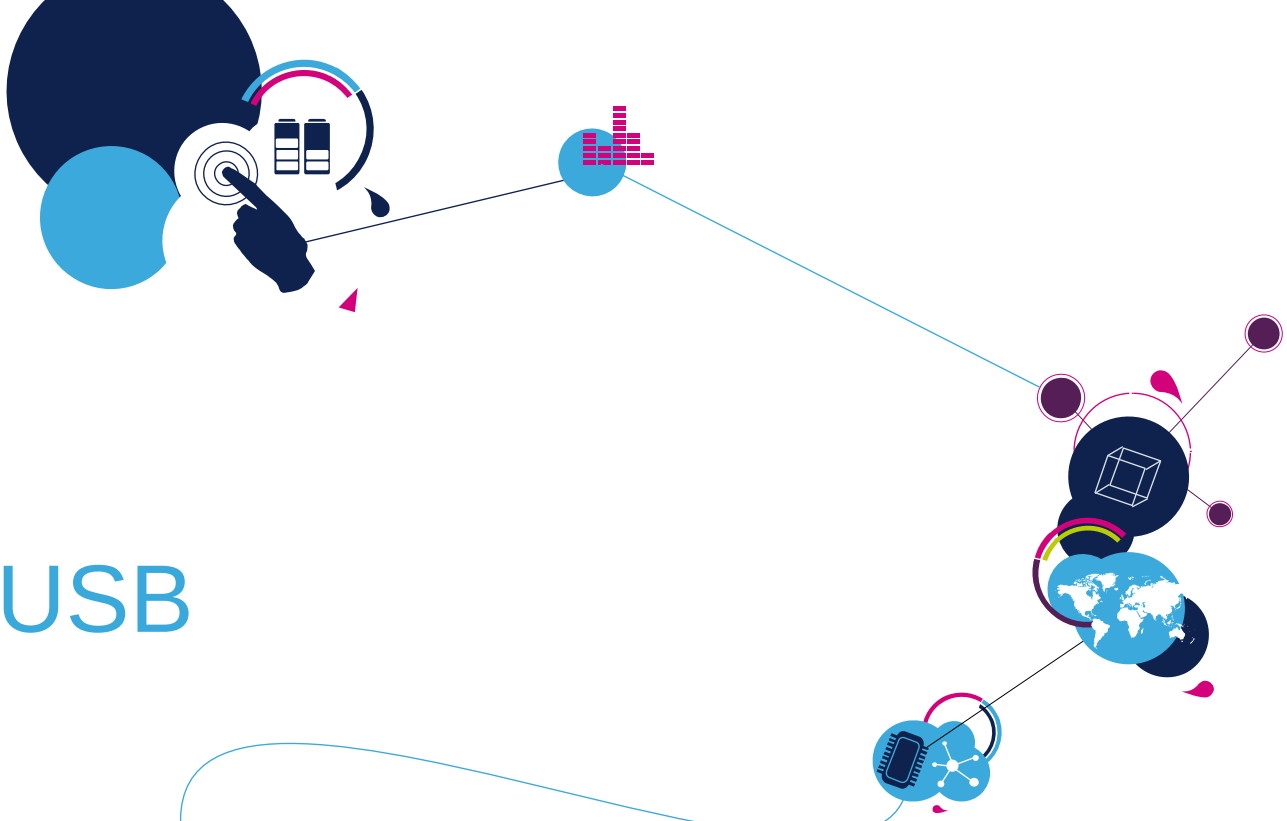
STM32 & USB

Ver. 1.0



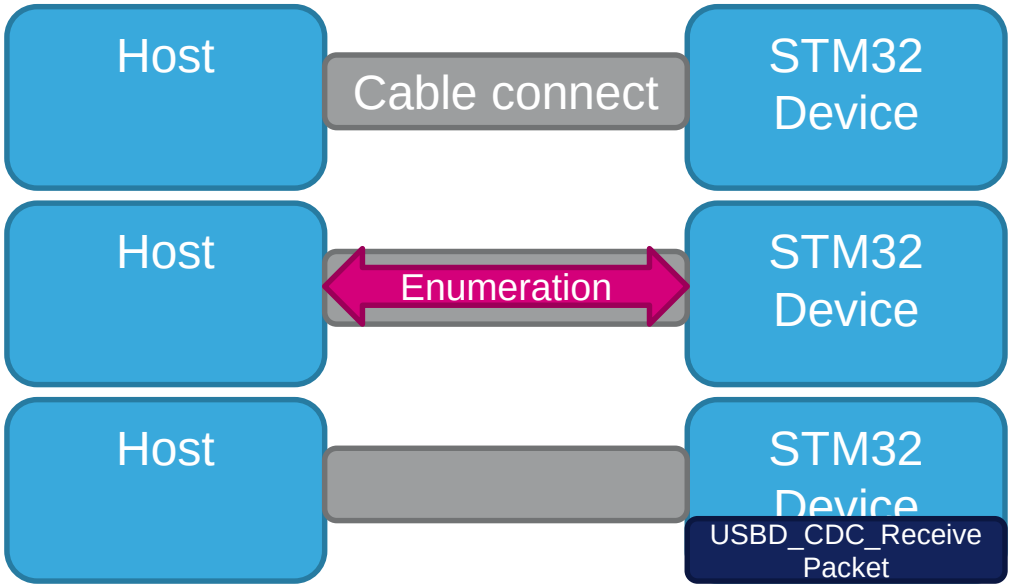
life.augmented

USB VCP Device with CubeMX

life.augmented

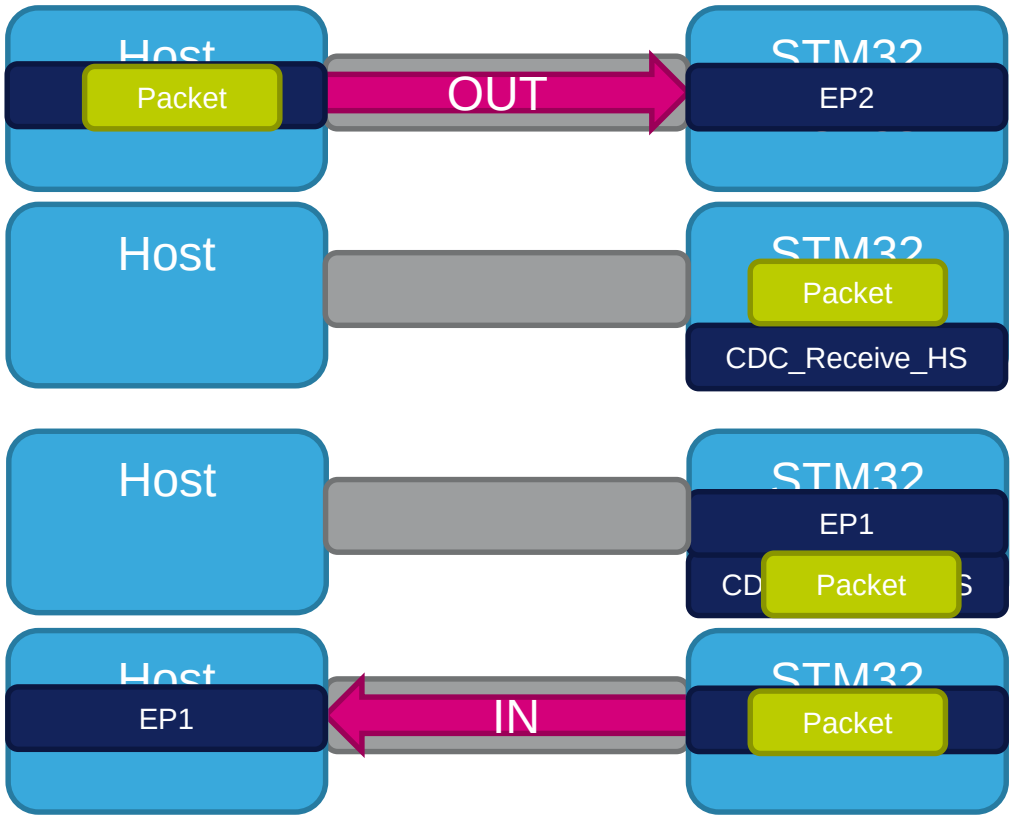
Cube VCP Functionality

- CDC FLOW 1/2
  - Endpoint 0 by default
  - Endpoint 1 bulk in
  - Endpoint 2 bulk out
  - Endpoint 3 Interrupt in(for control purposes)



Cube VCP Functionality

- CDC FLOW 2/2

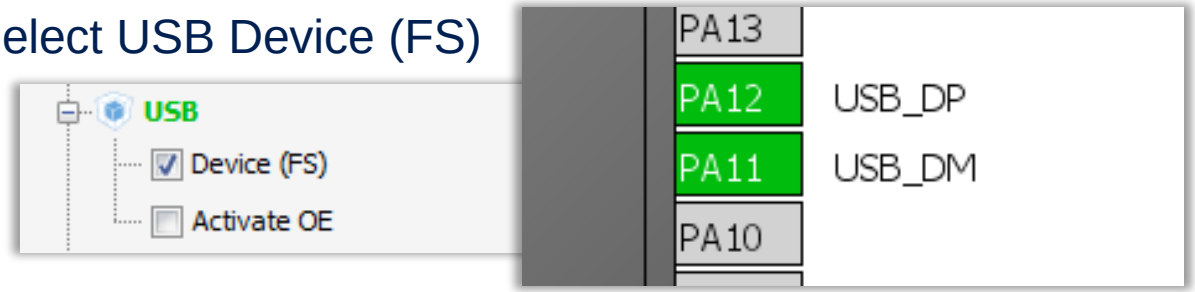


USB VCP Device
L0 crystall less

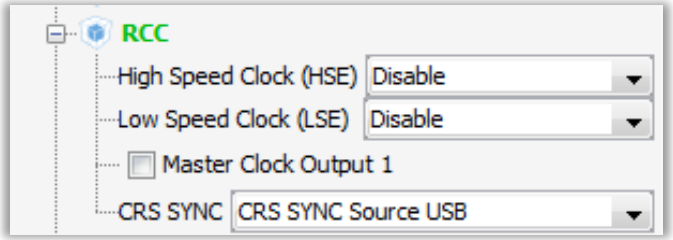
life.augmented

USB L0 VCP Device

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32L0 > STM32L0x3 > LQFP64 > STM32L053R8Tx
- Select USB Device (FS)



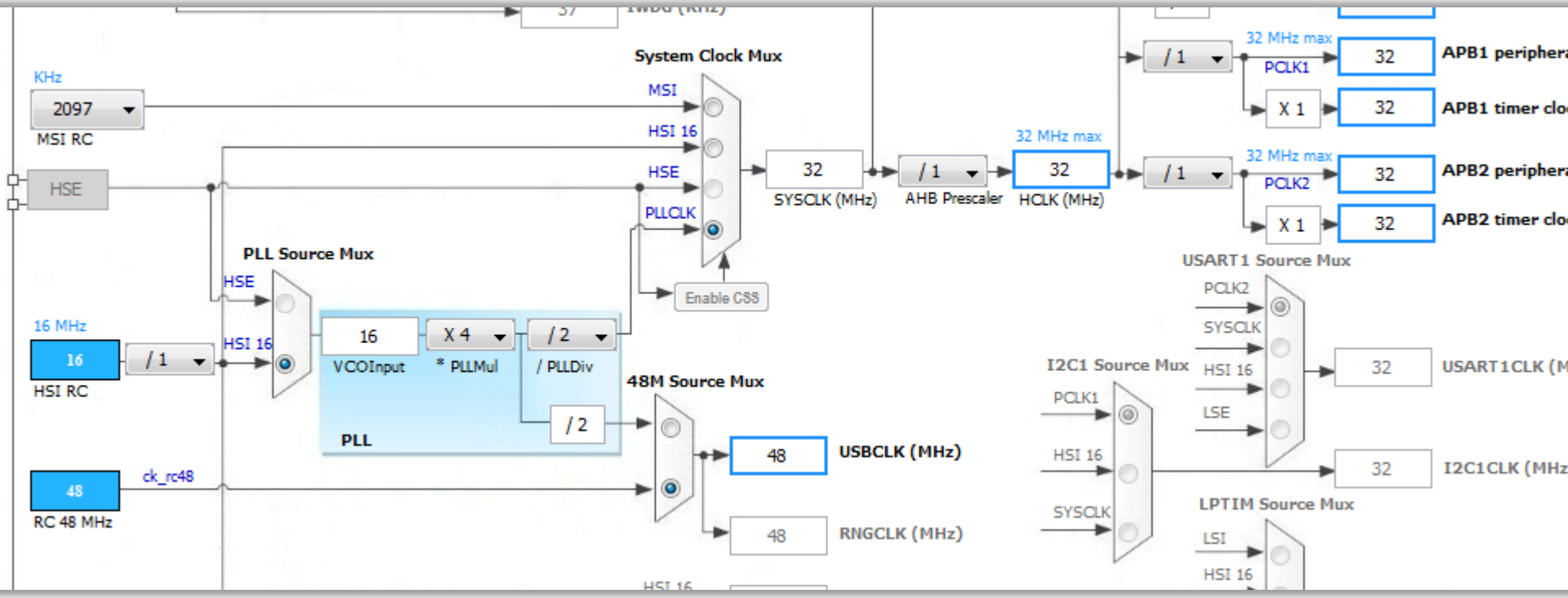
- Select RCC CRS SYNC to CRS SYNC Source USB
  - Because for crystal less device we need clock synchronization



- Select CDC class in MiddleWares

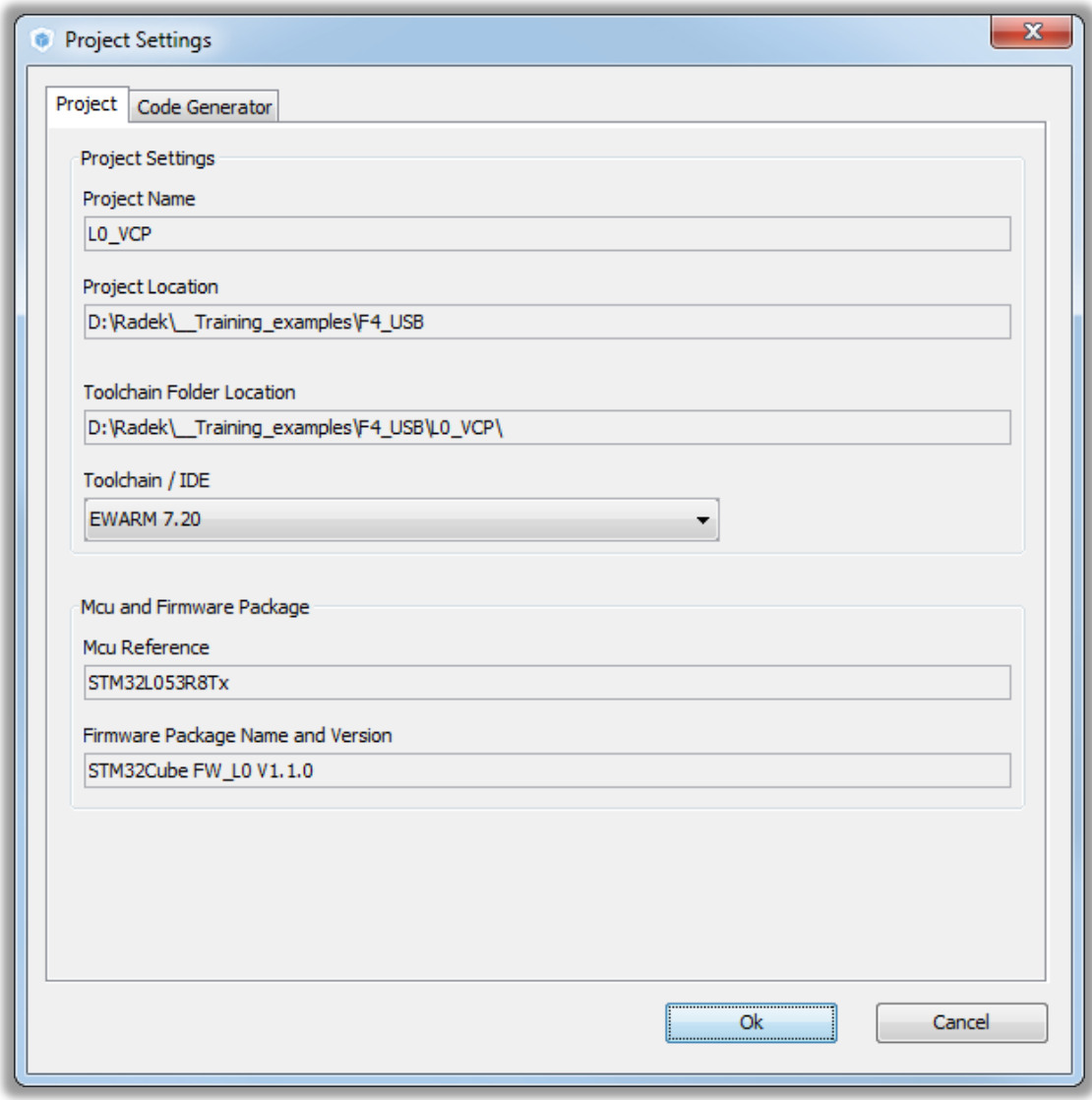
USB L0 VCP Device

- Configure RCC clocks
  - USBCLK source is RC48MHz
  - Clock core to 32MHz from HIS PLL mul is 4x and divider 2x
  - AHB/APB1/APB2 prescalers set to 1x



