life.augmented

STM32 AZURE RTOS workshop

USBX

Introduction

- Rich class offer
- Native support for chaining multiple classes in one device – either host or device
  - Composite device and universal host capable communicate with composite or various classes
- Robustness, Certification
- ThreadX like memory allocation options
- Native connection to ThreadX and other Azure RTOS components
  - Consistent API

USBX device classes

USB device classes	
AUDIO	Version 2.0 supported
CDC/ACM	Communication Data Class – Abstract Control Model (virtual COM port)
CDC/ECM	Communication Data Class – Ethernet Control Module
DFU	Device Firmware Update
HID	Human Interface Device
PIMA (PTP/MTP)	Picture (media) Transfer Protocol ★
Storage	Mass Storage class
RNDIS	Remote Network Driver Interface Specification

Legend: ★ New class in STM32 middleware offer

USBX host classes

USBX host classes	
ASIX	USB/ethernet adaptor with proprietary vendor commands ★
Audio	Version 2.0 supported
CDC/ACM	Communication Data Class – Abstract Control Model (virtual COM port)
CDC/ECM	Communication Data Class – Ethernet Control Module ★
GSER	Gadget serial ★
HID	Human Interface Device
HUB	USB hub class ★
PIMA (PTP/MTP)	Picture (media) Transfer Protocol
Printer	USB printer class ★
Prolific	USB/serial convertors ★
Storage	Mass Storage class

Legend: ★ New class in STM32 middleware offer

HandsOn: Create USB composite device – HID + CDC ACM (VCP)

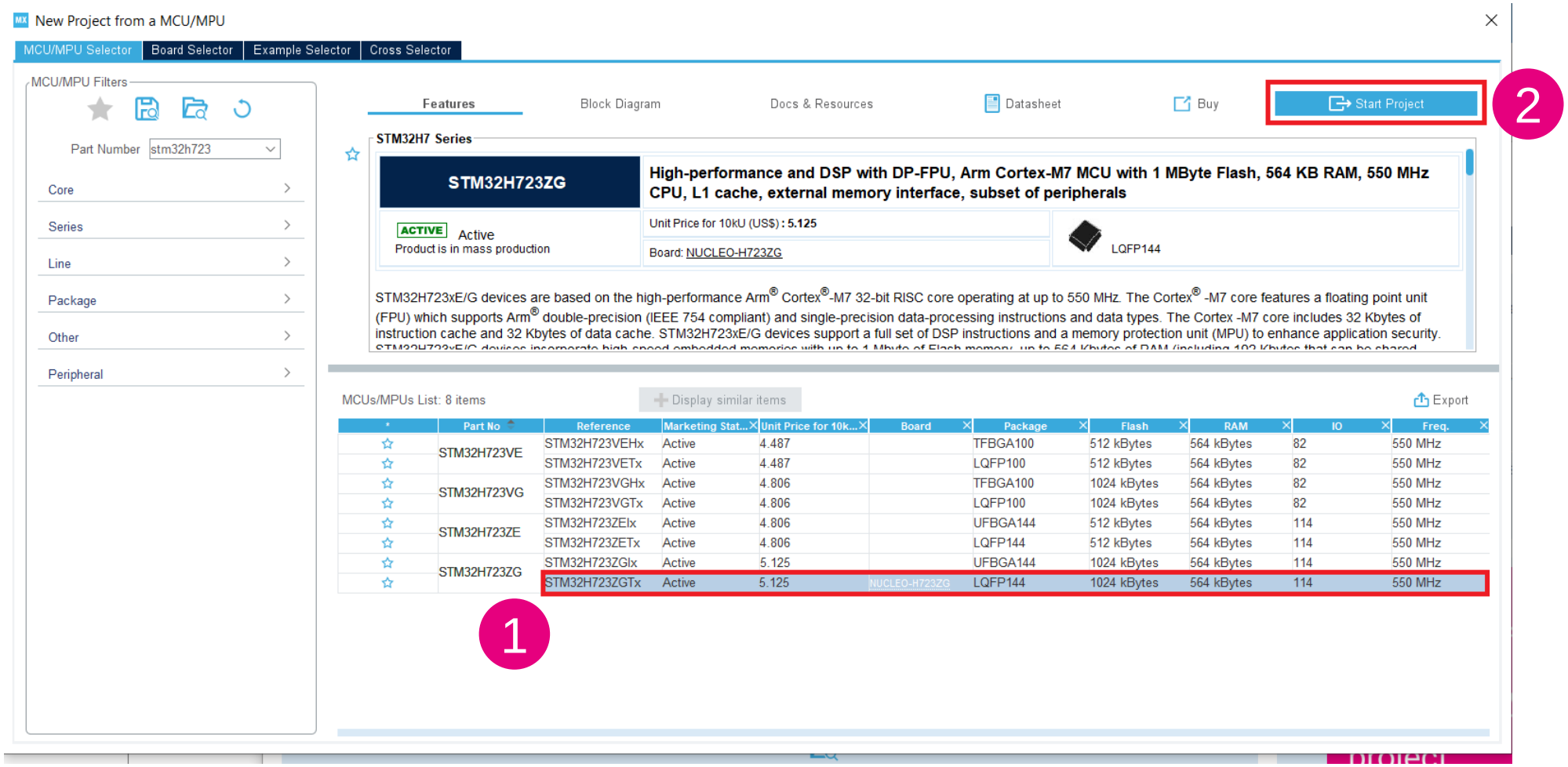
life.augmented

Example description

- Create USB composite device configuration with HID and CDC-ACM interface
  - HID periodically moves mouse cursor
  - CDC virtual COM port echoes incoming data packets
- STM32CubeMX creates project with basic configuration
- Incomplete application functionality is copied from prepared template
  - Templates are based on repository example Ux_Device_HID_CDC_ACM
  - Easy migration of USB functionality between different STM32 MCUs

