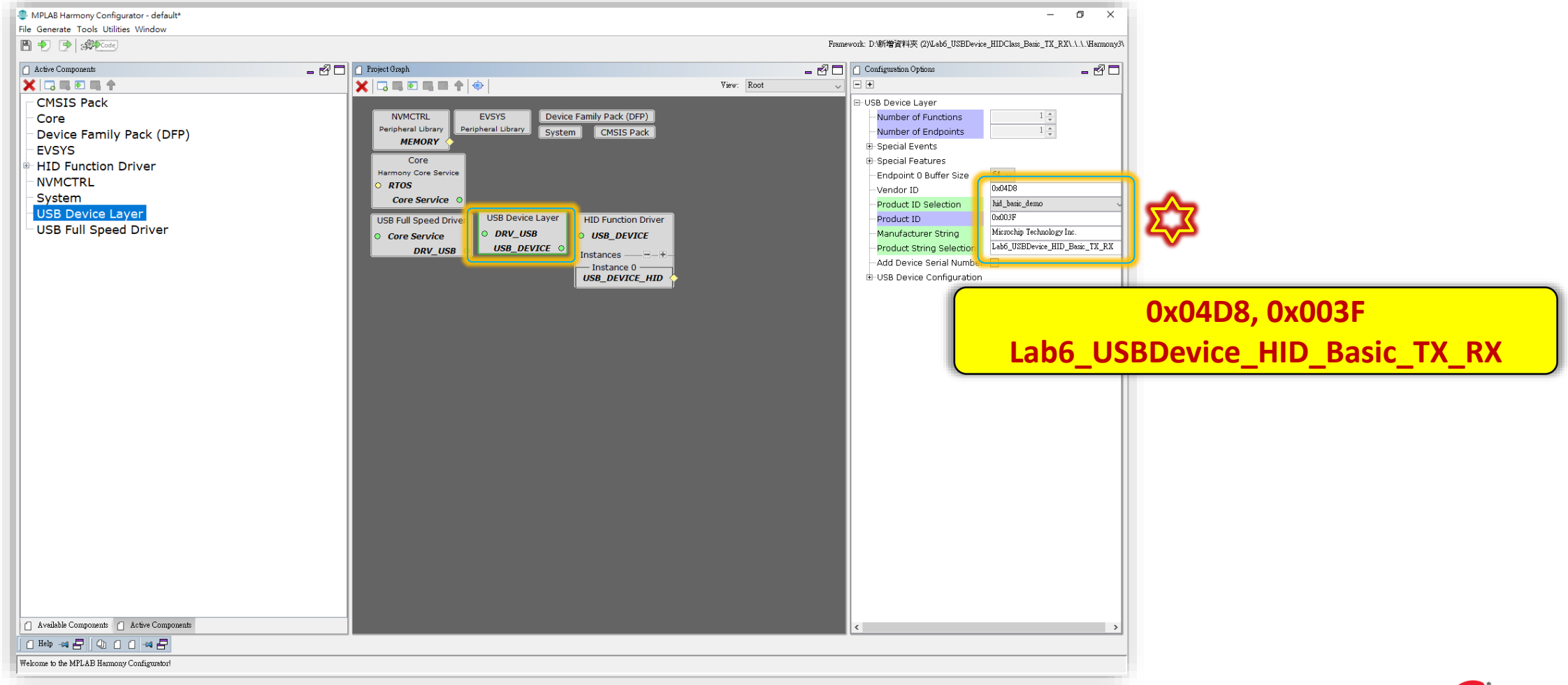
🛠 Lab6 USB Device HID Class - Basic TX and RX