USB L0 VCP Device

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code

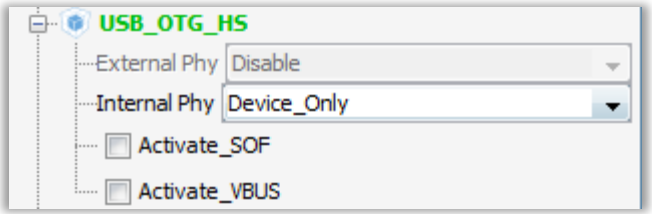


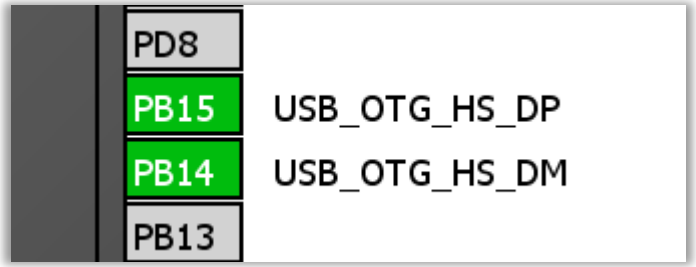
USB VCP Device
F429 - Discovery

life.augmented

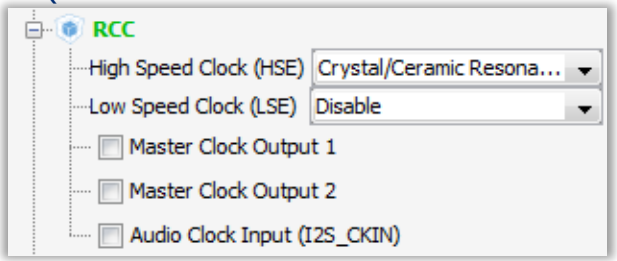
USB F4 VCP Device

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)

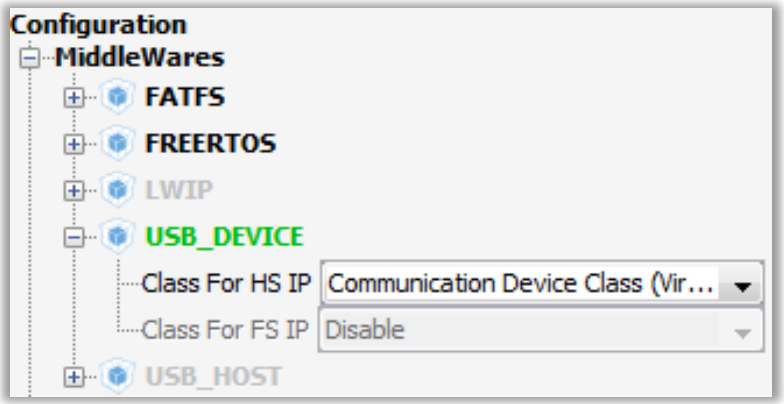




- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)

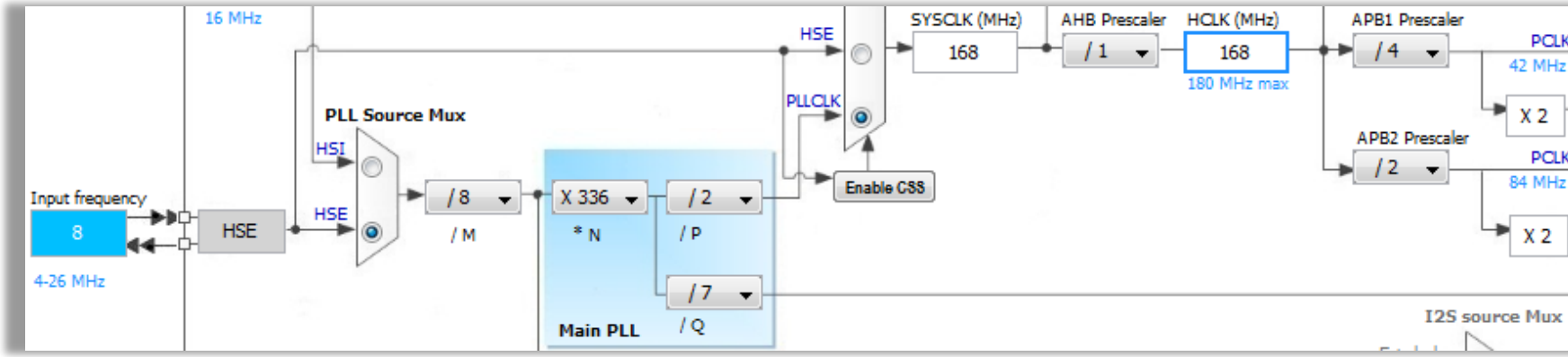


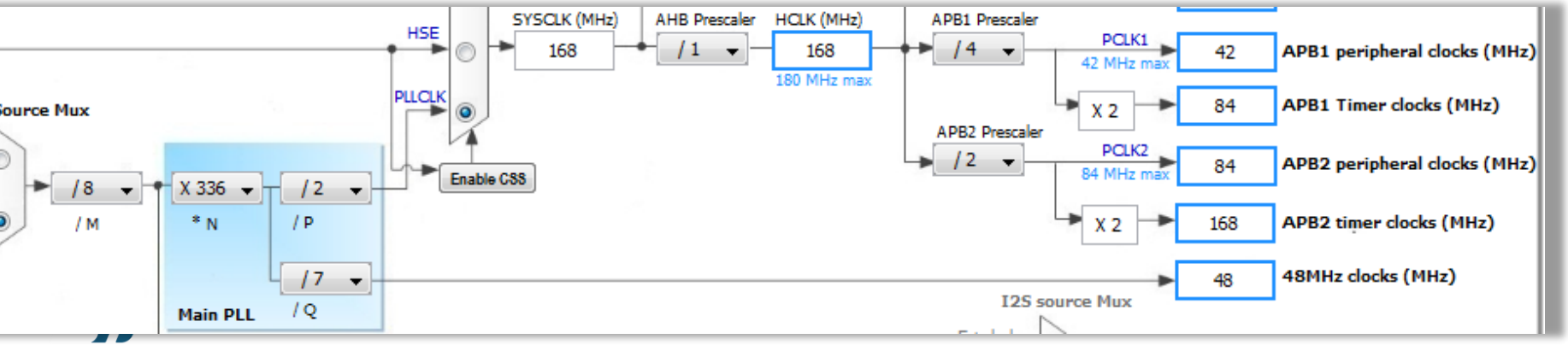
- Select CDC class in MiddleWares



USB F4 VCP Device

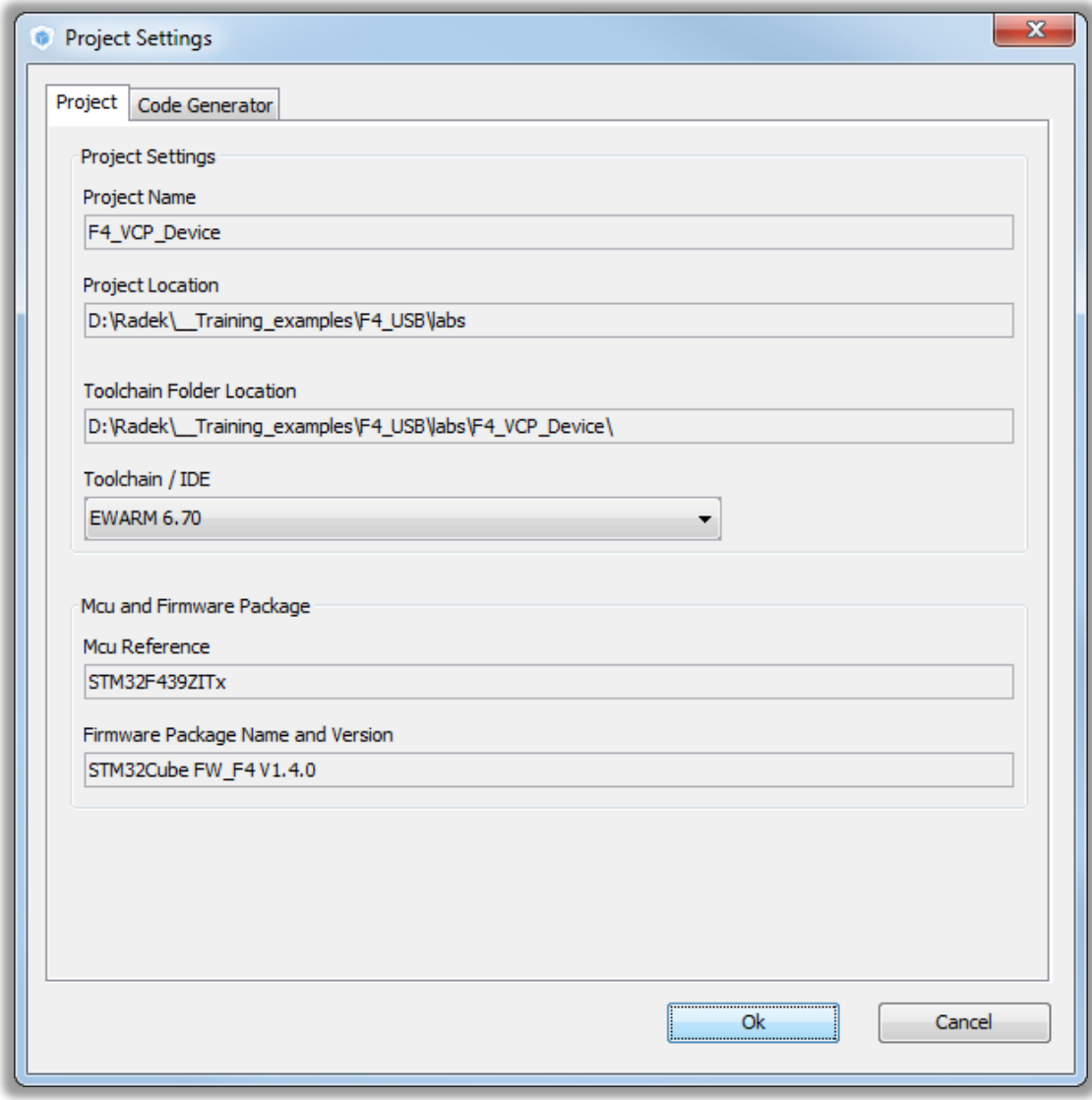
- Configure RCC clocks
  - For discovery kit set crystal frequency to 8MHz and M divider to 8x (1MHz)
  - PLL set to N multiplier to 336x and P divider to 2x(168MHz 180 is not possible) and Q divider to 7x(48MHz)
  - AHB prescaler to 1x, APB1 to 4x(42MHz) and APB2 to 2x(84MHz)





USB F4 VCP Device

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



USB VCP Device

- CubeMX will generate for you whole project
- For Keil is necessary in startup_stm32xxxx.s increase heap otherwise USB will be not functional(0x200 heap is to low for USB)

```
; <h> Heap Configuration
;   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
; </h>

Heap_Size       EQU     0x00000200
```

- Change it to:

```
; <h> Heap Configuration
;   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
; </h>

Heap_Size       EQU     0x00000800
```

- Then USB device will be successful enumerated

USB VCP Device

- How send receive data over VCP
- Function which handle VCP operation are in generated file usbd_cdc_if.c
- APP_RX_DATA_SIZE and APP_TX_DATA_SIZE define size of sending and receiving buffers

```
 /* USER CODE BEGIN 1 */
/* Define size for the receive and transmit buffer over CDC */
/* It's up to user to redefine and/or remove those define */
#define APP_RX_DATA_SIZE  64
#define APP_TX_DATA_SIZE  64
  /* USER CODE END 1 */
```

- Callback from control interface which allow to send COM port parameters
  Is used only if you really want to send data over COM port(UART)

```
static int8_t CDC_Control_FS  (uint8_t cmd, uint8_t* pbuf, uint16_t length)
```

USB VCP Device

- Receive callback function
- In case you wand to receive more bytes you must call USBD_CDC_ReceivePacket(hUsbDevice_0);
- Otherwise the USB will not accept any data until you call this function

```
static int8_t CDC_Receive_FS (uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 7 */
  USBD_CDC_ReceivePacket(hUsbDevice_0);
  return (USBD_OK);
  /* USER CODE END 7 */
}
```

USB VCP Device

- The Windows terminals using CDC commands to set correct line coding
- But they also want to read this coding back
- For this purpose we need to handle this actions
- This actions are done throe function:

```
static int8_t CDC_Control_FS  (uint8_t cmd, uint8_t* pbuf, uint16_t length)
```

- We use simply trick, we create buffer where we store this information from PC and the we can send them back

```
uint8_t tempbuf[6];
  /* USER CODE END 3 */
```

USB VCP Device

- This part in CDC_Control_FS handling the storing and riding part form buffer

```
case CDC_SET_LINE_CODING:
       tempbuf[0]=pbuf[0];
       tempbuf[1]=pbuf[1];
       tempbuf[2]=pbuf[2];
       tempbuf[3]=pbuf[3];
       tempbuf[4]=pbuf[4];
       tempbuf[5]=pbuf[5];
       tempbuf[6]=pbuf[6];
   break;

 case CDC_GET_LINE_CODING:
       pbuf[0]=tempbuf[0];
       pbuf[1]=tempbuf[1];
       pbuf[2]=tempbuf[2];
       pbuf[3]=tempbuf[3];
       pbuf[4]=tempbuf[4];
       pbuf[5]=tempbuf[5];
       pbuf[6]=tempbuf[6];
   break;
```

- Now will be communication with PC functional

USB VCP Device

- This function you need to call if you want to send data over VCP
- In CubeMX 4.6 wrong USBD_CDC_SetTxBuffer Buffer parameter, please correct it as bellow

```
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 8 */
  USBD_CDC_SetTxBuffer(hUsbDevice_0, Buf, Len);
  result = USBD_CDC_TransmitPacket(hUsbDevice_0);
  /* USER CODE END 8 */
  return result;
}
```

Example of wrong generated code

```
uint8_t CDC_Transmit_HS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 13 */
  USBD_CDC_SetTxBuffer(hUsbDevice_1, UserTxBufferHS, Len);
  result = USBD_CDC_TransmitPacket(hUsbDevice_1);
  /* USER CODE END 13 */
  return result;
}
```

Irelevant buffer change it to ‘Buf’ or store your data into this buffer

USB VCP Device

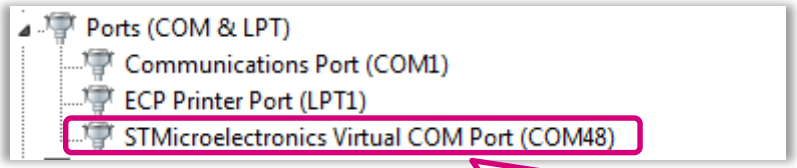
- If you want send lot of data with function CDC_Transmit_FS and you want to rewrite his buffer you must check first if the periphery release this buffer
- For this you need check the state of CDCUSBhandle something like this

```
if(((USBD_CDC_HandleTypeDef*)(hUsbDeviceFS.pClassData))->TxState==0){
   CDC_Transmit_FS(buffer,length);
 }
```

- The function first check if USB IN(Tx) is complete and allow to use transmit function
- Correct handling of transmit complete is use USBD_CDC_DataIn callback in usbd_cdc.c and implement callback to user application
  Unfortunately for this is necessary change library files!!