life.augmented

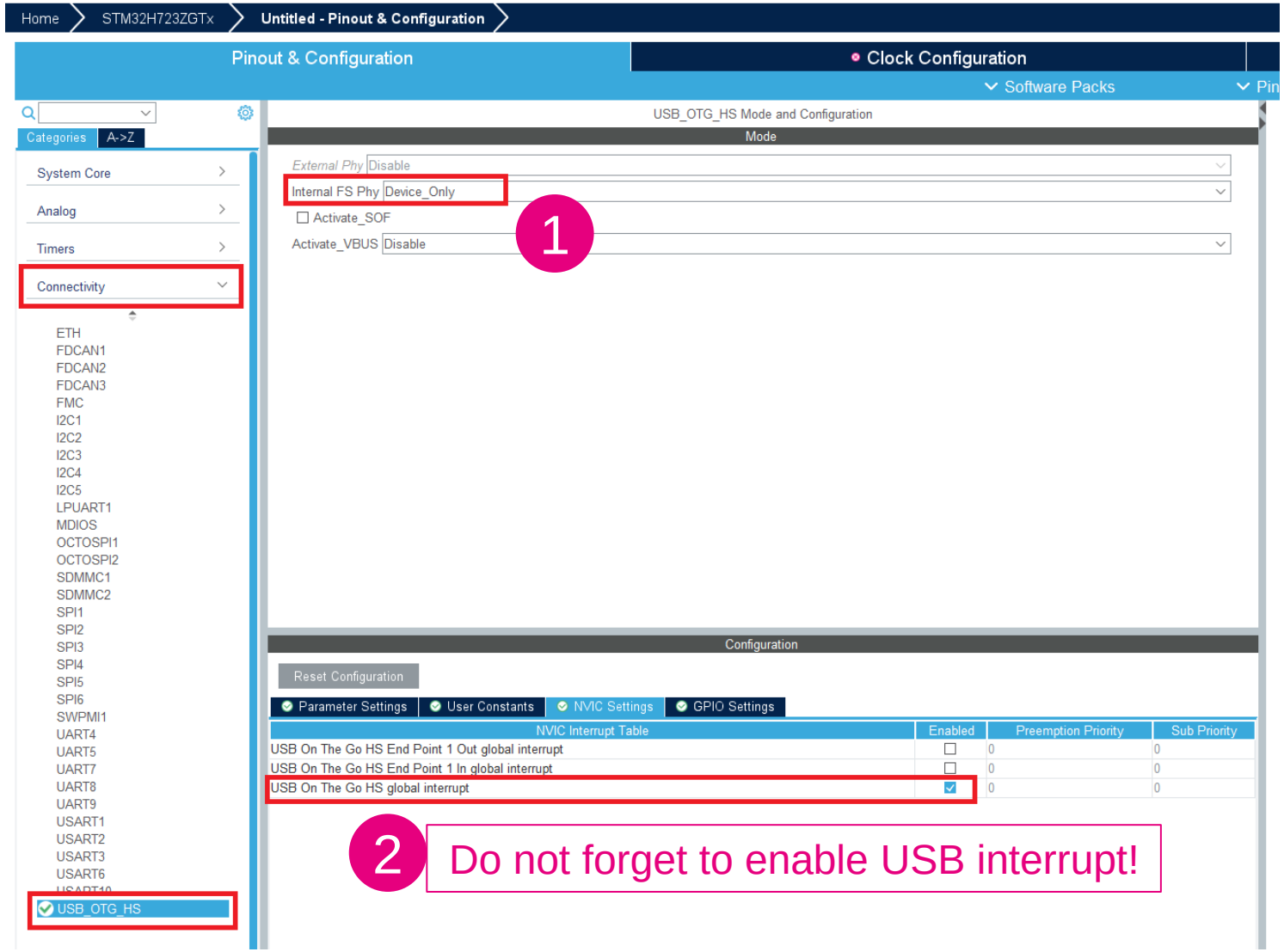
Start from STM32CubeMX – MCU Selector

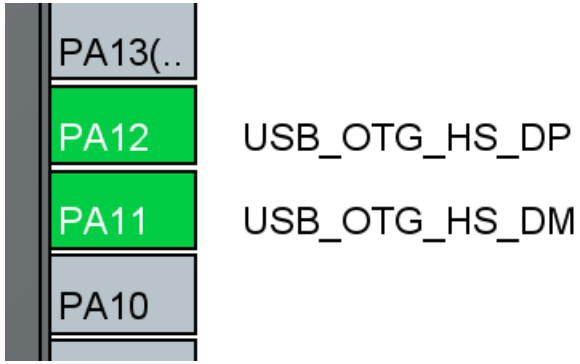


*	Part No	Reference	Marketing Stat...	Unit Price for 10k...	Board	Package	Flash	RAM	IO	Freq.
☆	STM32H723VE	STM32H723VEHx	Active	4.487		TFBGA100	512 kBytes	564 kBytes	82	550 MHz
☆		STM32H723VETx	Active	4.487		LQFP100	512 kBytes	564 kBytes	82	550 MHz
☆	STM32H723VG	STM32H723VGHx	Active	4.806		TFBGA100	1024 kBytes	564 kBytes	82	550 MHz
☆		STM32H723VGTx	Active	4.806		LQFP100	1024 kBytes	564 kBytes	82	550 MHz
☆	STM32H723ZE	STM32H723ZEIx	Active	4.806		UFBGA144	512 kBytes	564 kBytes	114	550 MHz
☆		STM32H723ZETx	Active	4.806		LQFP144	512 kBytes	564 kBytes	114	550 MHz
☆	STM32H723ZG	STM32H723ZGIx	Active	5.125		UFBGA144	1024 kBytes	564 kBytes	114	550 MHz
☆		STM32H723ZGTx	Active	5.125	NUCLEO-H723ZG	LQFP144	1024 kBytes	564 kBytes	114	550 MHz

life.augmented

Select USB device



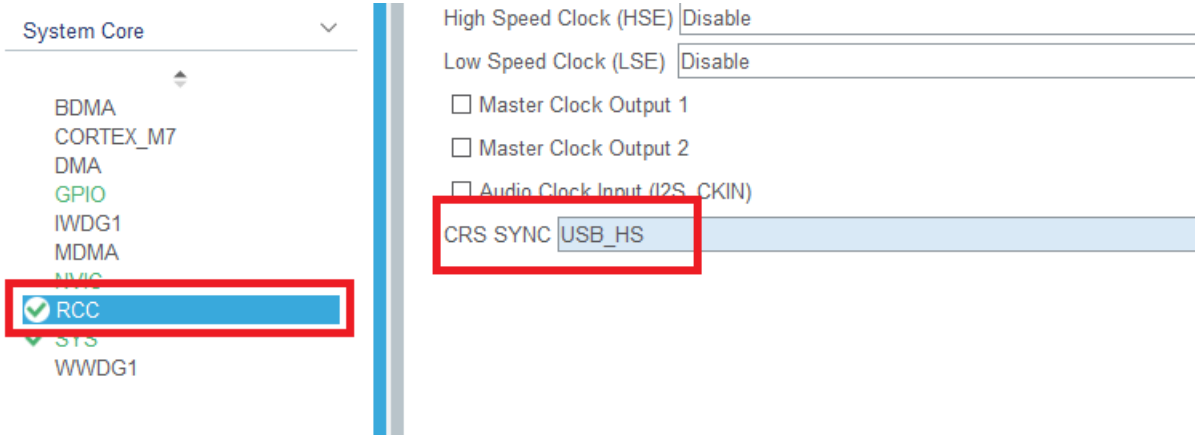


PA11 and PA12 are assigned to USB

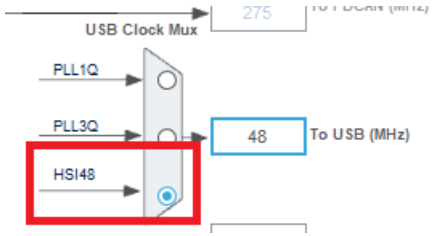
USB clock source

- Precise clock are mandatory for USB applications
  - STM32 USB host needs HSE
  - STM32 USB device can use HSE or advanced MCU features of some STM32 families
    - HSI48 + LSE (STM32L4)
    - HSI48 + CRS (multiple STM32 families including STM32H723), will be used in this example