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



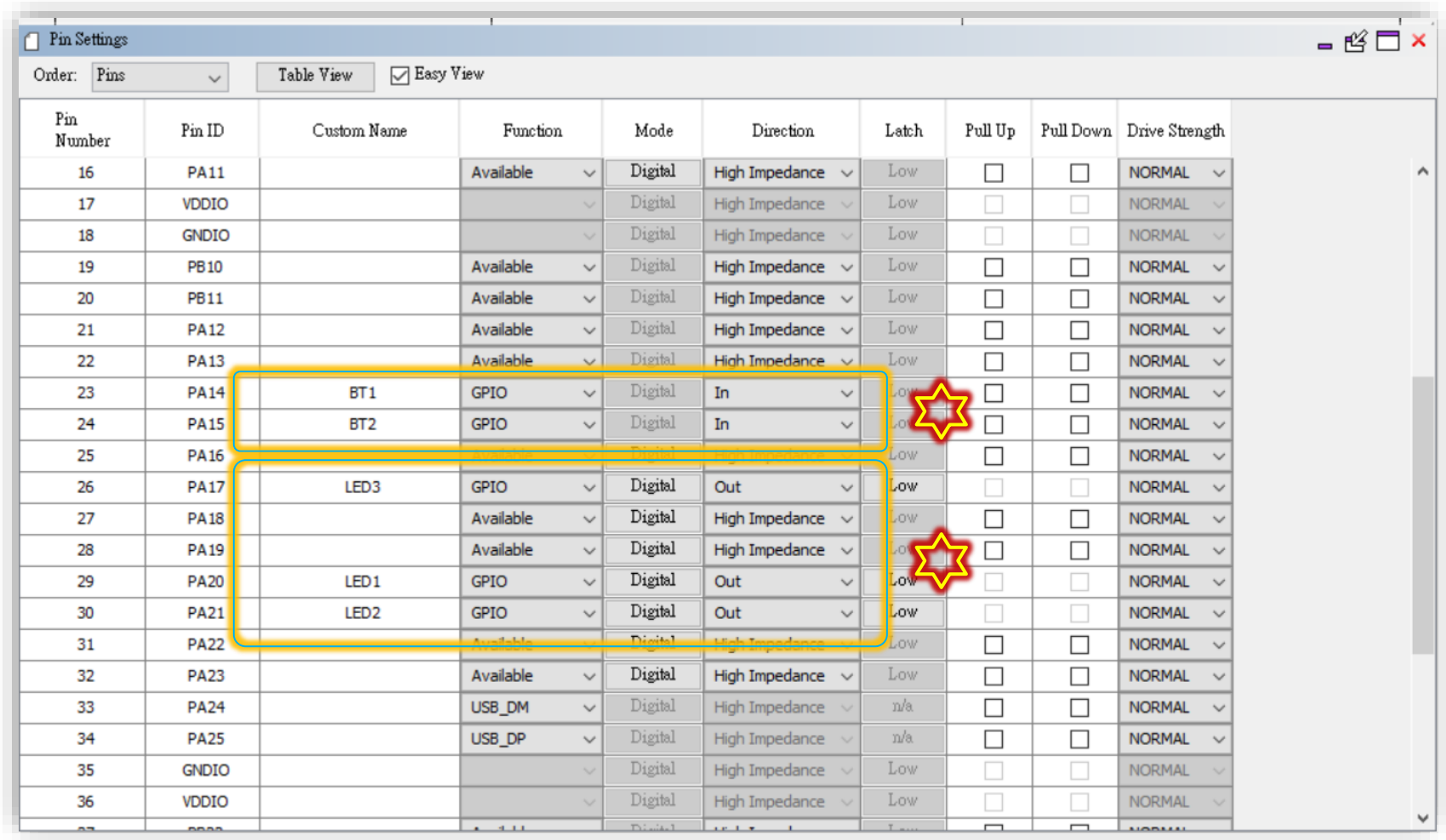
🛠 Lab6 USB Device HID Class - Basic TX and RX