USB VCP Device

- Because Windows can select for VCP very high com port number you need the terminal where you can select the com number
- For example: http://realterm.sourceforge.net/
- If the USB is connected to PC it must be displayed in Device Manager



VCP with assigned port number

- In case you have no driver for VCP download it from: http://www.st.com/web/en/catalog/tools/FM147/CL1794/SC961/SS1533/PF257938?s_searchtype=keyword

USB VCP Device

- Simple Loopback only for testing!!!

```
static int8_t CDC_Receive_FS (uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 7 */
  CDC_Transmit_FS(Buf,*Len);
  USBD_CDC_ReceivePacket(hUsbDevice_0);
  return (USBD_OK);
  /* USER CODE END 7 */
}
```

```
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 8 */
  USBD_CDC_SetTxBuffer(hUsbDevice_0, Buf, Len);
  result = USBD_CDC_TransmitPacket(hUsbDevice_0);
  /* USER CODE END 8 */
  return result;
}
```

USB VCP Device

- Transmit will be still same

```
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 8 */
  USBD_CDC_SetTxBuffer(hUsbDevice_0, Buf, Len);
  result = USBD_CDC_TransmitPacket(hUsbDevice_0);
  /* USER CODE END 8 */
  return result;
}
```

VCP Zero Length Packet

- Communication over VCP with Windows is specific
- There is one problematic part which is not obvious
- The Windows require for end of in transfer packet smaller then maximum size or zero length packet
- **If this condition is not meet you will never see data in your application!!!!**

USB specification 2.0
Chapter 5.8.3

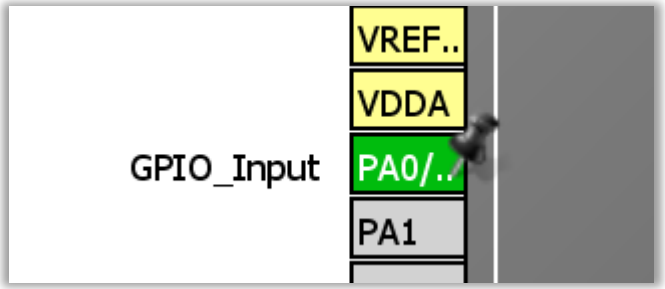
contain the remaining data. A bulk transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a packet with a payload size less than *wMaxPacketSize* or transfers a zero-length packet

Windows use in VCP this condition as end of transfer

VCP Zero Length Packet lab

- In CubeMX add PA0(Button) pin as input
- It will help with problem demonstration and protect terminal from spamming



- And regenerate code

VCP Zero Length Packet lab

- Corrected transmit function(usbf_cdc_if.c)

```
uint8_t CDC_Transmit_HS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  /* USER CODE BEGIN 13 */
  USBD_CDC_SetTxBuffer(hUsbDevice_1, Buf, Len);
  result = USBD_CDC_TransmitPacket(hUsbDevice_1);
  /* USER CODE END 13 */
  return result;
}
```

- We don’t need to do anything with receive

```
static int8_t CDC_Receive_HS (uint8_t* Buf, uint32_t *Len)
{
  /* USER CODE BEGIN 12 */
  return (USBD_OK);
  /* USER CODE END 12 */
}
```

VCP Zero Length Packet lab

- Include the usbd_cdc_if.h into main.c this allow us to use Transmit function

```
/* USER CODE BEGIN Includes */
#include "usbd_cdc_if.h"
/* USER CODE END Includes */
```

- Create buffer and buffer length variable and variable for loop limiting purpose, define extern USB handle(only for OTG devices)

```
/* USER CODE BEGIN PFP */
uint8_t buffer[64];
uint8_t length=64;
uint8_t count=0;
extern USBD_HandleTypeDef hUsbDeviceHS;
/* USER CODE END PFP */
```

VCP Zero Length Packet lab

- We will wait on PA0 button press
- After that program sent 5x buffer 64byte length
- But on windows terminal we not get any data

```
/* USER CODE BEGIN 2 */
while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_RESET){
}
 while(count<5){
   if(((USBD_CDC_HandleTypeDef*)(hUsbDeviceHS.pClassData))->TxState==0){
     if(CDC_Transmit_HS(buffer,length)==USBD_OK){
       count++;
      }
    }
 }
 /* USER CODE END 2 */
```

PA0 button press check

Data Send with value check

Check if is possible sent data

- Try to decrease length variable to for example to 63

VCP Zero Length Packet lab

- Same situation as on previous slide but now we send zero length packet on the end (length is 64)

```
/* USER CODE BEGIN 2 */
 while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_RESET){
 }
 while(count<5){
   if(((USBD_CDC_HandleTypeDef*)(hUsbDeviceHS.pClassData))->TxState==0){
     if(CDC_Transmit_HS(buffer,length)==USBD_OK){
       count++;
     }
   }
 }
 while(((USBD_CDC_HandleTypeDef*)(hUsbDeviceHS.pClassData))->TxState!=0)
 {
 }
 CDC_Transmit_HS(buffer,0);
 /* USER CODE END 2 */
```

Check if is possible send data and ZLP send

- Now windows terminal will receive data

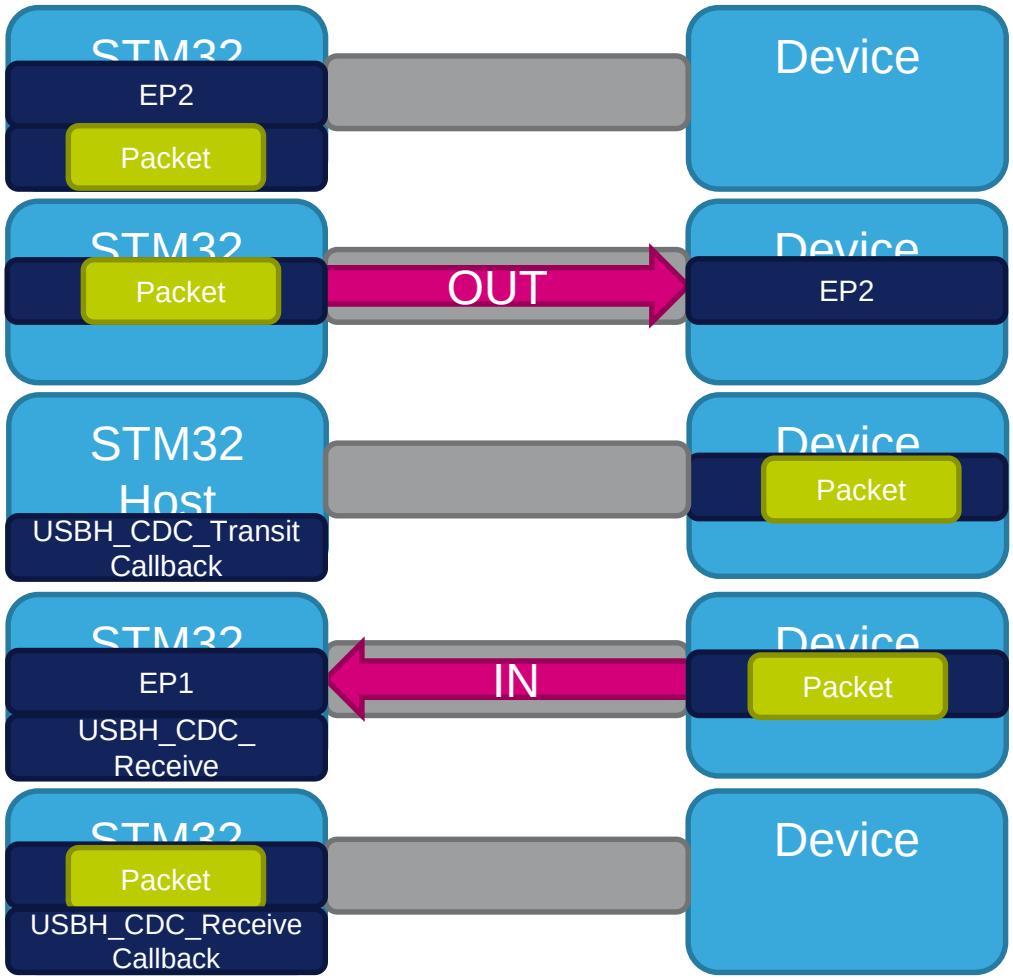
USB VCP Host

USB F4 VCP Host

- The CubeMX CDC host is very easy to handle
- There inly few function to handle
- Most important thing is function USBH_Process which must be periodically called
- This function us periodically called from main.c in projects generated by CubeMX
- For sending data over CDC we use function USBH_CDC_Transmit
- And for reading data from device USBH_CDC_Receive
- USBH_CDC_TransmitCallback is weak call-back called when data was succesfouly transferred
- USBH_CDC_ReceiveCallback is called when data was received

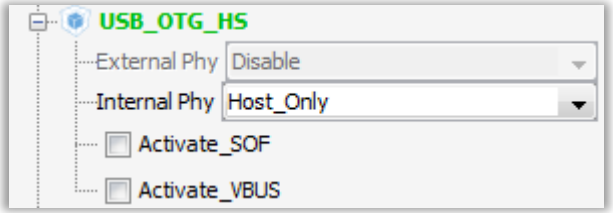
Cube VCP HOST Functionality

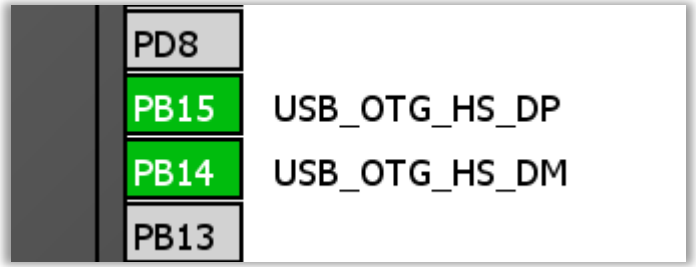
- CDC HOST FLOW



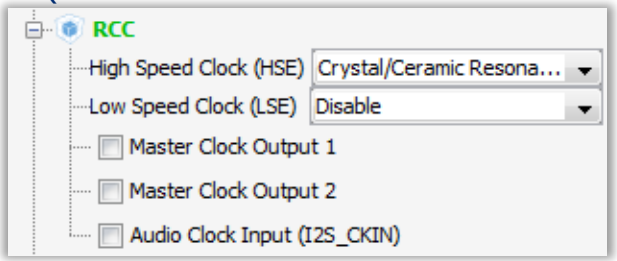
USB F4 VCP Host lab

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)

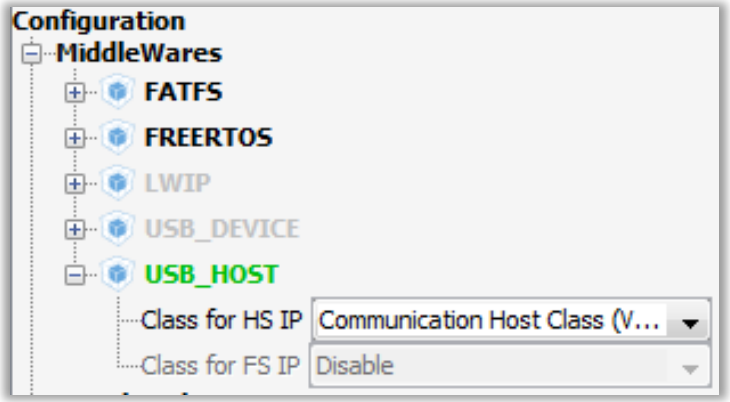




- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)



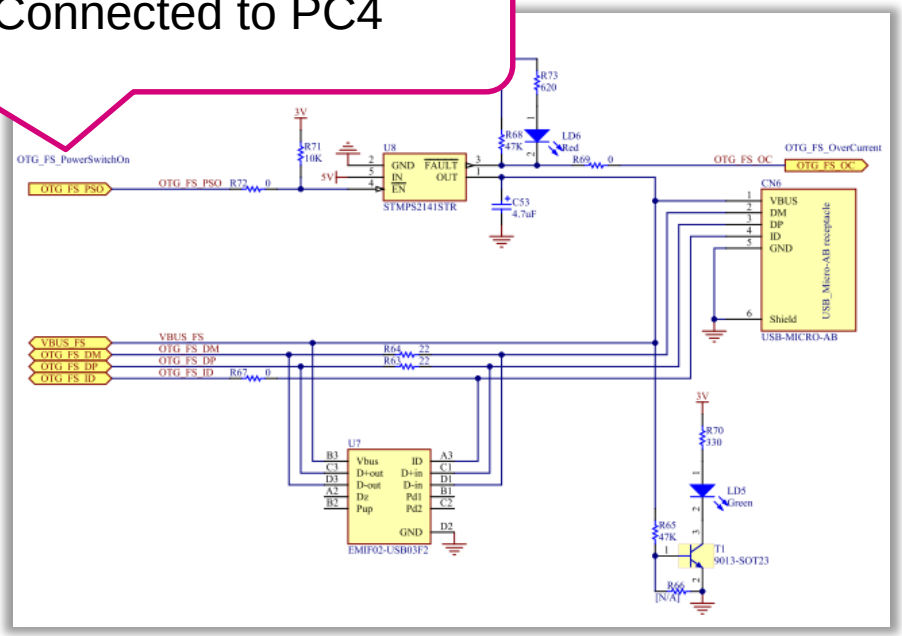
- Select CDC class in MiddleWares

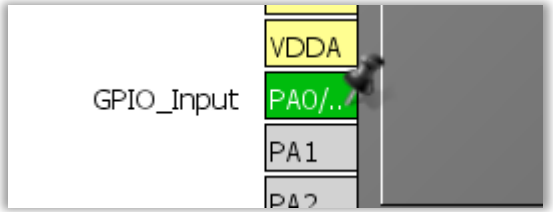


USB F4 VCP Host lab

- Because HOST must also power the device we need to enable voltage regulator connected to VBUS line
- Regulator enable pin is on PC4(only select as output is enough because default state then will be LOW)

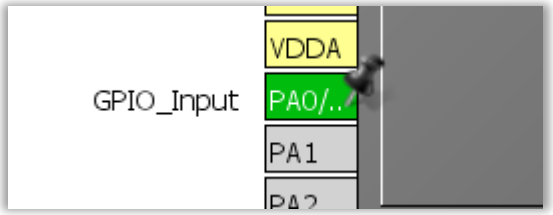
Connected to PC4





USB F4 VCP Host lab

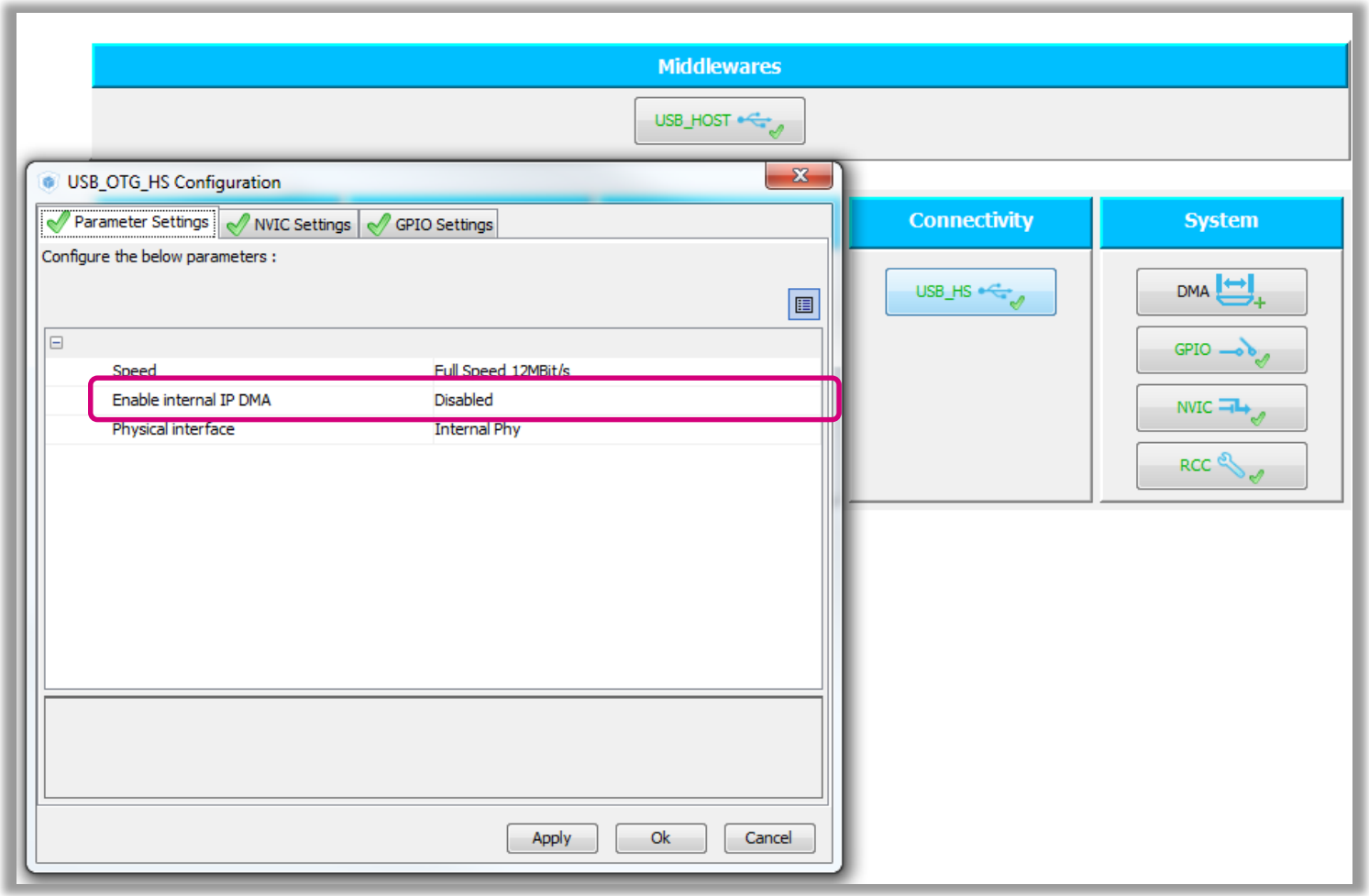
- We also enable PA0 where is button only for demo purpose