1

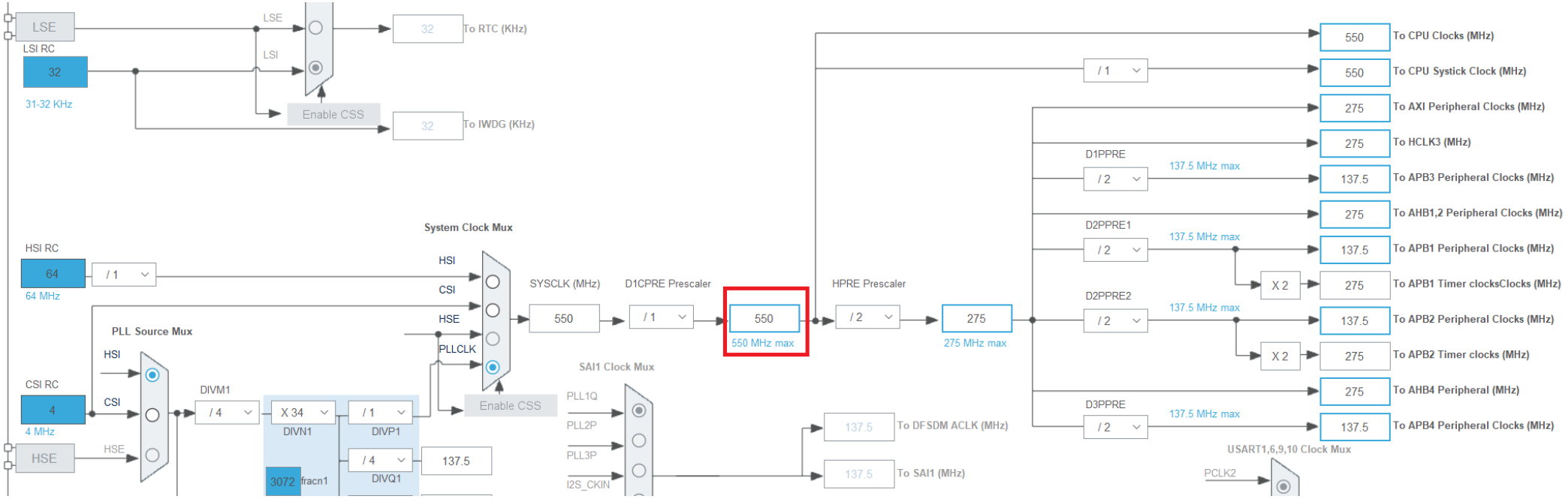


2 Clock Configuration panel

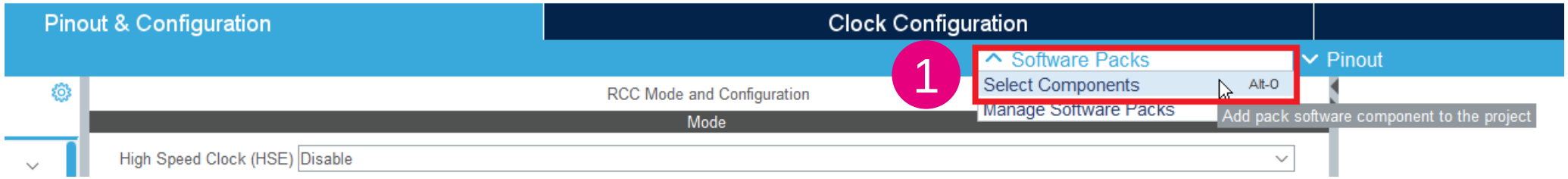


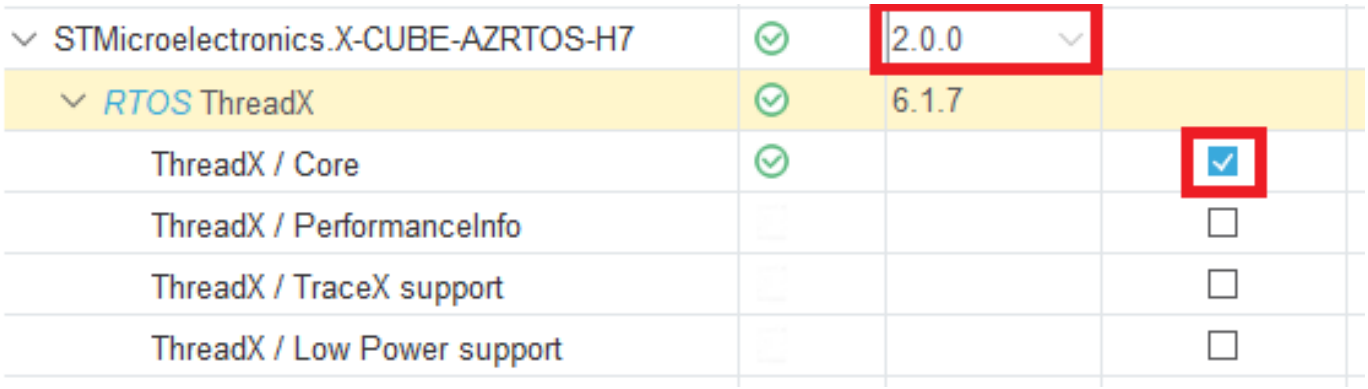
Setting system clock

- Using Clock configuration wizard set full MCU clock frequency – 550 MHz
  - Such high frequency is not mandatory, minimum HCLK frequency with OTG_xS used is 14.2 MHz



Add ThreadX


Pinout & Configuration
Clock Configuration
Software Packs
Select Components
Alt-O
Manage Software Packs
Pinout
1
RCC Mode and Configuration
Mode
Add pack software component to the project
High Speed Clock (HSE)
Disable

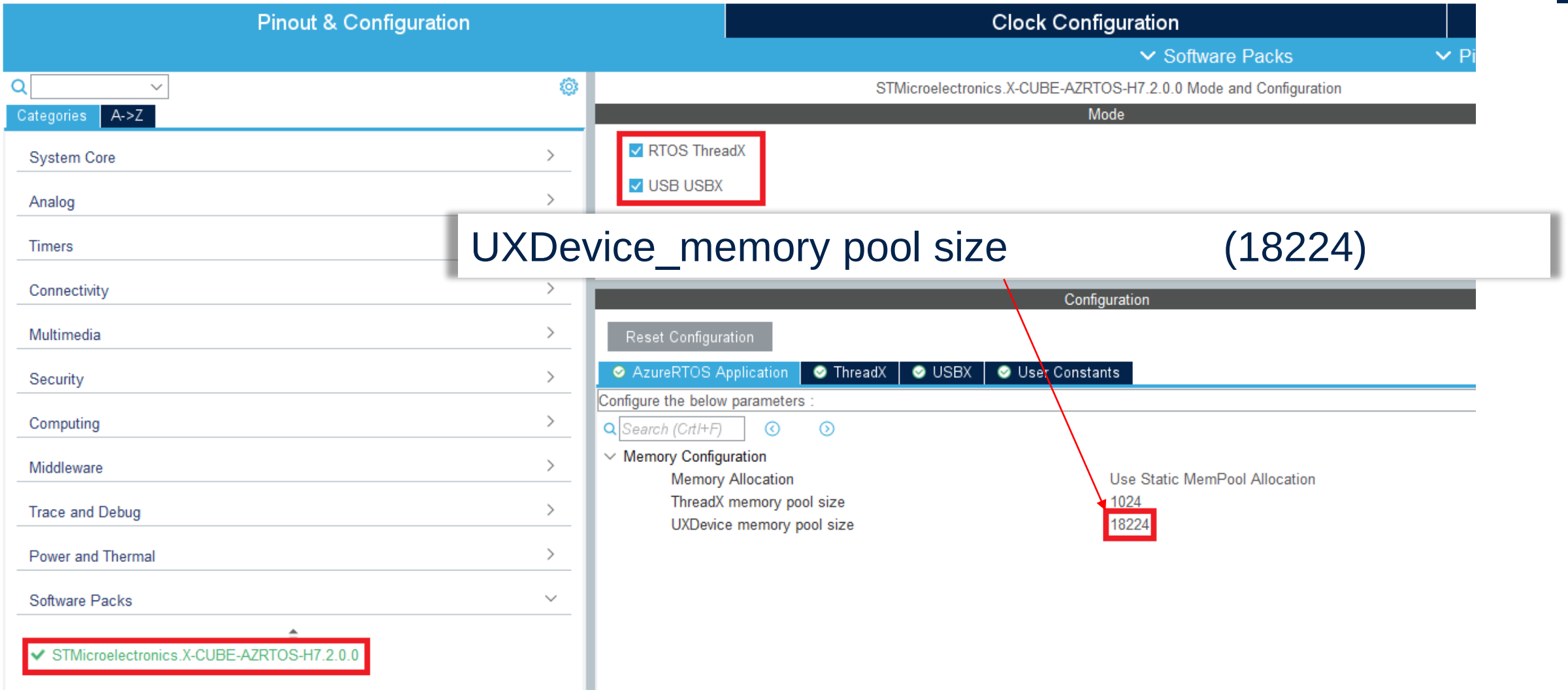

STMicroelectronics.X-CUBE-AZRTOS-H7
2.0.0
RTOS ThreadX
6.1.7
ThreadX / Core
ThreadX / PerformanceInfo
ThreadX / TraceX support
ThreadX / Low Power support
2

Add USBX

USB USBX	✓	6.1.7	
USBX / CoreSystem	✓		☑
USBX / TraceX Support			☐
USBX / UX Host CoreStack			☐
USBX / UX Host Controllers			☐
USBX / UX Device CoreStack	✓		☑
USBX / UX Device Controllers	✓		☑
USBX / UX Network Driver			☐
USBX / UX Host Class HID Core			☐
USBX / UX Host Class HID Keyboard			☐
USBX / UX Host Class HID Mouse			☐
USBX / UX Host Class HID RCU			☐
USBX / UX Host Class STORAGE			☐
USBX / UX Host Class CDC ACM			☐
USBX / UX Host Class CDC ECM			☐
USBX / UX Host Class AUDIO			☐
USBX / UX Host Class VIDEO			☐
USBX / UX Host Class PRINTER			☐
USBX / UX Host Class GSER			☐
USBX / UX Host Class ASIX			☐
USBX / UX Host Class PIMA			☐
USBX / UX Host Class PROLIFIC			☐
USBX / UX Host Class SWAR			☐
USBX / UX Device Class HID	✓		☑
USBX / UX Device Class STORAGE			☐
USBX / UX Device Class CDC ACM	✓		☑
USBX / UX Device Class CDC ECM			☐
USBX / UX Device Class DFU			☐
USBX / UX Device Class AUDIO			☐
USBX / UX Device Class PIMA			☐
USBX / UX Device Class RNDIS			☐