- **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**

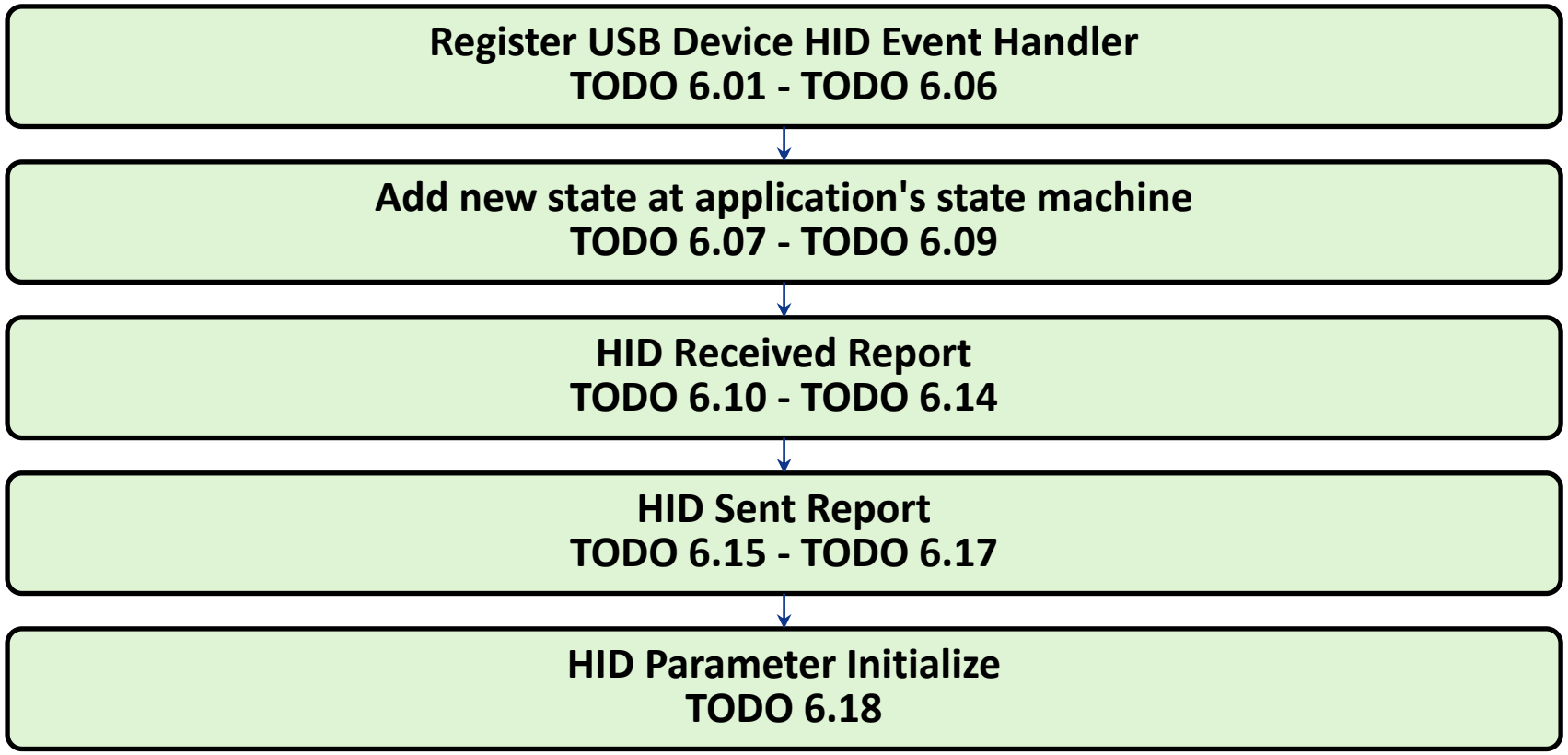


Pin Number	Pin ID	Custom Name	Function	Mode	Direction	Latch	Pull Up	Pull Down	Drive Strength
16	PA11		Available	Digital	High Impedance	Low	☐	☐	NORMAL
17	VDDIO			Digital	High Impedance	Low	☐	☐	NORMAL
18	GNDIO			Digital	High Impedance	Low	☐	☐	NORMAL
19	PB10		Available	Digital	High Impedance	Low	☐	☐	NORMAL
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					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					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

MICROCHIP

🛠 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;
}
```

MICROCHIP

🛠 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;
    }
    }
}
```

MICROCHIP

🛠 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;
```

```
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_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;
    }
    }
}
```

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

MICROCHIP

🛠 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;
```

```
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;
}
```

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

MICROCHIP

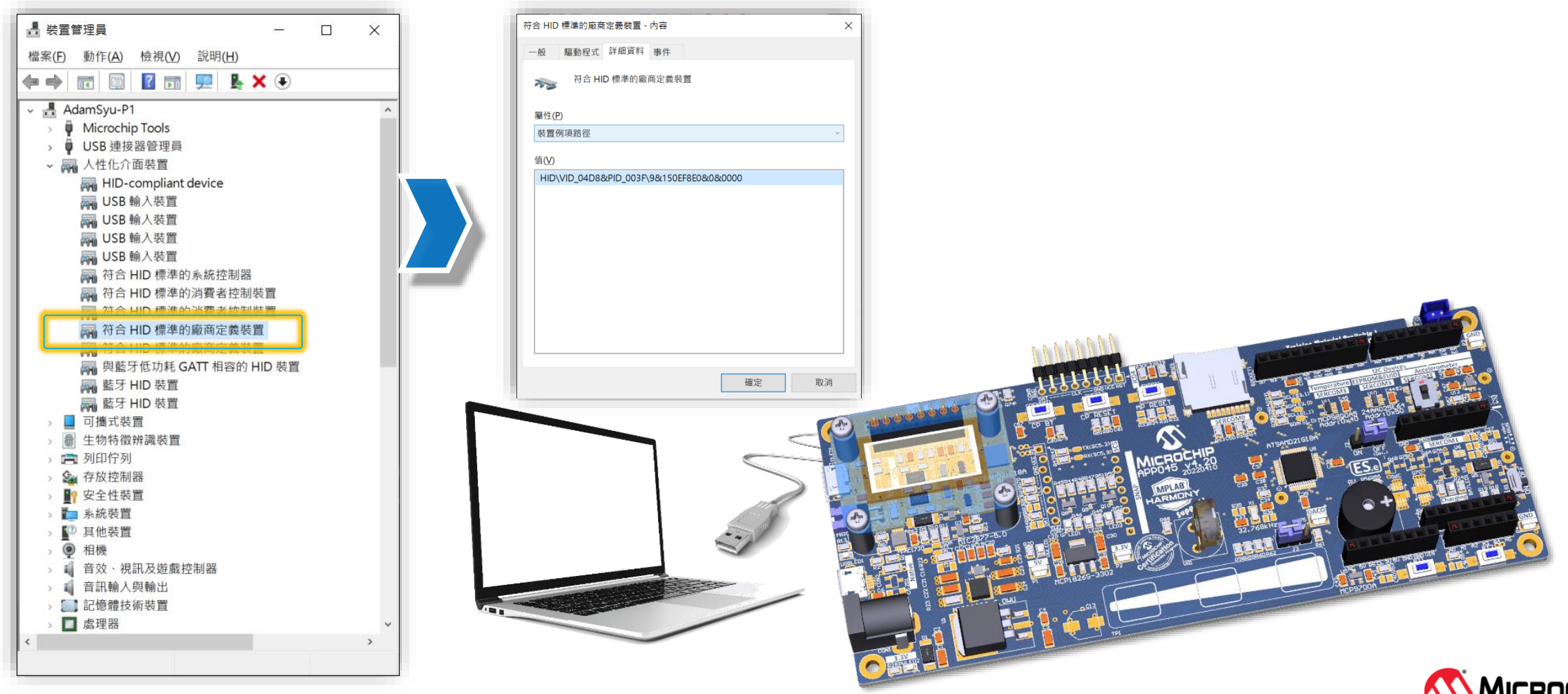
🛠 Lab6 USB Device HID Class - Basic TX and RX