- USB clock set to 48MHz and core clock at maximum

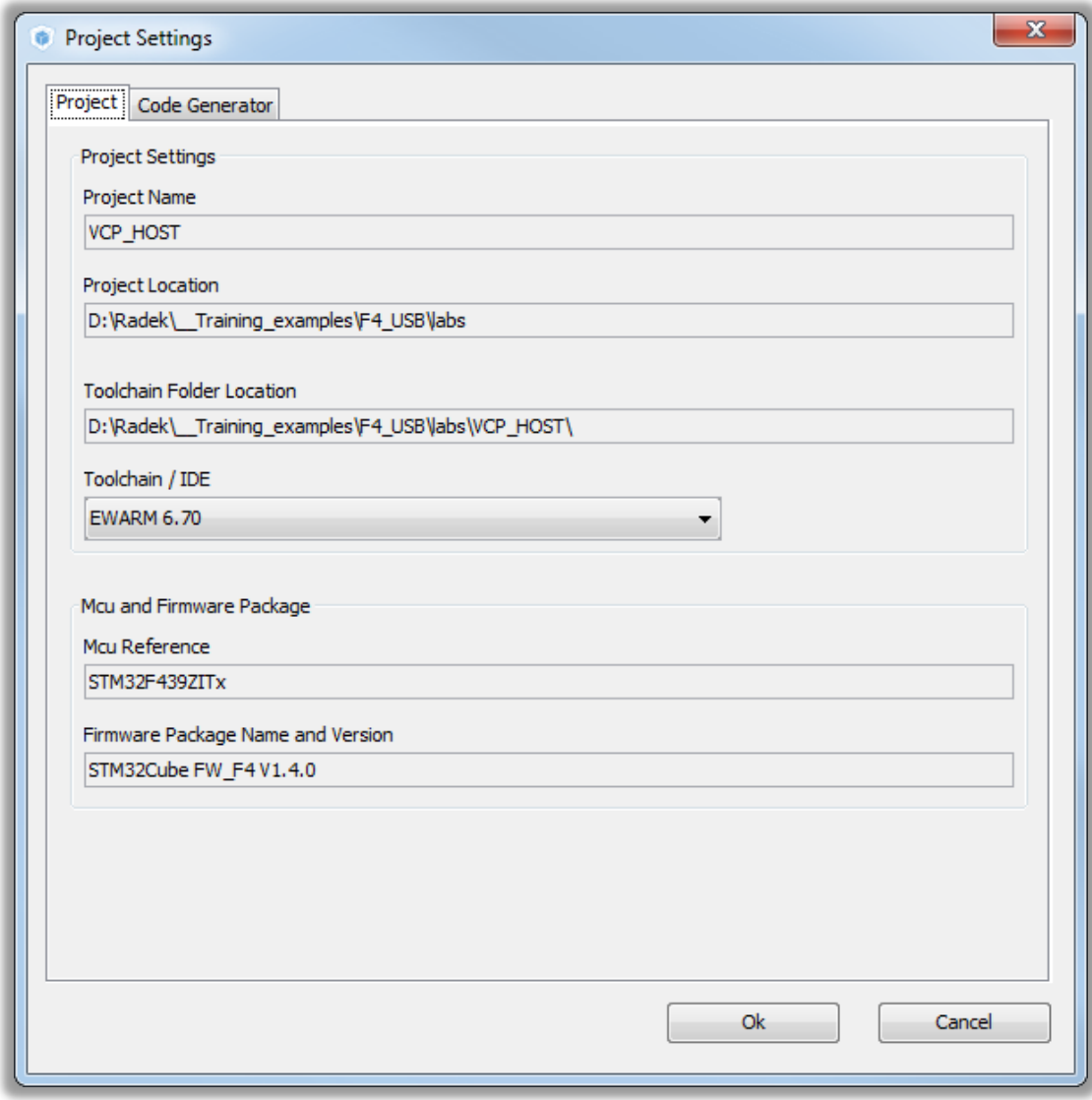
USB F4 VCP Host lab

- In Configuration tab select USB_HS in Connectivity
- Disable option use internal DMA
- Button OK



USB F4 VCP Device

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code
- If you have KEIL change HEAP size in startup file



USB F4 VCP Host lab

- In main.c is additional function MX_USB_HOST_Process this function must be periodically called, if not USB Host will be not functional

```
/* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   MX_USB_HOST_Process();
 }
 /* USER CODE END 3 */
```

- CubeMX generate is in infinite loop put I recommend you to handle it by interrupt or in RTOS put it into task

USB F4 VCP Host lab

- In usb_host.c you may find callbacks from CDC
- USBH_UserProcess callback storing state of connected device into Appli_state variable
- If the Device is connected and enumerated into Appli_state is stored APPLICATION_READY and we can commutate with device

```
/*
 * user callbak definition
*/
static void USBH_UserProcess  (USBH_HandleTypeDef *phost, uint8_t id)
{
  /* USER CODE BEGIN 2 */
  switch(id)
  {
  case HOST_USER_SELECT_CONFIGURATION:
  break;
  case HOST_USER_DISCONNECTION:
  Appli_state = APPLICATION_DISCONNECT;
  break;
  case HOST_USER_CLASS_ACTIVE:
  Appli_state = APPLICATION_READY;
  break;
  case HOST_USER_CONNECTION:
  Appli_state = APPLICATION_START;
  break;
  default:
  break;
  }
  /* USER CODE END 2 */
}
```

Device not connected

Device can communicate

USB F4 VCP Host lab

- In usb_host.c we define buffers for sending data and receiving

```
/* USER CODE BEGIN 0 */
uint8_t rx_buffer[100];
uint8_t tx_buffer[]="Hello\n";
/* USER CODE END 0 */
```

- In user section we define function which will send data into CDC device after button press

```
/* USER CODE BEGIN 1 */
void userFunction(void);
void userFunction(void){
  if(Appli_state==APPLICATION_READY){
    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_SET){
      USBH_CDC_Transmit(&hUsbHostHS,tx_buffer,0x9);
    }
  }
}
```

Check if we can communicate with device

Send data to host if the button is pressed
We send tx_buffer long 9bytes

USB F4 VCP Host lab

- In usb_host.c we also define two callbacks
- USBH_CDC_TransmitCallback which is called when data was successfully transmitted
- USBH_CDC_ReceiveCallback called if data was received

After data was transmitted to CD device we Request reading from CDC device

```
void USBH_CDC_TransmitCallback(USBH_HandleTypeDef *phost){
  USBH_CDC_Receive(phost,rx_buffer,0x9);
}

void USBH_CDC_ReceiveCallback(USBH_HandleTypeDef *phost){
  printf(rx_buffer);
}
/* USER CODE END 1 */
```

When data was read from device we print them to terminal(SWO)

USB F4 VCP Host lab

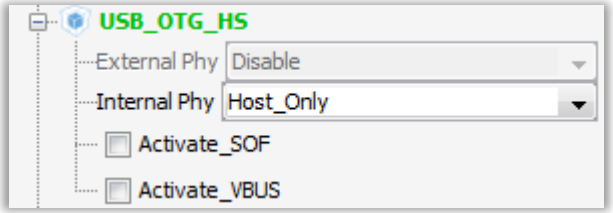
- Now only thing what is missing is call userFunction which will send data after button press
- I put it into MX_USB_HOST_Process is not ideal because CubeMX can regenerate it but for demonstration purpose it is inapt

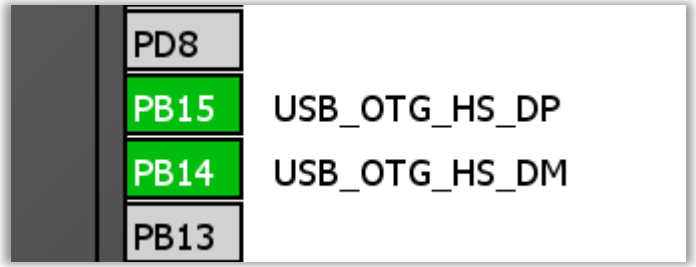
```
/*
 * Background task
*/
void MX_USB_HOST_Process()
{
  /* USB Host Background task */
    USBH_Process(&hUsbHostHS);
    userFunction();
}
```

USB MSP Host lab

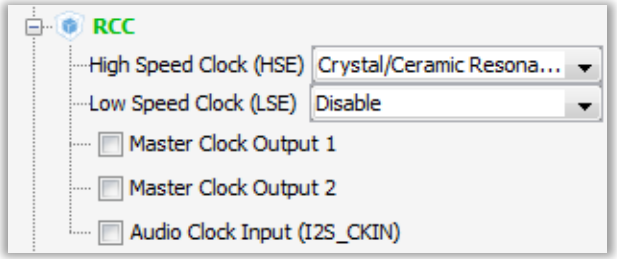
USB MSP Host lab

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)

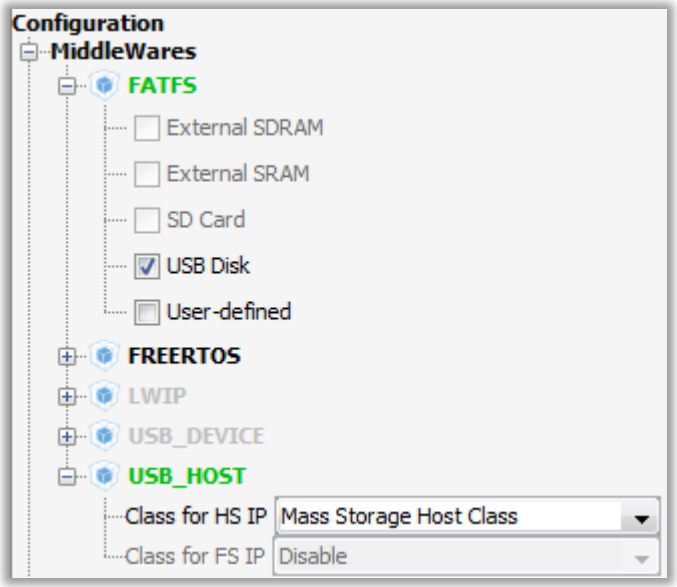




- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)



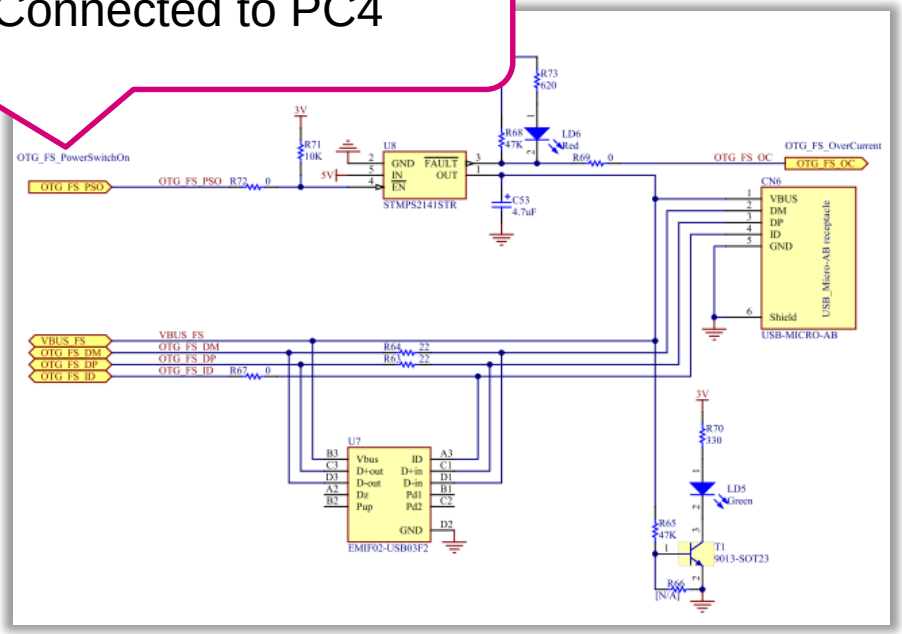
- Select MSP class in MiddleWares and FATFS USB Disk

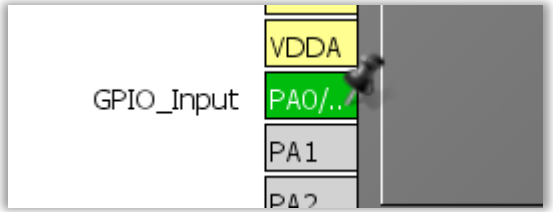


USB MSP Host lab

- Because HOST must also power the device we need to enable voltage regulator connected to VBUS line
- Regulator enable pin is on PC4(only select as output is enough because default state then will be LOW)

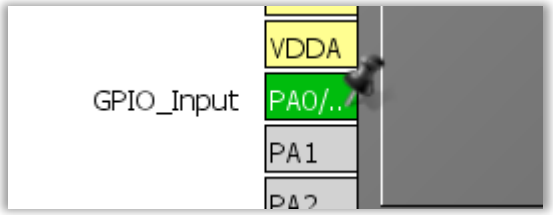
Connected to PC4





USB MSP Host lab

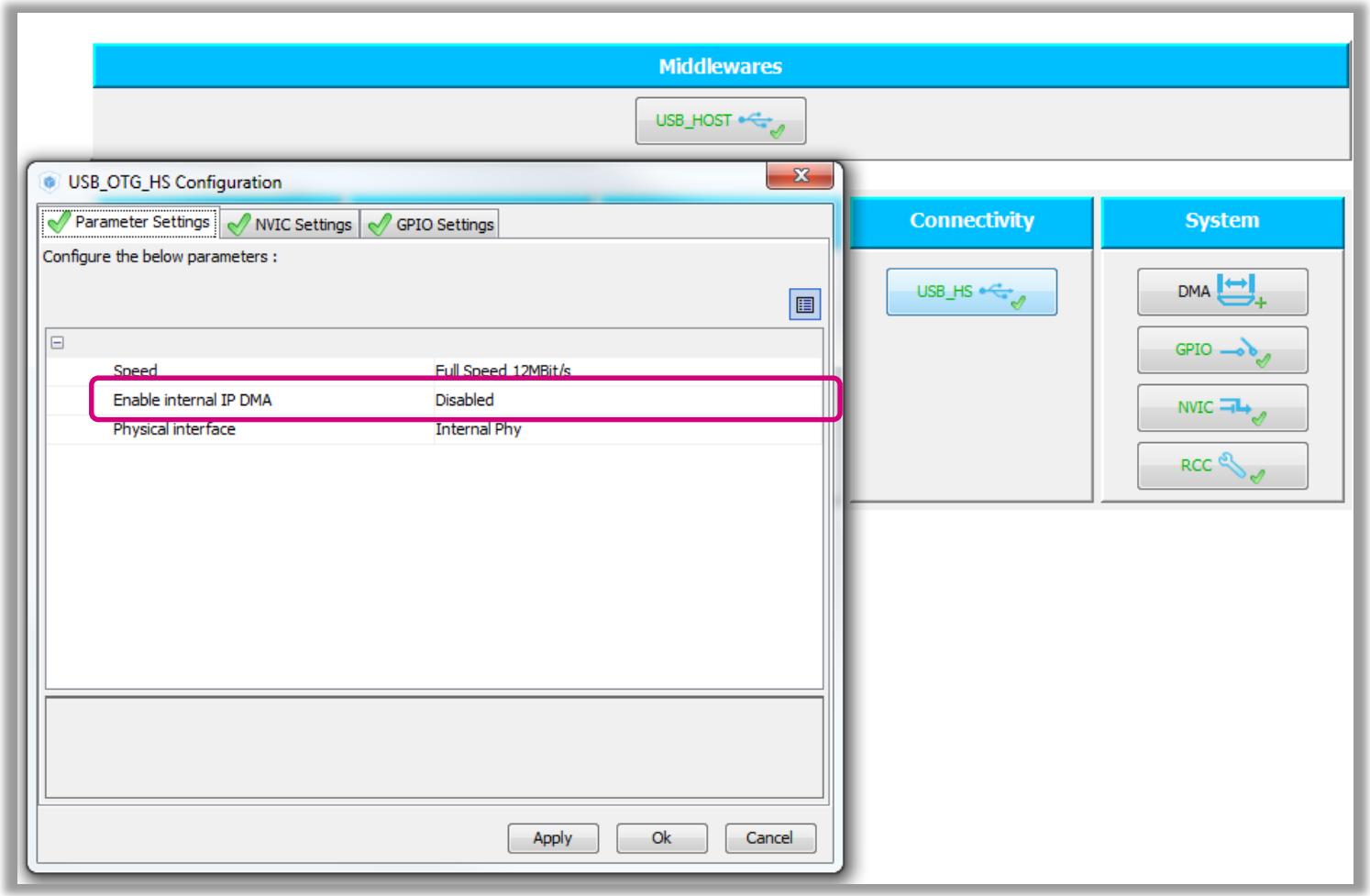
- We also enable PA0 where is button only for demo purpose