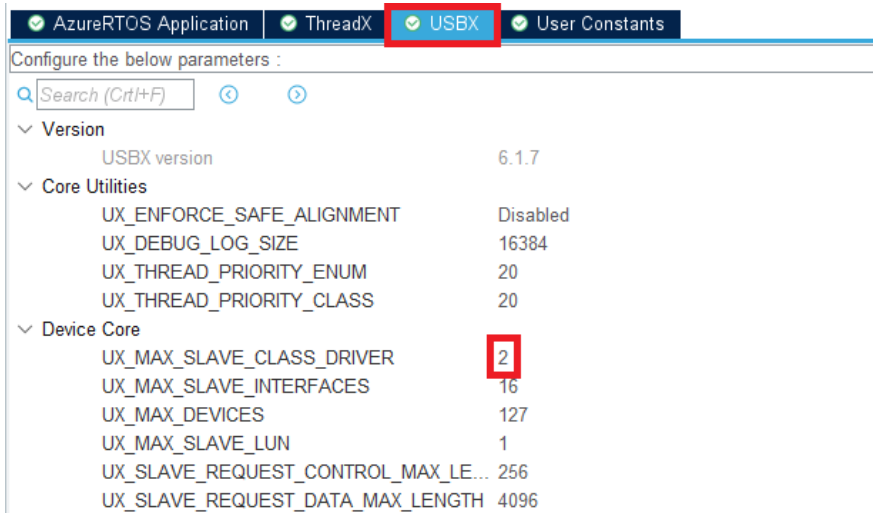
life.augmented

Configure AzureRTOS memory



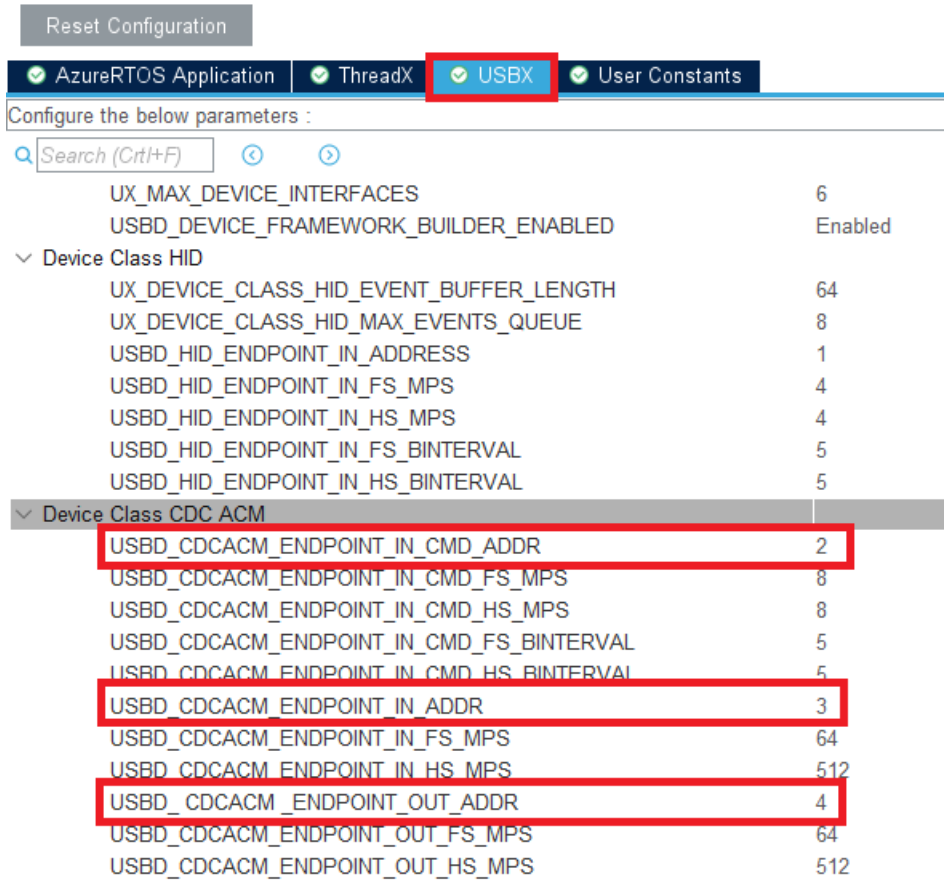
USBX setting 1

- Set UX_MAX_SLAVE_CLASS_DRIVER to 2
  - Two classes are used in composite device



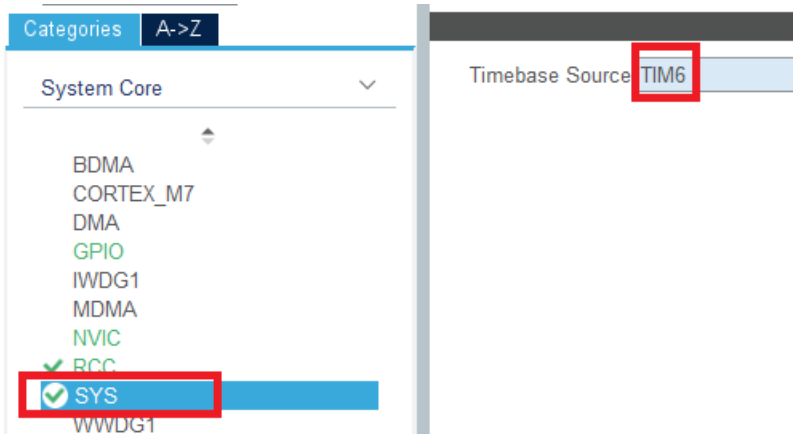
USBX setting 2

- Chose endpoint addresses
  - Different address need to be used for each endpoint.