- **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;
}
```

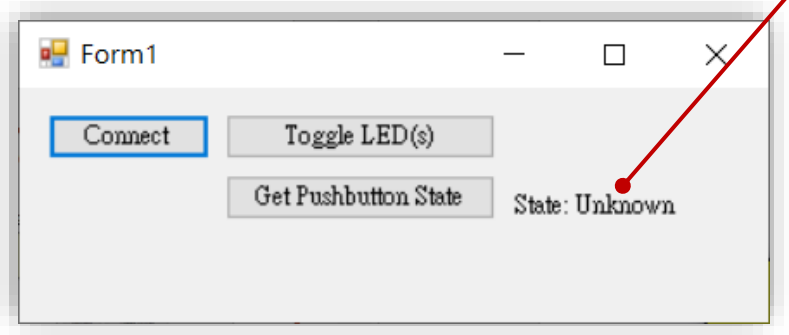
🛠 Lab6 USB Device HID Class - Basic TX and RX

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

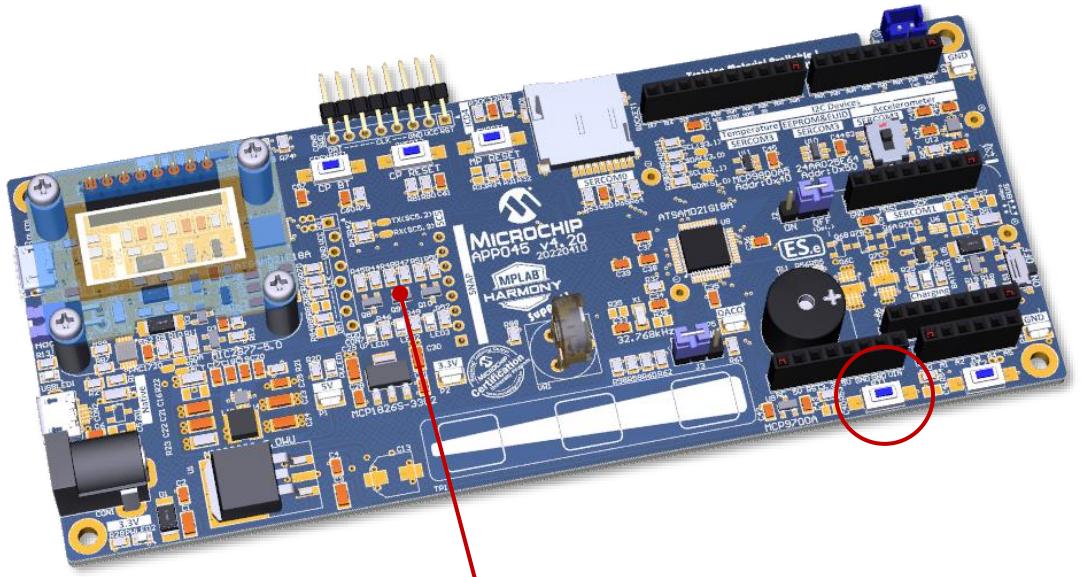


🛠 Lab6 USB Device HID Class - Basic TX and RX

- 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, SQI, 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