- USB clock set to 48MHz and core clock at maximum

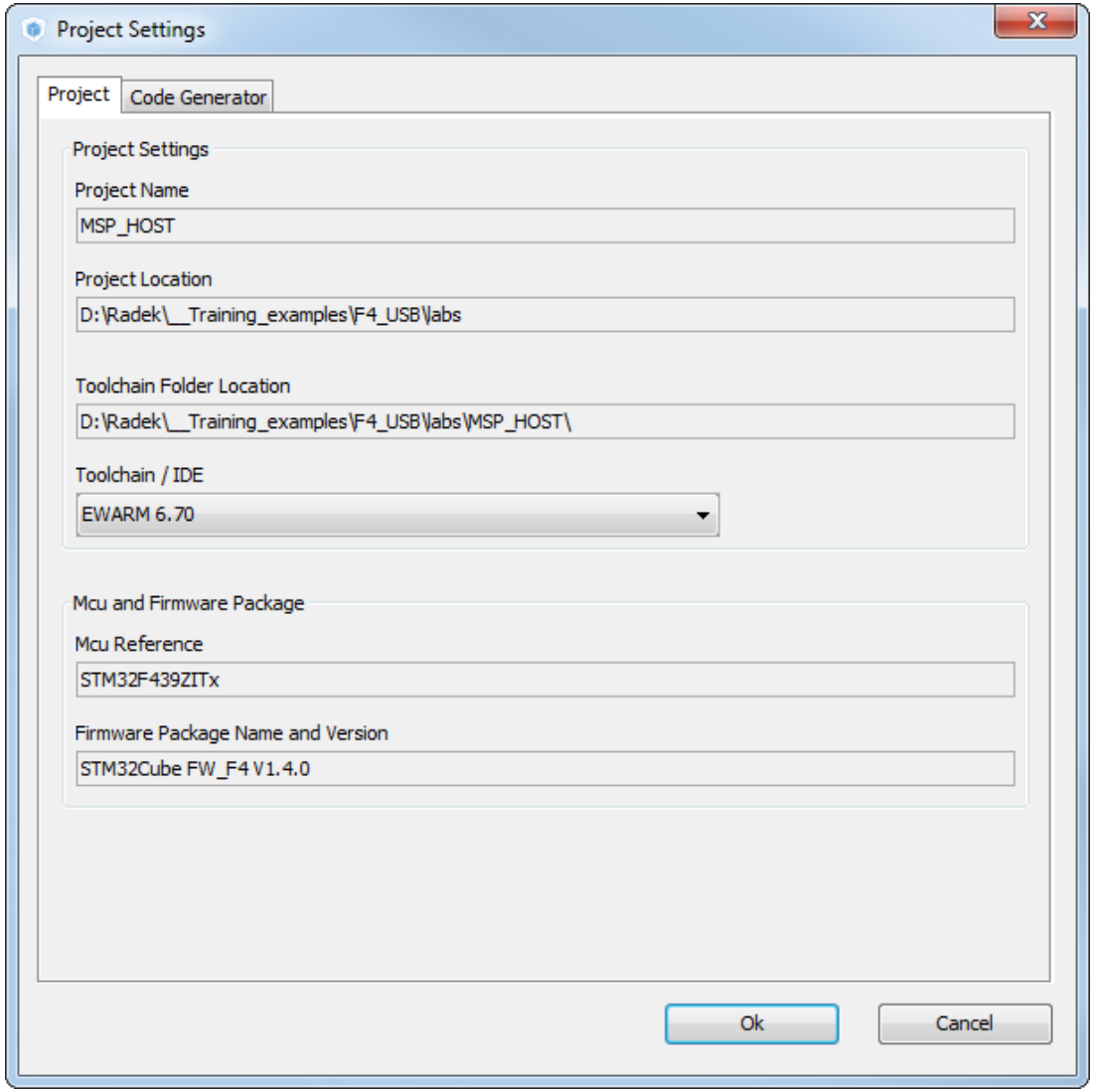
USB MSP Host lab

- In Configuration tab select USB_HS in Connectivity
- Disable option use internal DMA
- Button OK



USB MSP Host lab

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code
- If you have KEIL change HEAP size in startup file



USB MSP Host lab

- If the Device is connected and enumerated into appli_state is stored APPLICATION_READY and we can commutate with device

- For this reason we import into main.c appli_state variable

```
extern ApplicationTypeDef Appli_state;
```

- We also need FATFS variable and FIL for file operations

```
/* USER CODE BEGIN PV */
extern ApplicationTypeDef Appli_state;
FIL fp; //file handle
FATFS fatfs; //structure with file system information
char text[]="test";//text which will be written into file
char name[]="test.txt";//name of the file
char text2[100];//buffer for data read from file
uint32_t ret;//return variable
/* USER CODE END PV */
```

- Other variable are for lab purposes

USB MSP Host lab

- First we need mount the USB flash disk.

```
/* USER CODE BEGIN 3 */
  /* Initialises the File System*/
  if ( f_mount( &fatfs,"" ,0) != FR_OK )
     {
    /* fs initialisation fails*/
    while(1);
  }
```

- Please note that FLASH disk must be formatted in FAT32 file system otherwise is not possible to mount it

USB MSP Host lab

- Basic operation with file system, reading and writing data from file “text.txt”

```
/* Infinite loop */
  while (1)
  {
    MX_USB_HOST_Process();
    if(Appli_state==APPLICATION_READY){
      /*open or create file for writing*/
      if(f_open(&fp,name,FA_CREATE_ALWAYS | FA_WRITE)!=FR_OK){
        while(1);
      }
      /*write data into flashdisk*/
      if(f_write(&fp,text,strlen(text),&ret)!=FR_OK){
        while(1);
      }
      f_close(&fp);
      /*open file for reading*/
      if(f_open(&fp,name,FA_READ)!=FR_OK){
        while(1);
      }
      /*red data from flash*/
      if(f_read(&fp,text2,100,&ret)!=FR_OK){
        while(1);
      }
      f_close(&fp);
    }
  }
  /* USER CODE END 3 */
```

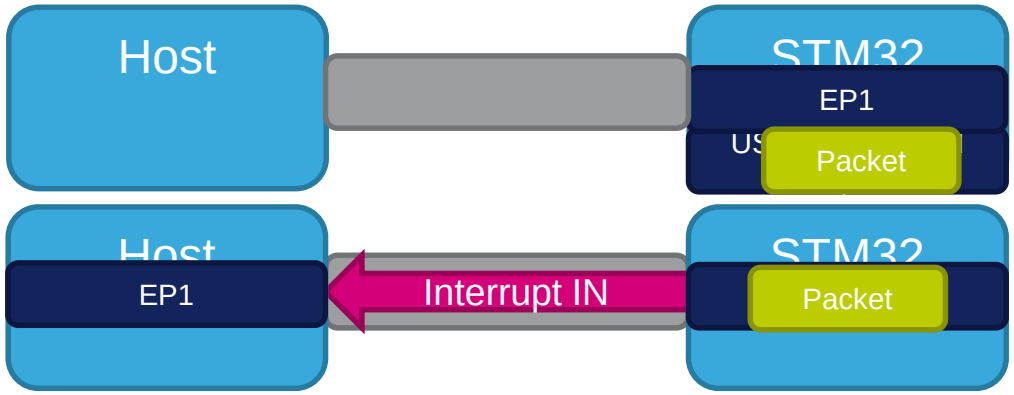
USB MSP Host lab

- From the past we know that some flash sticks can have problems with out library(STD)
- The USB MSP library is now only interface between flash drive and file system
- The basic operation which are done with MSP USB part is calling two BULK transfer one for READ BLOCK and second WRITE BLOCK

USB HID Device lab

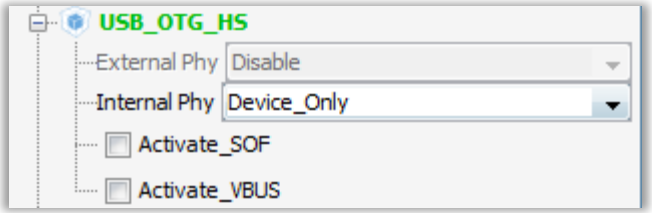
USB HID Device lab

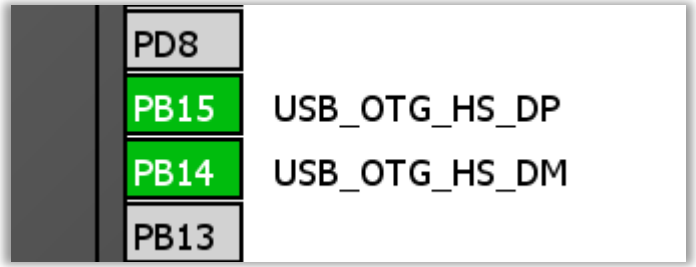
- HID device communicate over interrupt endpoint which guarantee the delivery in finite time
- In our CubeMX library is implemented the mouse report descriptor
- For change it you need to modify report descriptor first



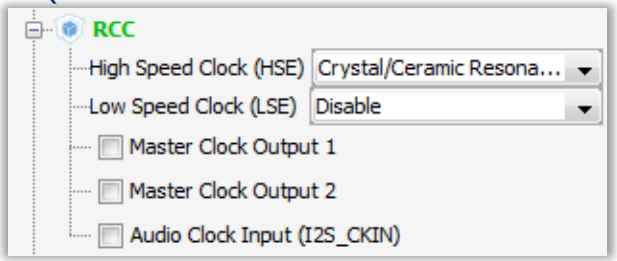
USB HID Device lab

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)

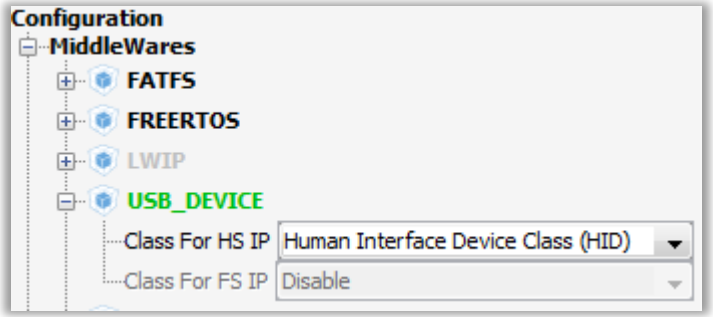




- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)

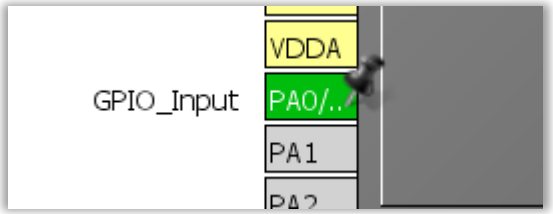


- Select HID class in MiddleWares



USB HID Device lab

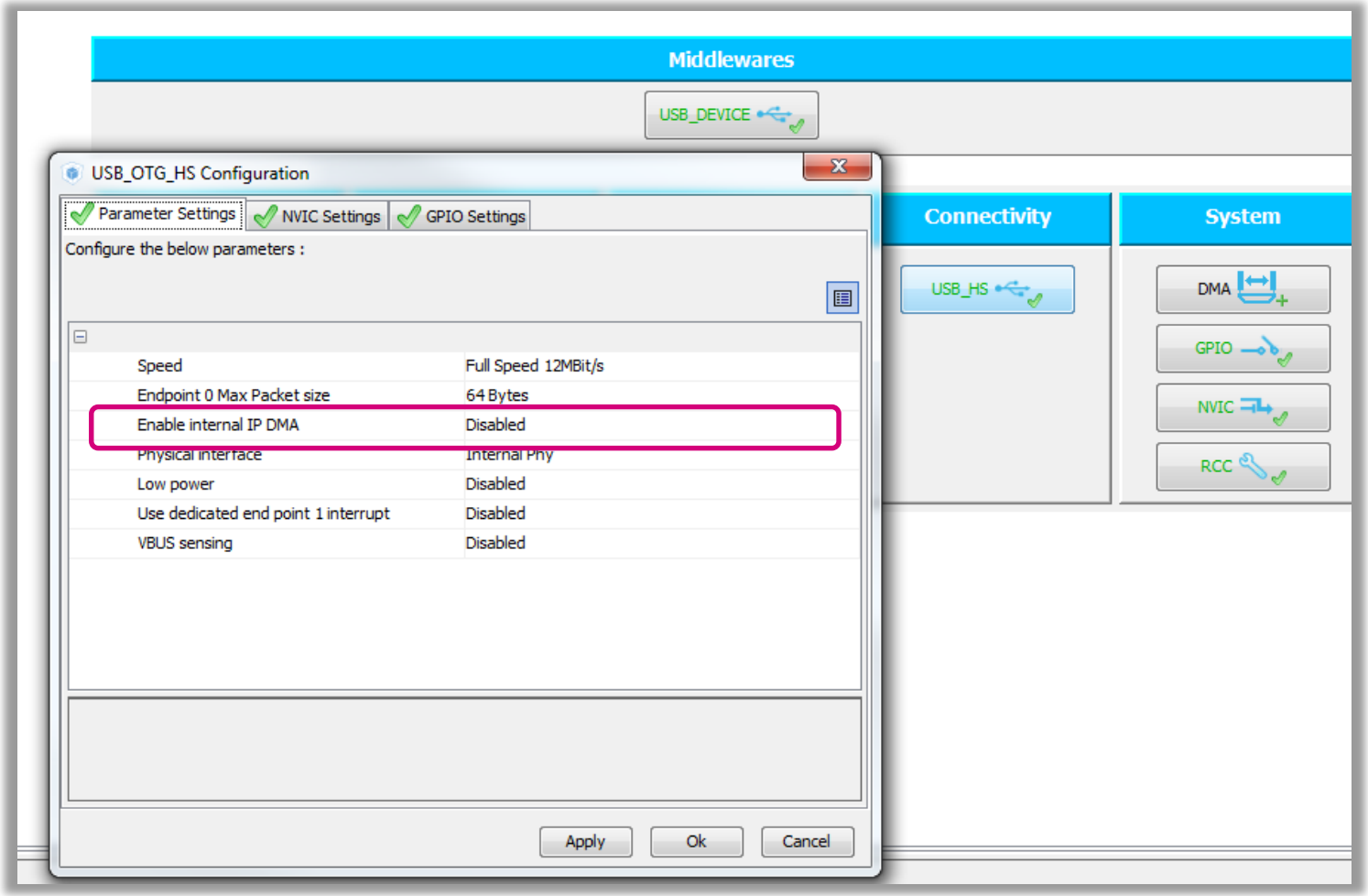
- We also enable PA0 where is button only for demo purpose