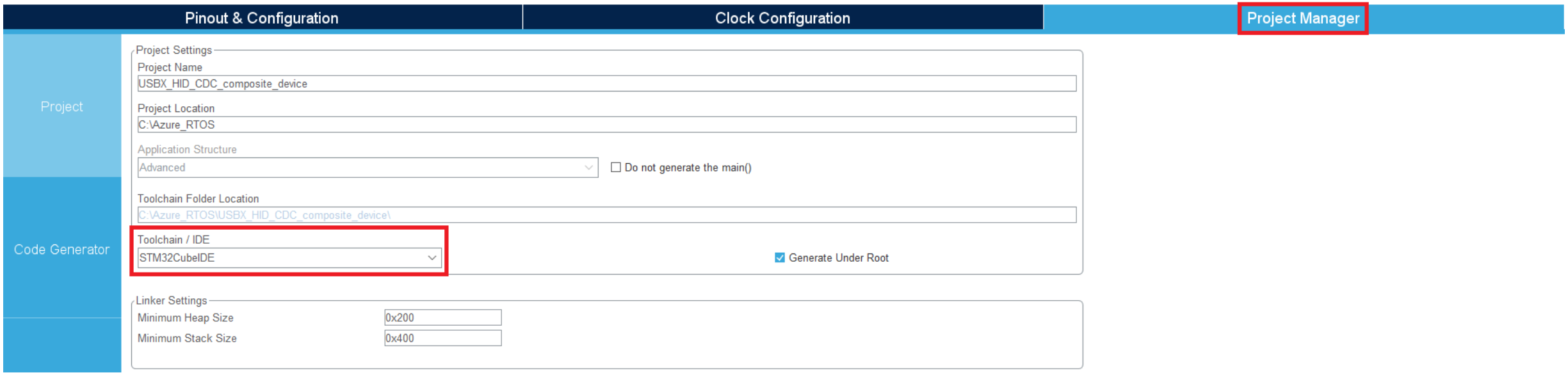
System Timebase Source

- SysTick used by Azure, use different source for HAL timing

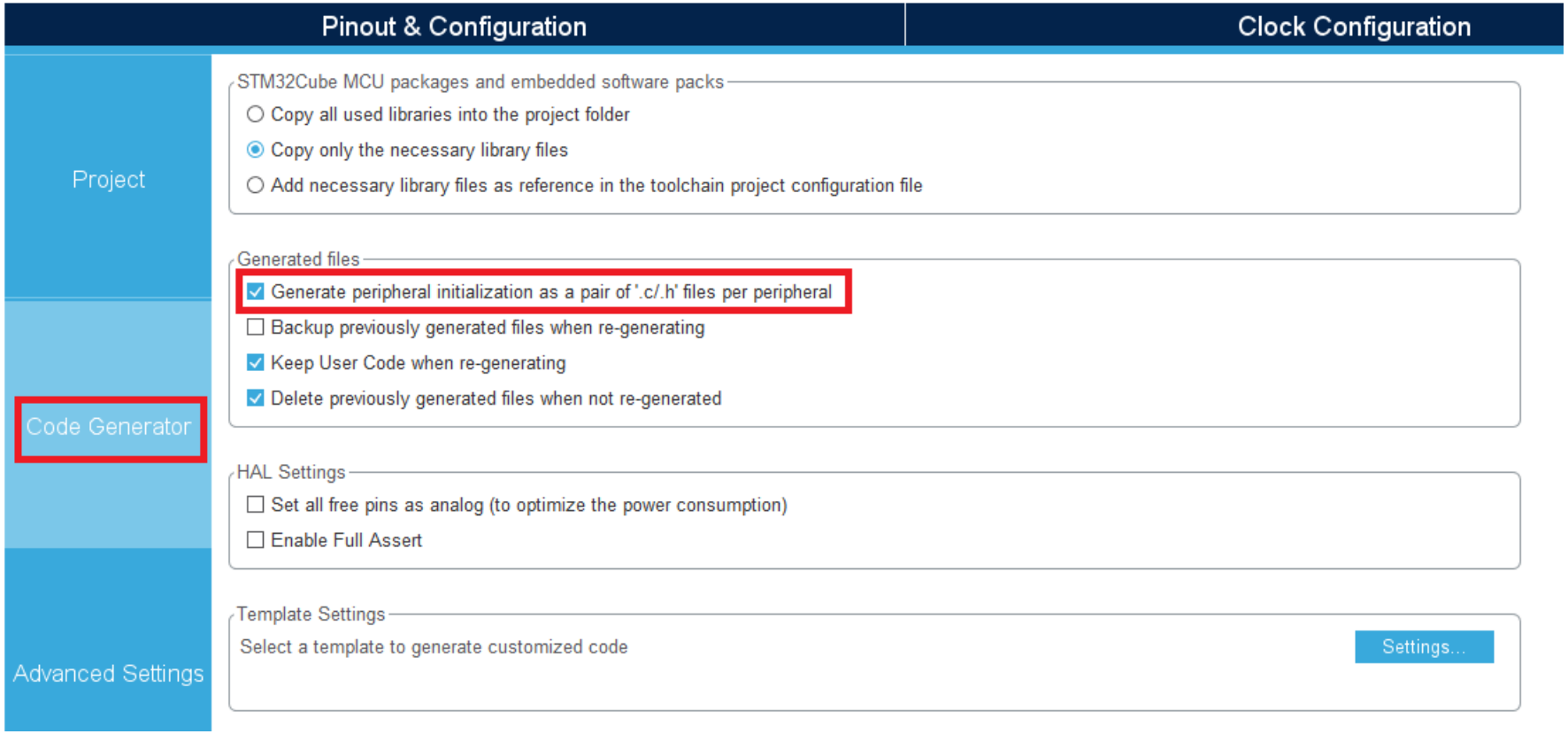


Project setting

- Set project name
- Used IDE – STM32CubeIDE



Code Generator options



Advanced settings

Pinout & Configuration | Clock Configuration | Project Manager

Project

Code Generator

Advanced Settings

Driver Selector

Search (Crtl+F)

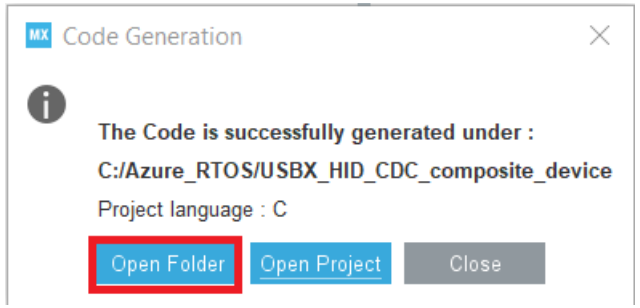
GPIO	HAL
RCC	HAL
USB_OTG_HS	HAL
CORTEX_M7	HAL

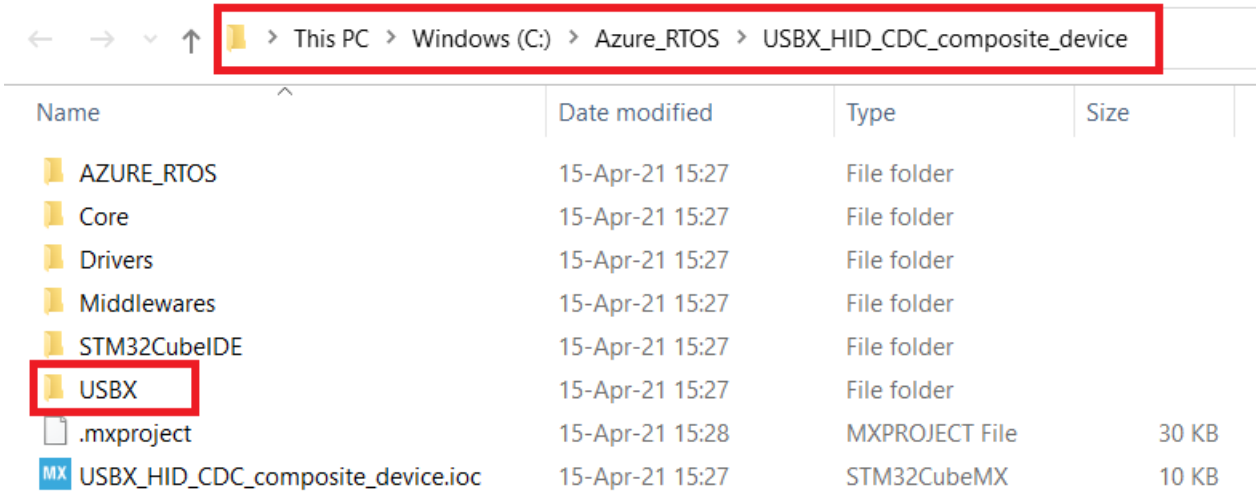
Generated Function Calls

Generate Code	Rank	Function Name	Peripheral Instance Name	Do Not Generate Function Call	Visibility (Static)
☑	1	MX_GPIO_Init	GPIO	☐	☑
☑	2	SystemClock_Config	RCC	☐	☐
☑	3	MX_USB_OTG_HS_PCD_Init	USB_OTG_HS	☑	☑