- USB clock set to 48MHz and core clock at maximum

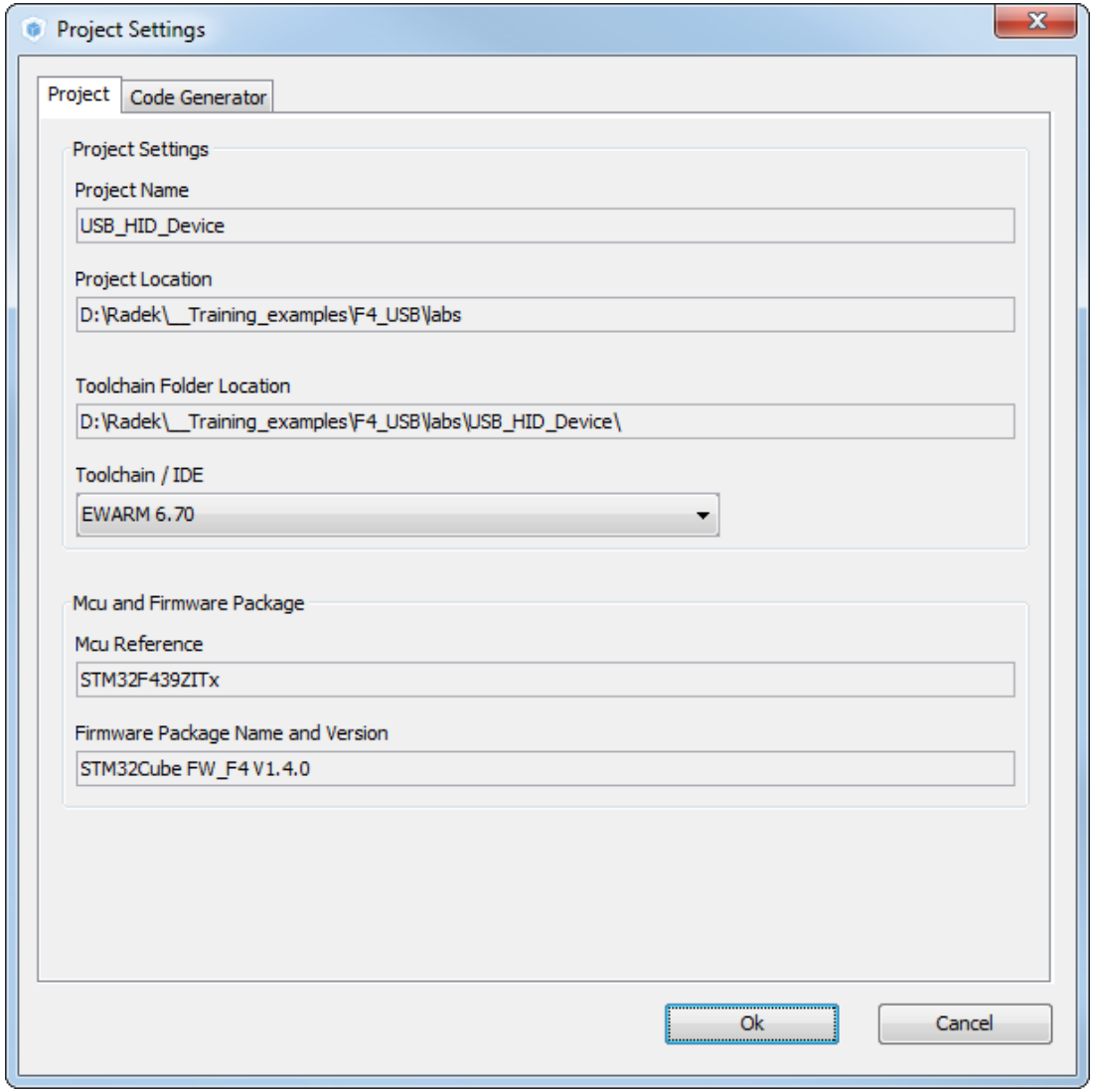
USB HID Device lab

- In Configuration tab select USB_HS in Connectivity
- Disable option use internal DMA
- Button OK



USB HID Device lab

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code
- If you have KEIL change HEAP size in startup file



USB HID Device lab

- The message which the HID device send have format defined in REPORT descriptor
- This format have only basic rules but descriptor for one device can look very different but functionality will be same
- Handling and parsing descriptors is on host
- Descriptor generated by CubeMX PC expects in this format:

- If you want to change format of this message you need to change the REPORT DESCRIPTOR in file usbd_hid.c the report descriptor array is called HID_MOUSE_ReportDesc

USB HID Device lab

- We will work only in main.c
- First include the USB handle

```
/* USER CODE BEGIN PV */
extern USBD_HandleTypeDef hUsbDeviceHS;
/* USER CODE END PV */
```

- And include hid header file

```
/* USER CODE BEGIN Includes */
#include "usbd_hid.h"
/* USER CODE END Includes */
```

- Define buffer which will be send to the host

```
/* USER CODE BEGIN PFP */
uint8_t buffer[4];
/* USER CODE END PFP */
```

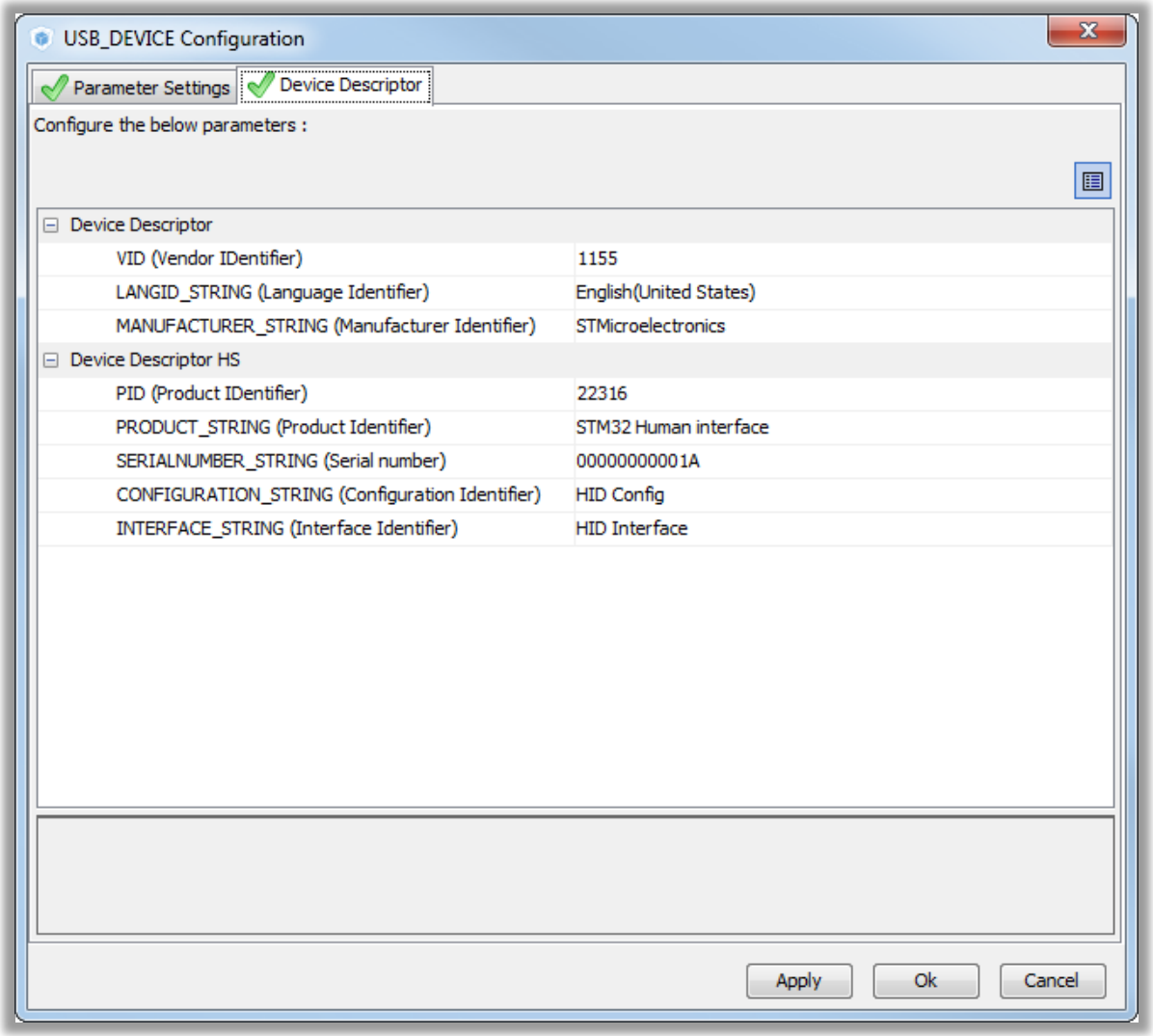
USB HID Device lab

- USBD_HID_SendReport will send the buffer on button press
- The buffer variable contains data about the mouse move and state of buttons
- With settings bellow, every button press move with cursor

```
/* USER CODE BEGIN 2 */
 buffer[0]=0;//buttons first 3 bits
 buffer[1]=100;//X axis 8bit value signed
 buffer[2]=0;//Y axis 8bit value signed
 buffer[3]=0;//Wheel 8bit value signed
 /* USER CODE END 2 */

 /* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_SET){
     USBD_HID_SendReport(&hUsbDeviceHS,buffer,4);
     HAL_Delay(100);
   }
 }
 /* USER CODE END 3 */
```

Change to HID Keyboard Lab

- In CubeMX change PID to 22316
- And regenerate code



Change to HID Keyboard Lab

- In usbd_hid.h
- Change size of report descriptor to 187

```
#define HID_MOUSE_REPORT_DESC_SIZE    187
```

- In usbd_hid.c change the protocol interface to keyboard

```
/*************** Descriptor of Joystick Mouse interface ****************/
 /* 09 */
 0x09,         /*bLength: Interface Descriptor size*/
 USB_DESC_TYPE_INTERFACE,/*bDescriptorType: Interface descriptor type*/
 0x00,         /*bInterfaceNumber: Number of Interface*/
 0x00,         /*bAlternateSetting: Alternate setting*/
 0x01,         /*bNumEndpoints*/
 0x03,         /*bInterfaceClass: HID*/
 0x01,         /*bInterfaceSubClass : 1=BOOT, 0=no boot*/
 0x01,         /*nInterfaceProtocol : 0=none, 1=keyboard, 2=mouse*/
 0,            /*iInterface: Index of string descriptor*/
 /******************** Descriptor of Joystick Mouse HID ********************/
```

Change to HID Keyboard Lab

- Change report descriptor to(1):

```
__ALIGN_BEGIN static uint8_t HID_MOUSE_ReportDesc[HID_MOUSE_REPORT_DESC_SIZE]  __ALIGN_END =
{
     0x05    ,//bSize: 0x01, bType: Global, bTag: Usage Page
     0x01    ,//Usage Page(Generic Desktop Controls )
     0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
     0x06    ,//Usage(Keyboard)
     0xA1    ,//bSize: 0x01, bType: Main, bTag: Collection
     0x01    ,//Collection(Application )
     0x85    ,//bSize: 0x01, bType: Global, bTag: Report ID
     0x01    ,//Report ID(0x1 )
     0x05    ,//bSize: 0x01, bType: Global, bTag: Usage Page
     0x07    ,//Usage Page(Keyboard/Keypad )
     0x19    ,//bSize: 0x01, bType: Local, bTag: Usage Minimum
     0xE0    ,//Usage Minimum(0xE0 )
     0x29    ,//bSize: 0x01, bType: Local, bTag: Usage Maximum
     0xE7    ,//Usage Maximum(0xE7 )
     0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
     0x00    ,//Logical Minimum(0x0 )
     0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
     0x01    ,//Logical Maximum(0x1 )
     0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
     0x01    ,//Report Size(0x1 )
```

Change to HID Keyboard Lab

- Change report descriptor to(2):

```
0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
0x08    ,//Report Count(0x8 )
0x81    ,//bSize: 0x01, bType: Main, bTag: Input
0x02    ,//Input(Data, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
0x08    ,//Report Size(0x8 )
0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
0x01    ,//Report Count(0x1 )
0x81    ,//bSize: 0x01, bType: Main, bTag: Input
0x01    ,//Input(Constant, Array, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
0x05    ,//bSize: 0x01, bType: Global, bTag: Usage Page
0x07    ,//Usage Page(Keyboard/Keypad )
0x19    ,//bSize: 0x01, bType: Local, bTag: Usage Minimum
0x00    ,//Usage Minimum(0x0 )
0x29    ,//bSize: 0x01, bType: Local, bTag: Usage Maximum
0x65    ,//Usage Maximum(0x65 )
0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
0x00    ,//Logical Minimum(0x0 )
0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
0x65    ,//Logical Maximum(0x65 )
0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
```

Change to HID Keyboard Lab

- Change report descriptor to(3):

```
0x08    ,//Report Size(0x8 )
0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
0x05    ,//Report Count(0x5 )
0x81    ,//bSize: 0x01, bType: Main, bTag: Input
0x00    ,//Input(Data, Array, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
0xC0    ,//bSize: 0x00, bType: Main, bTag: End Collection
0x05    ,//bSize: 0x01, bType: Global, bTag: Usage Page
0x0C    ,//Usage Page(Consumer )
0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
0x01    ,//Usage(Consumer Control)
0xA1    ,//bSize: 0x01, bType: Main, bTag: Collection
0x01    ,//Collection(Application )
0x85    ,//bSize: 0x01, bType: Global, bTag: Report ID
0x02    ,//Report ID(0x2 )
0x19    ,//bSize: 0x01, bType: Local, bTag: Usage Minimum
0x00    ,//Usage Minimum(0x0 )
0x2A    ,//bSize: 0x02, bType: Local, bTag: Usage Maximum
0x3C,
0x02,//3C    ,//Usage Maximum(0x23C )
0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
0x00    ,//Logical Minimum(0x0 )
0x26    ,//bSize: 0x02, bType: Global, bTag: Logical Maximum
```

Change to HID Keyboard Lab

- Change report descriptor to(4):

```
    0x3C,
    0x02,//3C    ,//Logical Maximum(0x23C )
    0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
    0x01    ,//Report Count(0x1 )
    0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
    0x10    ,//Report Size(0x10 )
    0x81    ,//bSize: 0x01, bType: Main, bTag: Input
    0x00    ,//Input(Data, Array, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
    0xC0    ,//bSize: 0x00, bType: Main, bTag: End Collection
    0x05    ,//bSize: 0x01, bType: Global, bTag: Usage Page
    0x01    ,//Usage Page(Generic Desktop Controls )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x80    ,//Usage(System Control)
    0xA1    ,//bSize: 0x01, bType: Main, bTag: Collection
    0x01    ,//Collection(Application )
    0x85    ,//bSize: 0x01, bType: Global, bTag: Report ID
    0x03    ,//Report ID(0x3 )
    0x19    ,//bSize: 0x01, bType: Local, bTag: Usage Minimum
    0x81    ,//Usage Minimum(0x81 )
    0x29    ,//bSize: 0x01, bType: Local, bTag: Usage Maximum
    0x83    ,//Usage Maximum(0x83 )
    0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
```

Change to HID Keyboard Lab

- Change report descriptor to(5):

```
0x00    ,//Logical Minimum(0x0 )
0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
0x01    ,//Logical Maximum(0x1 )
0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
0x01    ,//Report Size(0x1 )
0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
0x03    ,//Report Count(0x3 )
0x81    ,//bSize: 0x01, bType: Main, bTag: Input
0x02    ,//Input(Data, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
0x05    ,//Report Count(0x5 )
0x81    ,//bSize: 0x01, bType: Main, bTag: Input
0x01    ,//Input(Constant, Array, Absolute, No Wrap, Linear, Preferred State, No Null Position, Bit Field)
0xC0    ,//bSize: 0x00, bType: Main, bTag: End Collection
0x06    ,//bSize: 0x02, bType: Global, bTag: Usage Page
0x01,
0xFF, //01    ,//Usage Page(Undefined )
0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
0x01    ,//Usage(1)
0xA1    ,//bSize: 0x01, bType: Main, bTag: Collection
0x01    ,//Collection(Application )
```

Change to HID Keyboard Lab

- Change report descriptor to(6):

```
    0x85    ,//bSize: 0x01, bType: Global, bTag: Report ID
    0x04    ,//Report ID(0x4 )
    0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
    0x01    ,//Report Count(0x1 )
    0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
    0x08    ,//Report Size(0x8 )
    0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
    0x01    ,//Logical Minimum(0x1 )
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x0A    ,//Logical Maximum(0xA )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x20    ,//Usage(32)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x23    ,//Usage(35)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x4F    ,//Logical Maximum(0x4F )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
```

Change to HID Keyboard Lab

- Change report descriptor to(7):

```
    0x21    ,//Usage(33)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x30    ,//Logical Maximum(0x30 )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x22    ,//Usage(34)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
    0x03    ,//Report Count(0x3 )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x24    ,//Usage(36)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0xC0    ,//bSize: 0x00, bType: Main, bTag: End Collection
    0x06    ,//bSize: 0x02, bType: Global, bTag: Usage Page
    0x01,
    0xFF,//01    ,//Usage Page(Undefined )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
```