life.augmented

Code generation



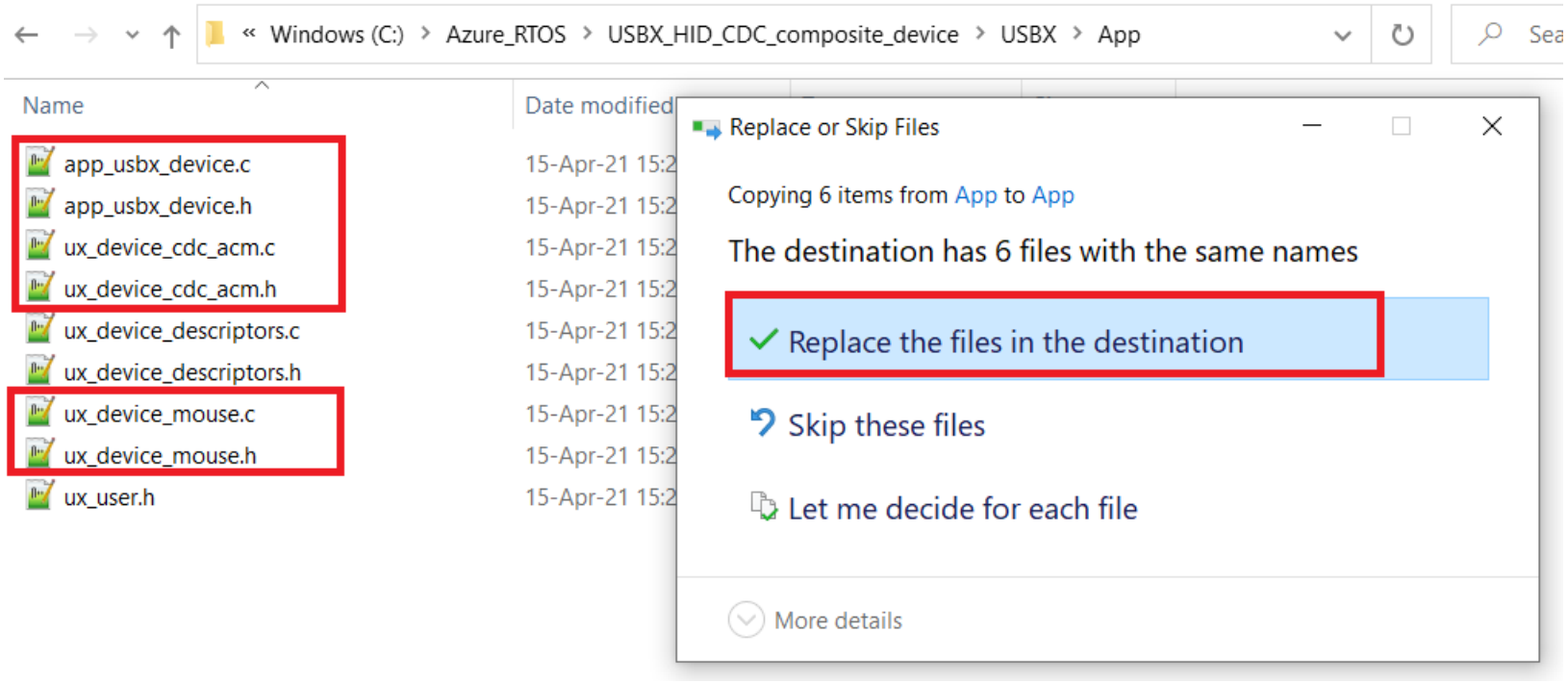


Replace empty templates

- Default path to files for copy

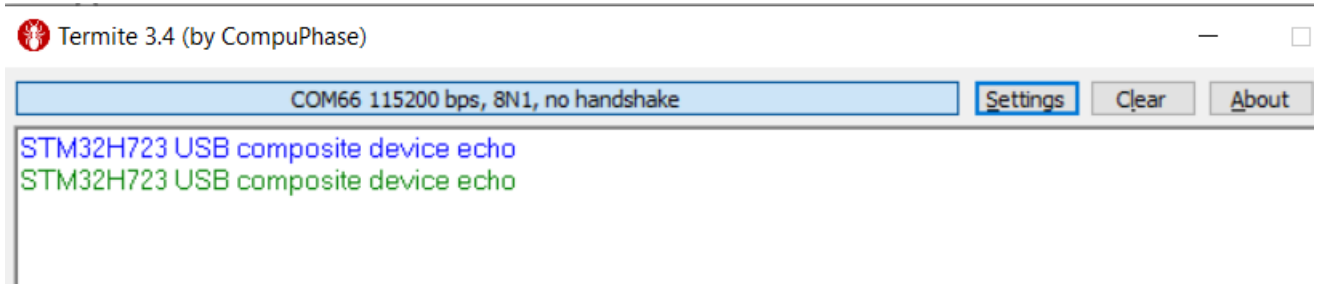
```
C:\tmp\STM32-AZURE-RTOS-WS-Material\HandsOn\USBX\USBX_copy_files
```

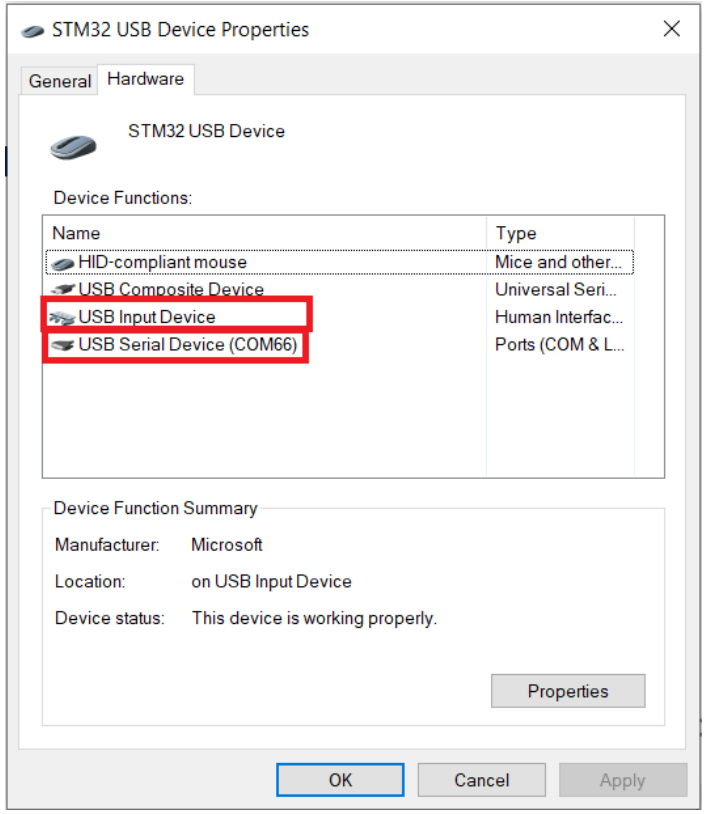
- Copy folder content into project folder opened in previous step



Project testing

- All set – run project on your STM32!
- Plug USB cable between PC and User USB port of nucleo board (CN13)
- Both HID and CDC interfaces are visible in one device for USB host
- HID class moves mouse cursor
- Virtual COM port echoes incoming messages





Template changes – app_usbx_device

- MX_USBX_Device_Init

1) Get frameworks (descriptors) created in ux_device_descriptor.c

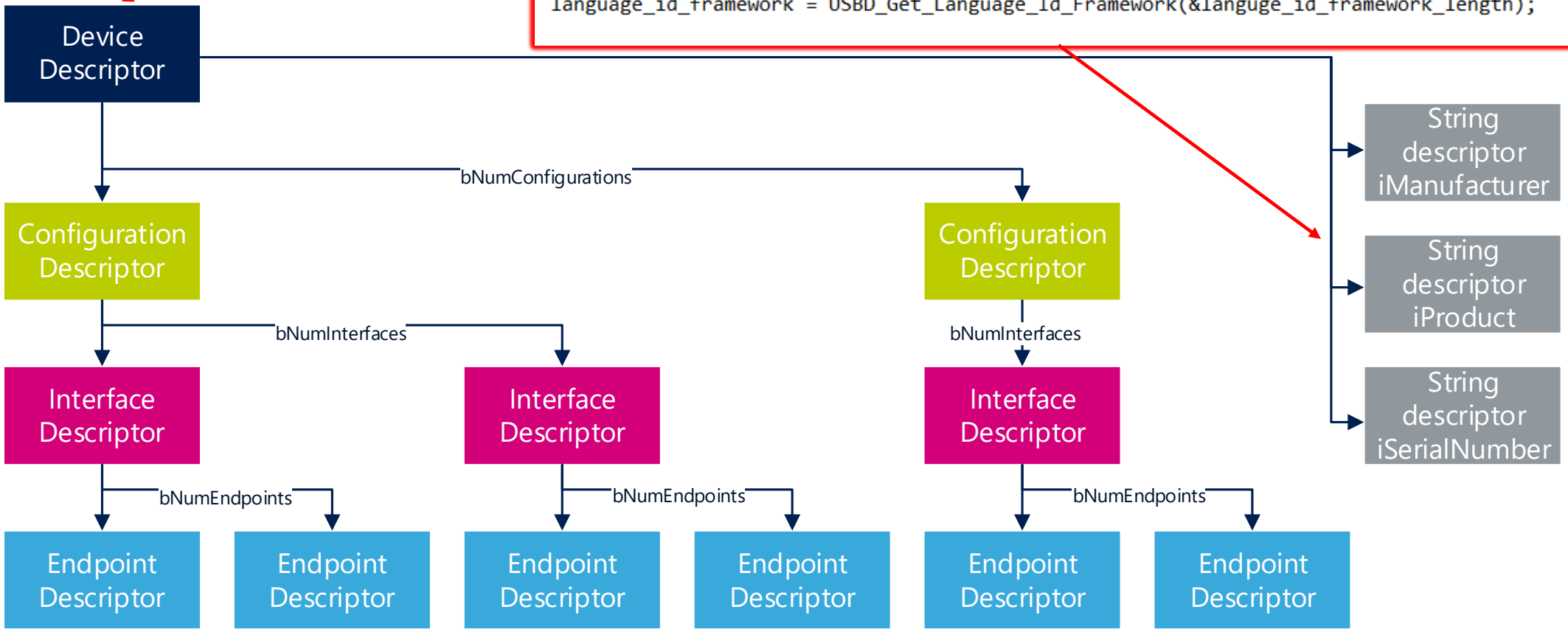
- Descriptor builder use parameters entered in STM32CubeMX

```
/* Get_Device_Framework_High_Speed and get the length */
  device_framework_high_speed = USBD_Get_Device_Framework_Speed(USBD_HIGH_SPEED,
                                &device_framework_hs_length);

  /* Get_Device_Framework_Full_Speed and get the length */
  device_framework_full_speed = USBD_Get_Device_Framework_Speed(USBD_FULL_SPEED,
                                &device_framework_fs_length);

  /* Get_String_Framework and get the length */
  string_framework = USBD_Get_String_Framework(&string_framework_length);

  /* Get_Language_Id_Framework and get the length */
  language_id_framework = USBD_Get_Language_Id_Framework(&languge_id_framework_length);
```

Device framework

```
/* Get_Device_Framework_High_Speed and get the length */
device_framework_high_speed = USBD_Get_Device_Framework_Speed(USBD_HIGH_SPEED,
                              &device_framework_hs_length);

/* Get_Device_Framework_Full_Speed and get the length */
device_framework_full_speed = USBD_Get_Device_Framework_Speed(USBD_FULL_SPEED,
                              &device_framework_fs_length);
```

```
/* Get_String_Framework and get the length */
string_framework = USBD_Get_String_Framework(&string_framework_length);

/* Get_Language_Id_Framework and get the length */
language_id_framework = USBD_Get_Language_Id_Framework(&languge_id_framework_length);
```



life.augmented

Template changes – app_usbx_device

- MX_USBX_Device_Init

2) Initialize USB device with obtained descriptors

- Does not contain any class dependent functionality yet

```
/* The code below is required for installing the device portion of USBX. In this application */
 ret =  ux_device_stack_initialize(device_framework_high_speed,
                                   device_framework_hs_length,
                                   device_framework_full_speed,
                                   device_framework_fs_length,
                                   string_framework,
                                   string_framework_length,
                                   language_id_framework,
                                   languge_id_framework_length, UX_NULL);
```

Template changes – app_usbx_device

- MX_USBX_Device_Init

3) Link HID class to USB device as interface 0

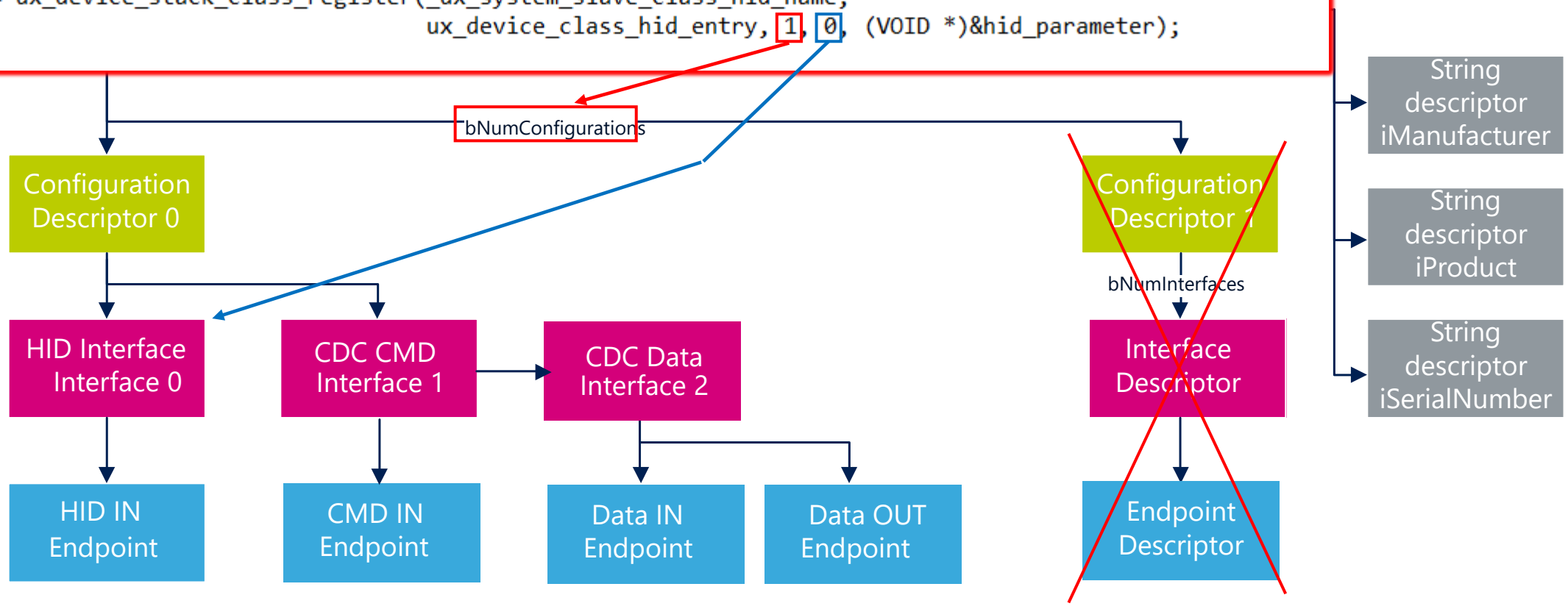
```
hid_parameter.ux_device_class_hid_parameter_report_address = USBD_Get_Device_HID_MOUSE_ReportDesc();

hid_parameter.ux_device_class_hid_parameter_report_length = USBD_HID_MOUSE_REPORT_DESC_SIZE;

hid_parameter.ux_device_class_hid_parameter_report_id = UX_TRUE;
hid_parameter.ux_device_class_hid_parameter_callback = app_usbx_device_thread_hid_callback;

/* Initialize the device hid class. The class is connected with interface 0 */
ret = ux_device_stack_class_register(_ux_system_slave_class_hid_name,
                                     ux_device_class_hid_entry, 1, 0, (VOID *)&hid_parameter);
```

Configuration number

Interface number

Class initialization

```
/* Initialize the hid class parameters for the device. */
hid_parameter.ux_device_class_hid_parameter_report_address = USBD_Get_Device_HID_MOUSE_ReportDesc();

hid_parameter.ux_device_class_hid_parameter_report_length = USBD_HID_MOUSE_REPORT_DESC_SIZE;

hid_parameter.ux_device_class_hid_parameter_report_id = UX_TRUE;
hid_parameter.ux_device_class_hid_parameter_callback = app_usbx_device_thread_hid_callback;

/* Initialize the device hid class. The class is connected with interface 0 */
ret = ux_device_stack_class_register(_ux_system_slave_class_hid_name,
                                     ux_device_class_hid_entry, 1, 0, (VOID *)&hid_parameter);
```



Template changes – app_usbx_device

- MX_USBX_Device_Init

4) Create one thread to maintain USB device and one thread for HID interface communication handling

```
ret = tx_byte_allocate(byte_pool, (VOID **) &pointer, USBX_APP_STACK_SIZE, TX_NO_WAIT);
ret = tx_thread_create(&ux_app_thread, "main_usbx_app_thread_entry", usbx_app_thread_entry, 0,
                    pointer, USBX_APP_STACK_SIZE, 20, 20, 1, TX_AUTO_START);

ret = tx_byte_allocate(byte_pool, (VOID **) &pointer, USBX_APP_STACK_SIZE, TX_NO_WAIT);
ret = tx_thread_create(&ux_hid_thread, "hid_usbx_app_thread_entry", usbx_hid_thread_entry, 1,
                    pointer, USBX_APP_STACK_SIZE, 20, 20, 1, TX_AUTO_START);
```

- Error check used in templates for each tx_* function are skipped in slides
- Steps 1-4 are needed for functionality of one class
  - Additionally step 7 is needed to init USB peripheral

life.augmented

Template changes – app_usbx_device

- MX_USBX_Device_Init
- For creating composite device next interface is added

5) Link CDC class to USB device as interface 1

```
cdc_acm_parameter.ux_slave_class_cdc_acm_instance_activate = CDC_Init_FS;
cdc_acm_parameter.ux_slave_class_cdc_acm_instance_deactivate = CDC_DeInit_FS;
cdc_acm_parameter.ux_slave_class_cdc_acm_parameter_change = ux_app_parameters_change;

 /* registers a slave class to the slave stack. The class is connected with interface 1 */

 ret = ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
                                      ux_device_class_cdc_acm_entry, 1, 1,
                                      (VOID *)&cdc_acm_parameter);
```

life.augmented

Template changes – app_usbx_device

- MX_USBX_Device_Init

6) CDC interface need dedicated thread for communication handling

- 1 thread is used for simplification in this demo, for regular application separate thread should be considered for Tx and Rx direction

```
ret = tx_thread_create(&ux_cdc_read_thread, "cdc_acm_read_usbx_app_thread_entry",
                                    usbx_cdc_acm_read_thread_entry, 2,
                                    pointer, USBX_APP_STACK_SIZE, 22, 22, TX_NO_TIME_SLICE,
                                    TX_AUTO_START);
```

life.augmented

Template changes – app_usbx_device

7) Usbx_app_thread_entry -This thread function initialize USB peripheral, run only once after reset