Change to HID Keyboard Lab

- Change report descriptor to(8):

```
    0x01    ,//Usage(1)
    0xA1    ,//bSize: 0x01, bType: Main, bTag: Collection
    0x01    ,//Collection(Application )
    0x85    ,//bSize: 0x01, bType: Global, bTag: Report ID
    0x05    ,//Report ID(0x5 )
    0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
    0x01    ,//Report Count(0x1 )
    0x75    ,//bSize: 0x01, bType: Global, bTag: Report Size
    0x08    ,//Report Size(0x8 )
    0x15    ,//bSize: 0x01, bType: Global, bTag: Logical Minimum
    0x01    ,//Logical Minimum(0x1 )
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x0A    ,//Logical Maximum(0xA )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x20    ,//Usage(32)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x23    ,//Usage(35)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
```

Change to HID Keyboard Lab

- Change report descriptor to(9):

```
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x4F    ,//Logical Maximum(0x4F )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x21    ,//Usage(33)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x25    ,//bSize: 0x01, bType: Global, bTag: Logical Maximum
    0x30    ,//Logical Maximum(0x30 )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x22    ,//Usage(34)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0x95    ,//bSize: 0x01, bType: Global, bTag: Report Count
    0x03    ,//Report Count(0x3 )
    0x09    ,//bSize: 0x01, bType: Local, bTag: Usage
    0x24    ,//Usage(36)
    0xB1    ,//bSize: 0x01, bType: Main, bTag: Feature
    0x03    ,//Feature(Constant, Variable, Absolute, No Wrap, Linear, Preferred State, No Null Position, Non
VolatileBit Field)
    0xC0    ,//bSize: 0x00, bType: Main, bTag: End Collection

};
```

Change to HID Keyboard Lab

- In main change buffer size:

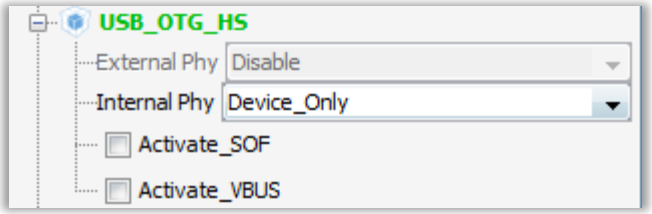
```
/* USER CODE BEGIN PFP */
uint8_t buffer[8];
/* USER CODE END PFP */
```

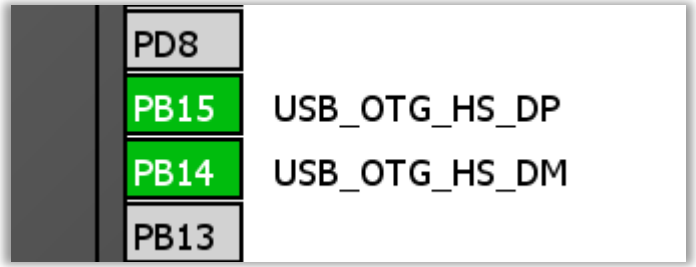
```
 /* USER CODE BEGIN 2 */
  buffer[0]=1;//reportID
  buffer[1]=0;//modifier
  buffer[2]=0;//OEM
  buffer[3]=0x4E;//keycode data - PgDwn
  buffer[4]=0;//keycode data
  buffer[5]=0;//keycode data
  buffer[6]=0;//keycode data
  buffer[7]=0;//keycode data

  /* USER CODE END 2 */
  /* USER CODE BEGIN 3 */
  /* Infinite loop */
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==GPIO_PIN_SET)
    {
      buffer[3]=0x4E;//keycode data - PgDwn press
      USBD_HID_SendReport(&hUsbDeviceHS,buffer,8);
      HAL_Delay(100);
      buffer[3]=0x0;//keycode data - PgDwn release
      USBD_HID_SendReport(&hUsbDeviceHS,buffer,8);
      HAL_Delay(100);

     }
  }
```

USB DFU Device lab

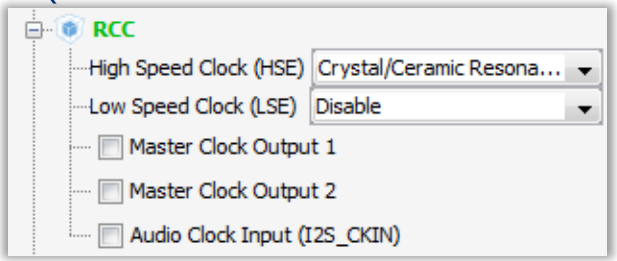
USB DFU Device lab

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)

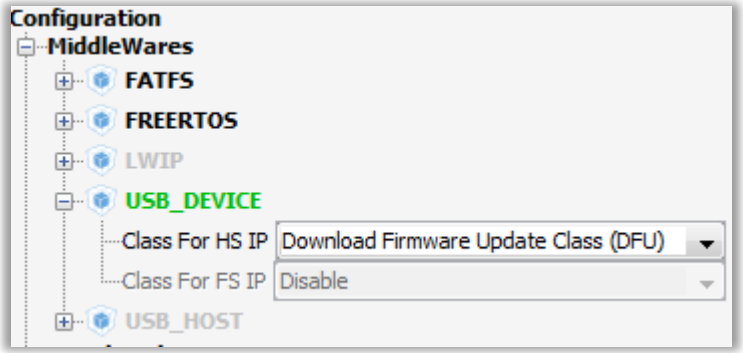




- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)

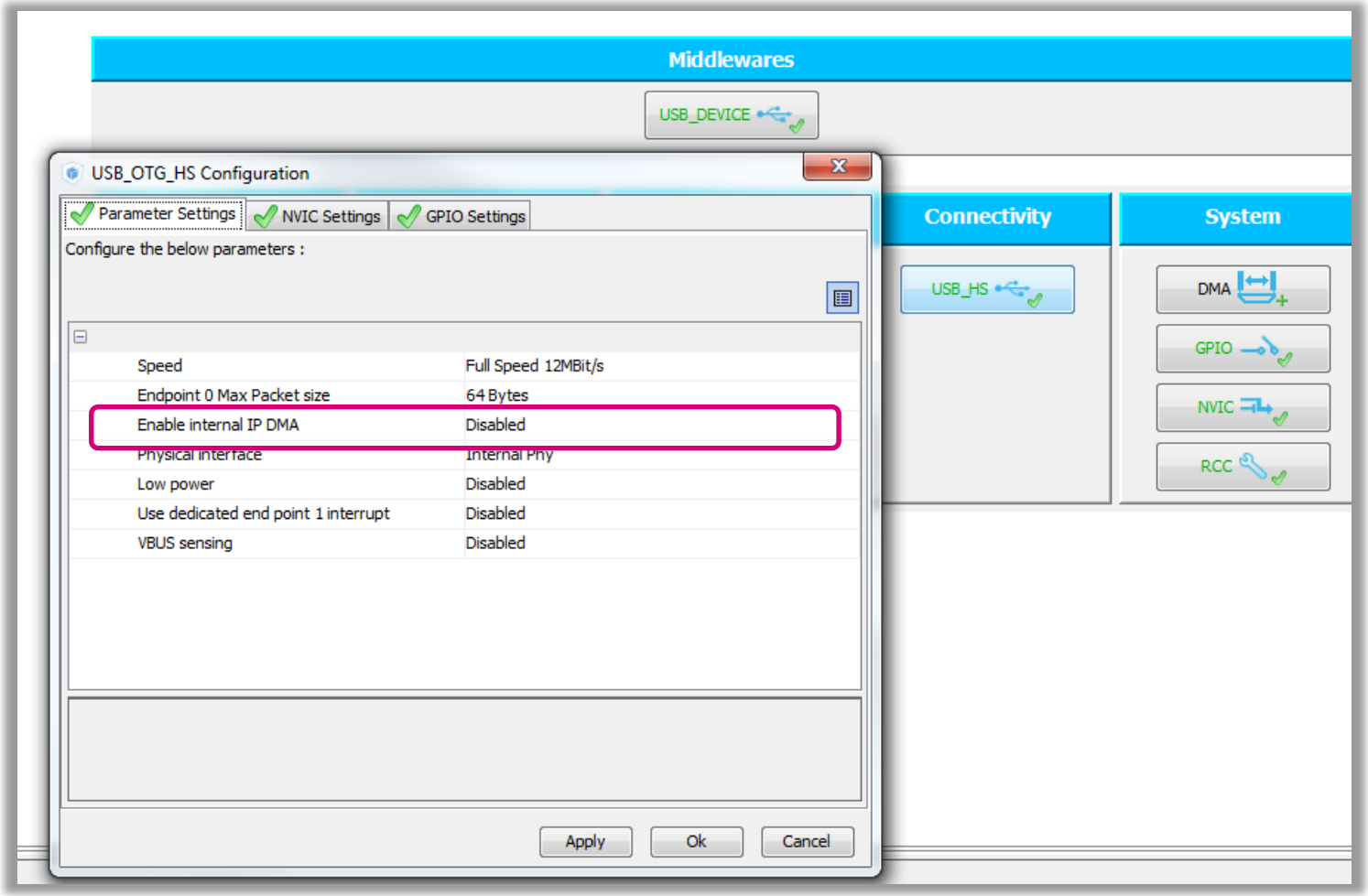


- Select HID class in MiddleWares



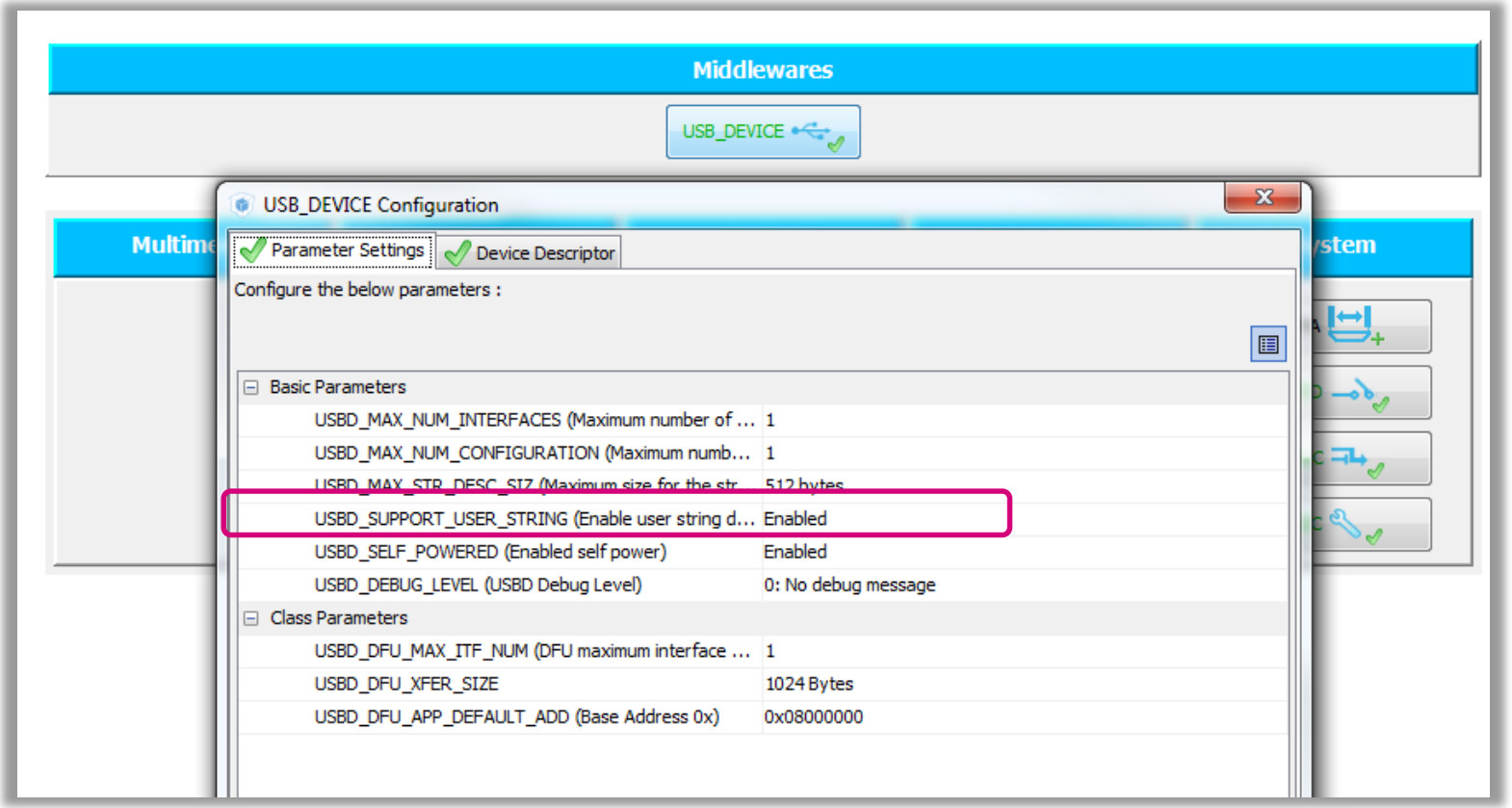
USB DFU Device lab

- In Configuration tab select USB_HS in Connectivity
- Disable option use internal DMA
- Button OK



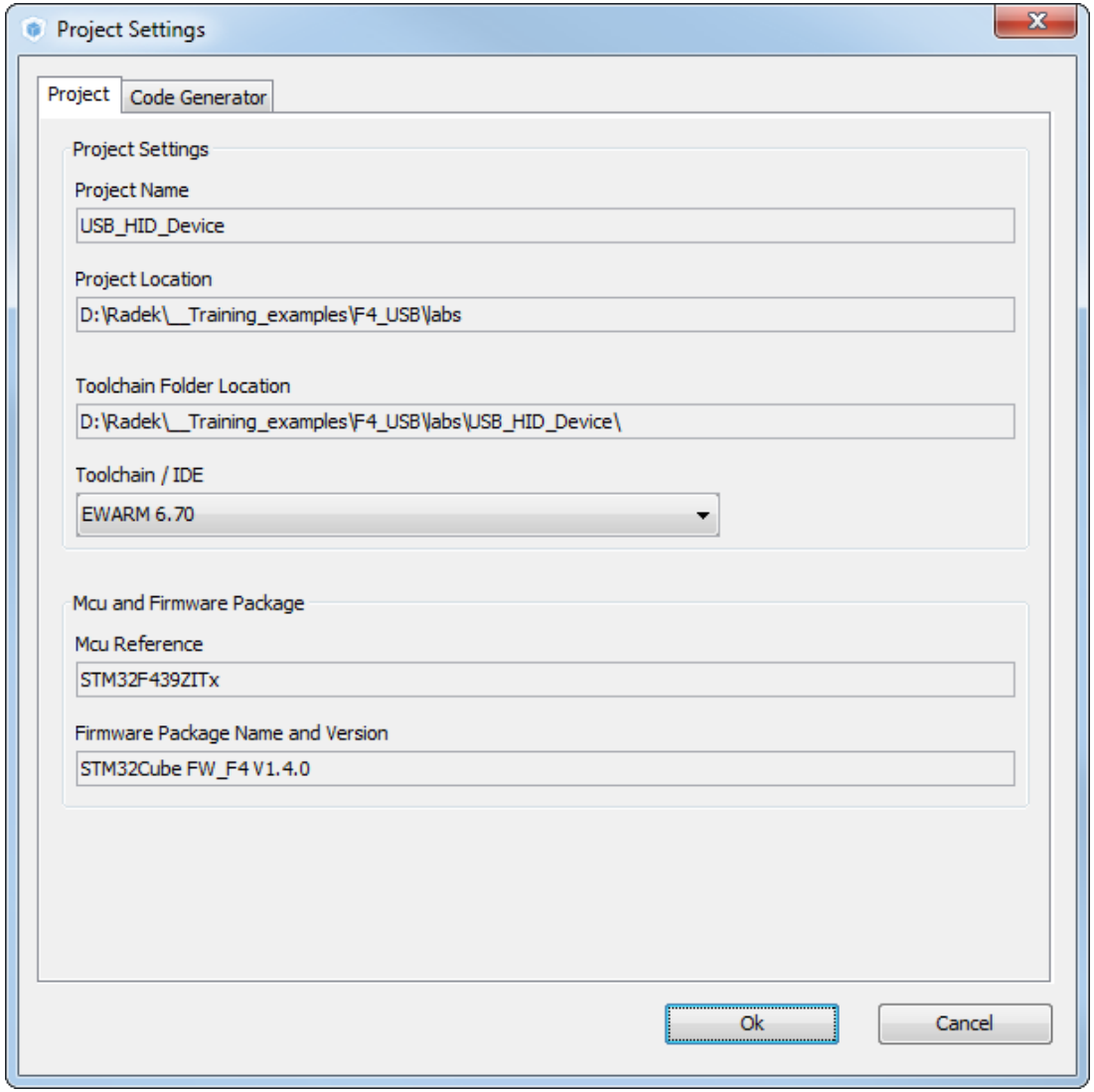
USB DFU Device lab

- In Configuration tab select USB_DEVICE in Middleware's
- Enable user string descriptor support
- Button OK



USB DFU Device lab

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code
- If you have KEIL change HEAP size in startup file



USB DFU Device lab

- CubeMX create for us file usbd_dfu.c
- This file handling reading and writing into memory
- MEM_If_Init_HS
  - Initialize programing, called on programing start
- MEM_If_DeInit_HS
  - Deinitialize programing, called on programing end
- MEM_If_Erase_HS
  - Erase selected part of memory
- MEM_If_Write_HS
  - Write into selected memory
- MEM_If_Read_HS
  - Read from selected memory
- MEM_If_GetStatus_HS
  - Return state of programing
  - Busy or ready

USB DFU Device lab

- We need to modify the usbd_dfu_it.c file
- We ned to change the string description of memory:

```
__ALIGN_BEGIN USBD_DFU_MediaTypeDef USBD_DFU_fops_HS __ALIGN_END =
{
    (uint8_t *) "@Internal Flash   /0x20020000/1*016Kg",
    MEM_If_Init_HS,
    MEM_If_DeInit_HS,
    MEM_If_Erase_HS,
    MEM_If_Write_HS,
    MEM_If_Read_HS,
    MEM_If_GetStatus_HS,
};
```

- Now the DFU tool will be able recognize that we can program RAM memory on address 0x20020000 and size of this memory is 16kB

USB DFU Device lab

- MEM_If_Init_HS and MEM_If_DeInit_HS function can be empty because we want program RAM which it to necessary to lock or unlock

```
uint16_t MEM_If_Init_HS(void)
{
  /* USER CODE BEGIN 7 */
  return (USBD_OK);
  /* USER CODE END 7 */
}
```

```
uint16_t MEM_If_DeInit_HS(void)
{
  /* USER CODE BEGIN 8 */
  return (USBD_OK);
  /* USER CODE END 8 */
}
```

USB DFU Device lab

- MEM_If_Erase_HS function simply set our RAM memory space to zero

```
uint16_t MEM_If_Erase_HS(uint32_t Add)
{
  /* USER CODE BEGIN 9 */
  uint32_t i;
  for(i=0;i<0x3FFF;i=i+4){
    *(uint32_t*)(0x20020000+i)=0;
  }
  return (USBD_OK);
  /* USER CODE END 9 */
}
```

USB DFU Device lab

- MEM_If_Write_HS program the source buffer to destination buffer

```
uint16_t MEM_If_Write_HS(uint8_t *src, uint8_t *dest, uint32_t Len)
{
  /* USER CODE BEGIN 10 */
  uint32_t i = 0;
  for(i = 0; i < Len; i+=4)
  {
     *(uint32_t*)(dest+i)=*(uint32_t*)(src+i);
    /* Check the written value */
    if(*(uint32_t *)(src + i) != *(uint32_t*)(dest+i))
    {
       return USBD_FAIL;
    }
  }
  return (USBD_OK);
  /* USER CODE END 10 */
}
```

USB DFU Device lab

- MEM_If_Read_HS read data from source address and copy it into destination address

```
uint8_t *MEM_If_Read_HS (uint8_t *src, uint8_t *dest, uint32_t Len)
{
  /* Return a valid address to avoid HardFault */
  /* USER CODE BEGIN 11 */
  uint32_t i = 0;
  uint8_t *psrc = src;

  for(i = 0; i < Len; i++)
  {
    dest[i] = *psrc++;
  }
  /* Return a valid address to avoid HardFault */
  return (uint8_t*)(dest);
  /* USER CODE END 11 */
}
```

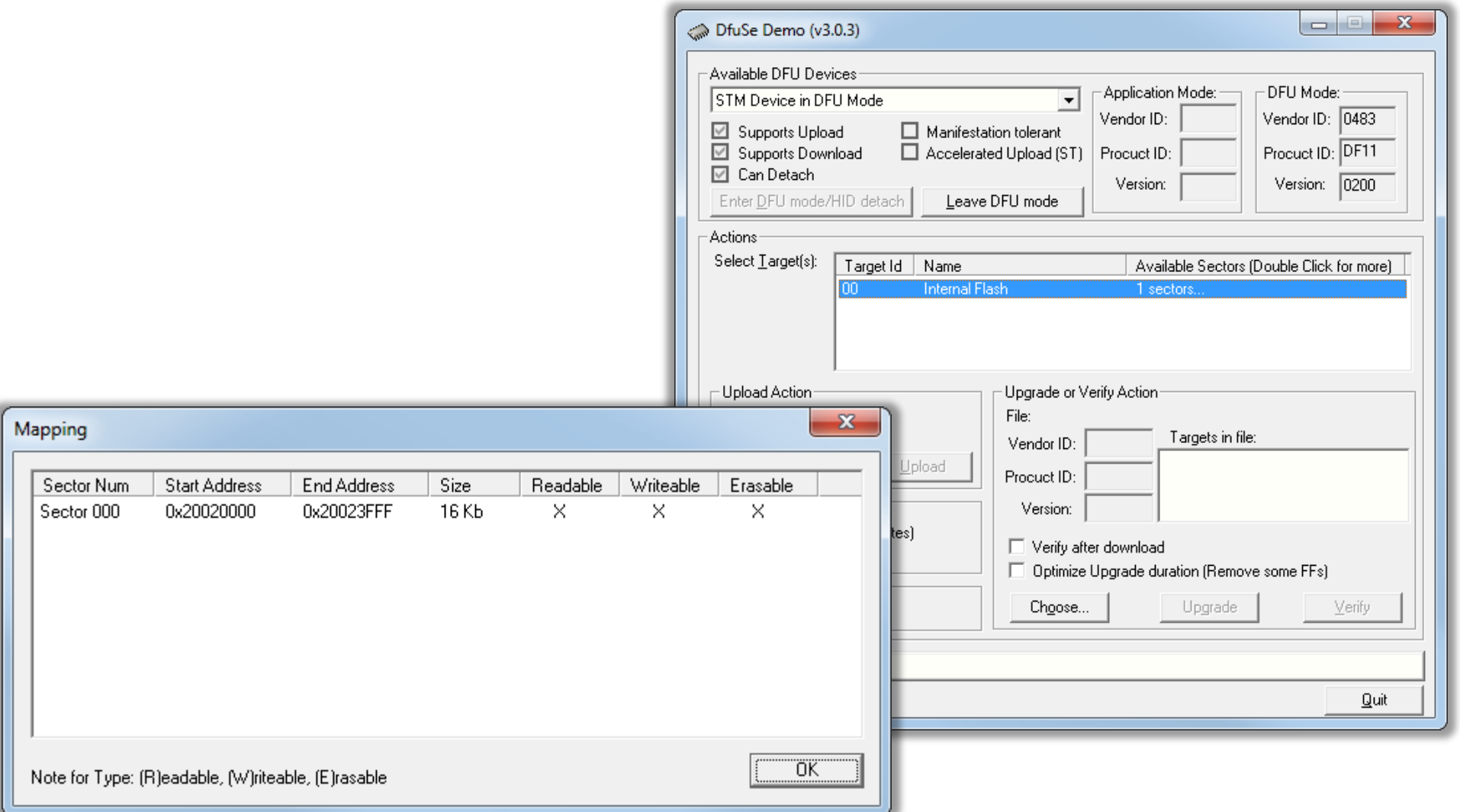
USB DFU Device lab

- MEM_If_GetStatus_HS read information how long take programing and erasing

```
uint16_t MEM_If_GetStatus_HS (uint32_t Add, uint8_t Cmd, uint8_t *buffer)
{
  /* USER CODE BEGIN 12 */
#define MEMORY_ERASE_TIME    (uint16_t)50
#define MEMORY_PROGRAM_TIME  (uint16_t)50
  switch (Cmd)
  {
  case DFU_MEDIA_PROGRAM:
    buffer[1] = (uint8_t)MEMORY_PROGRAM_TIME;
    buffer[2] = (uint8_t)(MEMORY_PROGRAM_TIME << 8);
    buffer[3] = 0;
    break;
  case DFU_MEDIA_ERASE:
  default:
    buffer[1] = (uint8_t)MEMORY_ERASE_TIME;
    buffer[2] = (uint8_t)(MEMORY_ERASE_TIME << 8);
    buffer[3] = 0;
    break;
  }
  return  (USBD_OK);
  /* USER CODE END 12 */
}
```

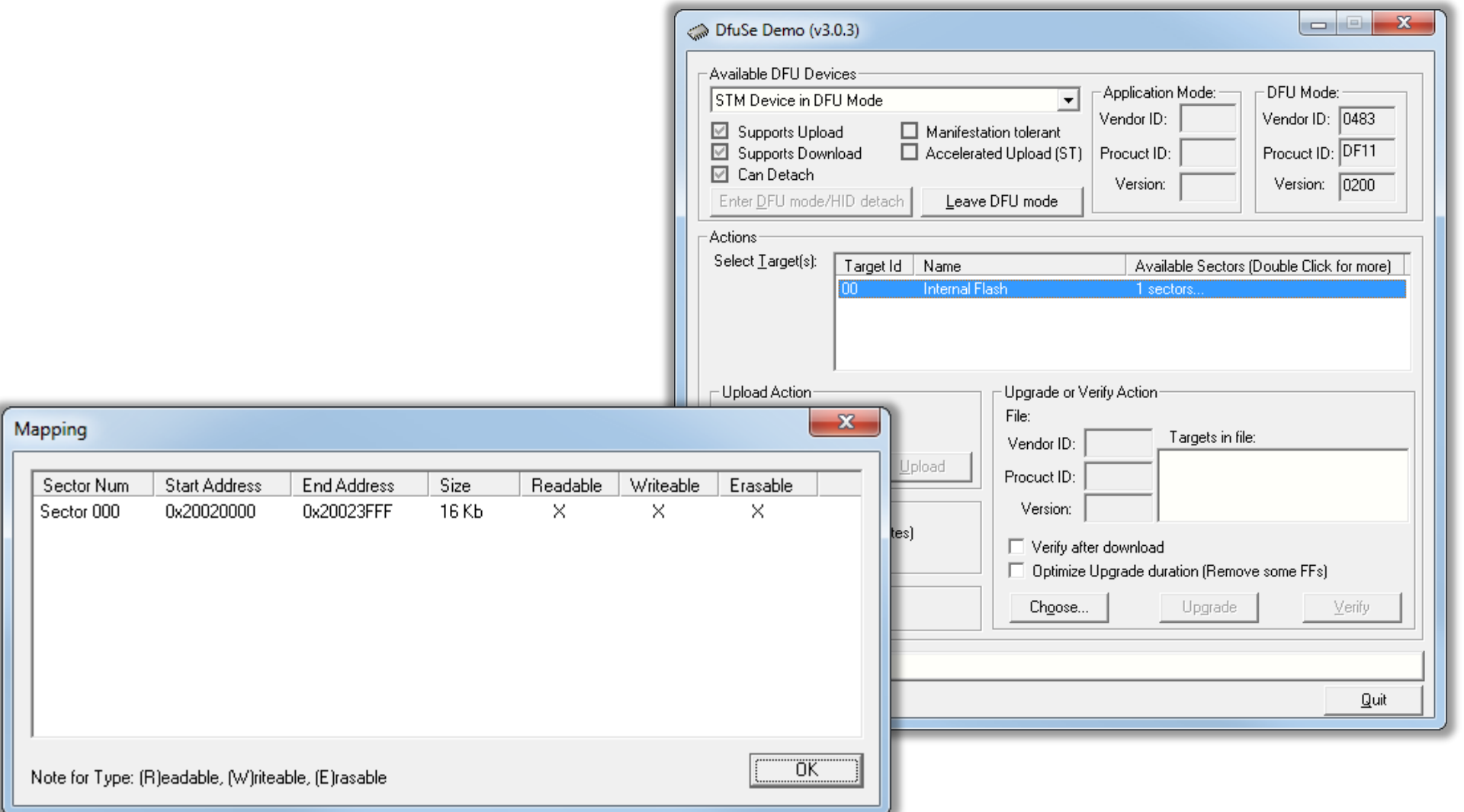
USB DFU Device lab

- We can use DfuSe Demo to try program the selected memory



USB DFU Device lab

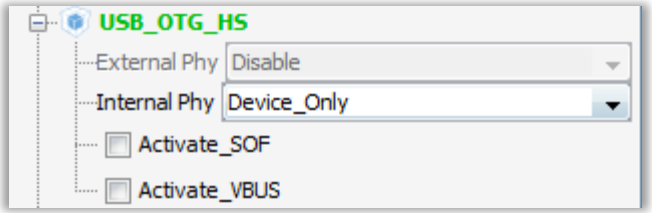
- We can use DfuSe Demo to try program the selected memory

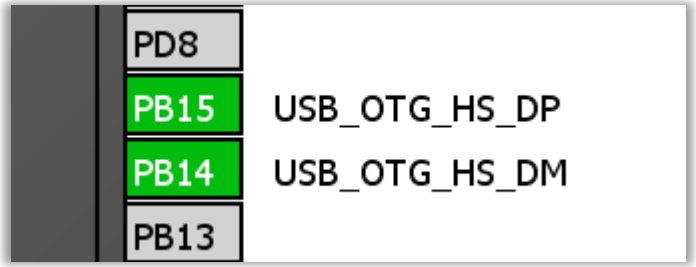


USB MSP Device lab

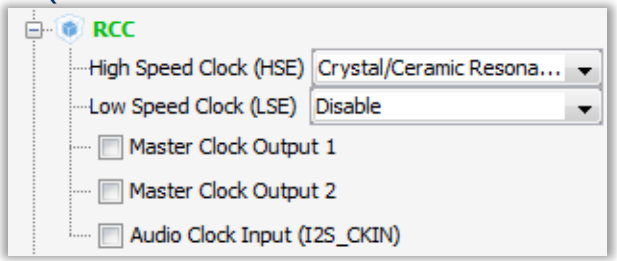
USB DFU Device lab

- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F429/439 > LQFP144 > STM32F439ZITx
- Select USB HS OTG internal PHY(FS)





- Select HSE clock
  - (HSI cannot be used and STM32F4 have no clock synchronization)



- Select MSP class in MiddleWares

USB DFU Device lab

- CubeMX create for us file usbd_dfu.c
- This file handling reading and writing into memory
- XXXX

- STORAGE_IsWriteProtected_FS
  - Return state of programing
  - Busy or ready

- STORAGE_IsReady_FS
  - Read from selected memory

USB DFU Device lab

- MEM_If_Read_HS read data from source address and copy it into destination address

```
#define STORAGE_LUN_NBR                  1
#define STORAGE_BLK_NBR                  200//1600
#define STORAGE_BLK_SIZ                  512//64/*0x200*/
```

```
const int8_t  STORAGE_Inquirydata_FS[] = {//36

  /* LUN 0 */
  0x00,
  0x80,
  0x02,
  0x02,
  (STANDARD_INQUIRY_DATA_LEN - 5),
  0x00,
  0x00,
  0x00,
  'S', 'T', 'M', ' ', ' ', ' ', ' ', ' ', /* Manufacturer : 8 bytes */
  'P', 'r', 'o', 'd', 'u', 'c', 't', ' ', /* Product      : 16 Bytes */
  ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
  '0', '.', '0' ,'1',                     /* Version      : 4 Bytes */
};
```

USB DFU Device lab

- MEM_If_Read_HS read data from source address and copy it into destination address

```
/* USER CODE BEGIN 3 */
 *block_num  = STORAGE_BLK_NBR;
 *block_size = STORAGE_BLK_SIZ;
 return (USBD_OK);
 /* USER CODE END 3 */
```

```
/* USER CODE BEGIN 4 */
 return 0;
/* USER CODE END 4 */
```

```
/* USER CODE BEGIN 5 */
 return (0);
 /* USER CODE END 5 */
```

```
/* USER CODE BEGIN 6 */
 memcpy(buf,&buffer[blk_addr*blk_len*STORAGE_BLK_SIZ],blk_len*STORAGE_BLK_SIZ);
 return (USBD_OK);
 /* USER CODE END 6 */
```

```
/* USER CODE BEGIN 7 */
 memcpy(&buffer[blk_addr*blk_len*STORAGE_BLK_SIZ],buf,blk_len*STORAGE_BLK_SIZ);
 return (USBD_OK);
 /* USER CODE END 7 */
```