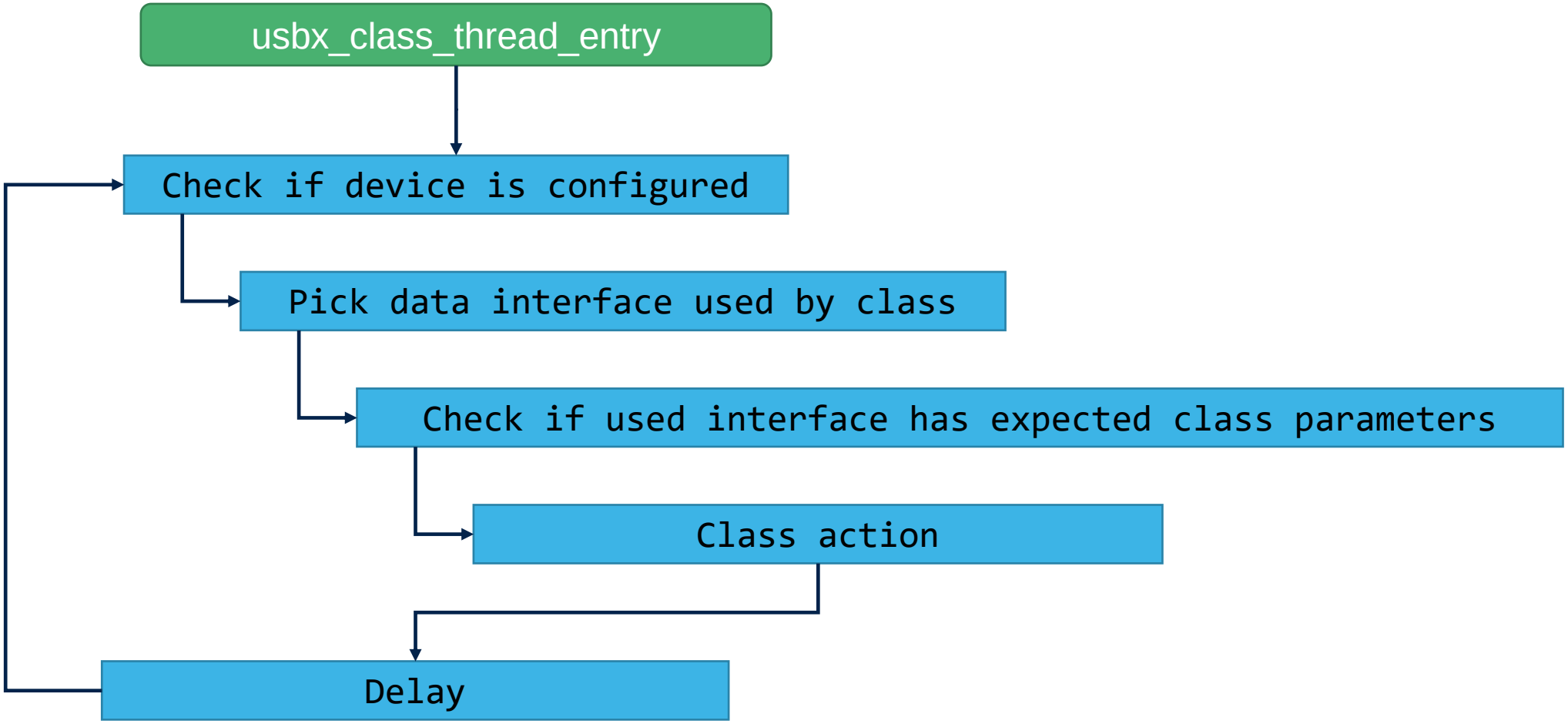
- Then USB events are handled in interrupt

```
void usbx_app_thread_entry(ULONG arg)
{
  /* Sleep for 100 ms */
  tx_thread_sleep(0.1 * TX_TIMER_TICKS_PER_SECOND);
  MX_USB_Device_Init();
}
```

```
void MX_USB_Device_Init(void)
{
  HAL_PWREx_EnableUSBVoltageDetector();

  MX_USB_OTG_HS_PCD_Init();
  HAL_PCDEx_SetRxFiFo(&hpcd_USB_OTG_HS, 0x200);
  HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_HS, 0, 0x40);
  HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_HS, 1, 0x40);
  HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_HS, 2, 0x40);
  HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_HS, 3, 0x40);

  _ux_dcd_stm32_initialize((ULONG)USB_OTG_HS, (ULONG)&hpcd_USB_OTG_HS);

  HAL_PCD_Start(&hpcd_USB_OTG_HS);
}
```

life.augmented

USBX class thread function flow



life.augmented

USBX class thread function flow

```
void usbx_class_thread_entry(ULONG arg)
{
  UX_SLAVE_DEVICE *device;
  UX_SLAVE_INTERFACE *data_interface;
  UX_SLAVE_CLASS_CDC_ACM *cdc_acm;
  device = &_ux_system_slave->ux_system_slave_device;
  while (1)
  {
      /* Check if device is configured */
    if (device->ux_slave_device_state == UX_DEVICE_CONFIGURED)
    {
    data_interface = device->ux_slave_device_first_interface[INTERFACE_NUMBER].ux_slave_interface_next_interface;

    ux_status = ux_utility_memory_compare(data_interface->ux_slave_interface_class->ux_slave_class_name,
          _ux_system_slave_class_cdc_acm_name, ux_utility_string_length_get(_ux_system_slave_class_cdc_acm_name));
      if (ux_status == UX_SUCCESS)
      {
          /* Perform class action*/
      }}
          tx_thread_sleep(x);
}}
```

USBX class thread get data interface

- Class data interface use same number like was given in ux_device_stack_class_register function

```
void usbx_class_thread_entry(ULONG arg)
{
UX_SLAVE_INTERFACE *data_interface;
   …
   data_interface = device->ux_slave_device_first_interface[INTERFACE_NUMBER].ux_slave_interface_next_interface;
   …
}
```

```
ret = ux_device_stack_class_register(_ux_system_slave_class_name,
                                     ux_device_class_entry, 1, INTERFACE_NUMBER,
                                     (VOID *)&class_parameter);
```

life.augmented

Template changes – ux_device_mouse

- Function *usbx_hid_thread_entry* handle HID action when device connected
  - Device connection check
  - Pick correct interface
  - Class action - moves mouse cursor
  - 5 second delay

Template changes – ux_device_cdc_acm

- Function *usbx_cdc_acm_thread_entry* handle CDC data transfer when device connected
  - Echo functionality of this example – read message from host and immediately write it back
  - In more complex application additional write thread should be considered
- Function *ux_app_parameters_change* manage line coding
  - CTS, RTS and other virtual COM port events may be handled here as well

CDC HID composite memory consumption

- Overall memory consumption (GCC, Optimize for size)

	RAM	Flash
Default STM32CubeMX parameters	27.09 kB	36.22 kB
Optimized STM32CubeMX parameters	14.73 kB	36.20 kB

- USBX and USBX tight threads RAM memory usage

	Description	Allocated memory [bytes]
USBX allocation		15104
Threads memory	cdc_acm_usbx_app_thread_entry	1024
	hid_usbx_app_thread_entry	1024
	main_usbx_app_thread_entry	1024
	ux_slave_hid_thread**	1024**
	Ux_slave_class_cdc_acm_bulkin_thread**	1024**
	Ux_slave_class_cdc_acm_bulkout_thread**	1024**
Total		18224

**allocated already inside USBX allocation

Application memory consumption

	Description	Allocated memory [bytes]
ux_system_initialize	ux_system memory structures	576
	margin for initial memory pointer	32
ux_device_stack_initialize	memory for the classes	576
	transfer request buffer	272
	pool for the interfaces	192
	pool for the endpoints	576
	endpoint data memory	8256
HID - ux_device_stack_class_register	hid class memory	128
	hid class stack	1040
	hide event aray	720
CDC - ux_device_stack_class_register	cdc class memory	576
	Tx and Rx CDC threads	2064
ux_dcd_stm32_initialize	USB peripheral structure	96
Total		15104

USBX references

- Microsoft Azure RTOS USBX documentation
- STM32 wiki USBX
- Example project in X-CUBE-AZRTOS-H7 package

life.augmented