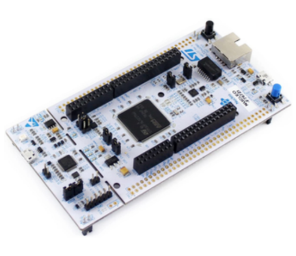
How to implement a dual CDC ACM USB device using the ST classic library

FBL
ST Employee

on 2024-08-29 5:00 AM - edited on 2025-11-25 6:27 AM by ADMIN Laurids_PETERSEN

Summary

This article presents a step-by-step tutorial on how to develop a USB device with dual CDC ACM in the STM32F7 microcontroller using the classic USB library. The tutorial is based on NUCLEO-F767 and can be easily tailored to any other STM32.



Introduction

If you are developing a USB device composite application for the STM32, this article is for you! Here we show you how to implement the classical USB device middleware to use more than one class in the same application.

For this tutorial, we use the F767 Nucleo board, which has a USB connector to open two CDC (Communications Device Class) class through the USB communication. The CDC is used to open a virtual COM port communication. This board has an STM32F767 microcontroller, and for further details about the board, refer to its user manual. The steps shown here can be easily adapted to any other STM32. The IAR IDE, the STM32CubeF7 were used to build this tutorial.

To get access to the code developed for this article, download the attached project F767_DualCDC.7z that you can find at the bottom of this article.

For a detailed explanation regarding the classic USB library, refer to the ST Wiki and the STM32 USB training:

ST Wiki - Introduction to USB with STM32

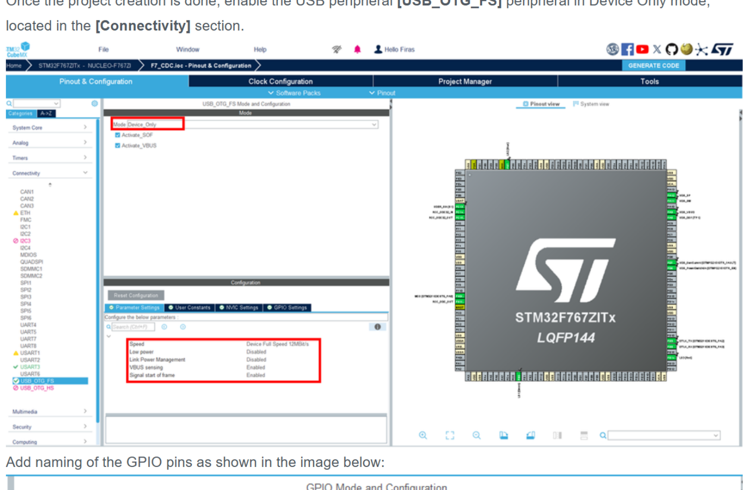
MOOC - STM32 USB Training

1. Development

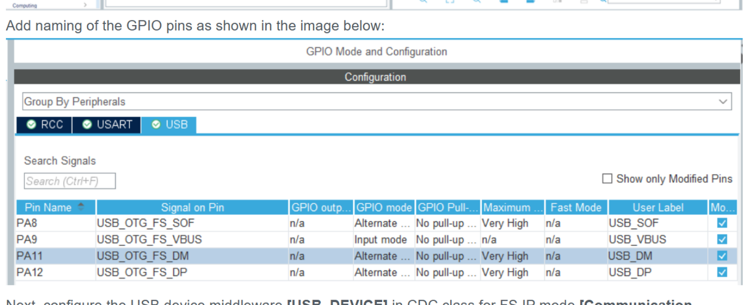
1.1. Project configuration

Start by creating a new project using the STM32CubeIDE by clicking **[File -> New -> STM32 Project]**. Use the **[Board Selector]** tab and select the **[NUCLEO-F767]**.

Once the project creation is done, enable the USB peripheral **[USB_OTG_FS]** peripheral in Device Only mode, located in the **[Connectivity]** section.



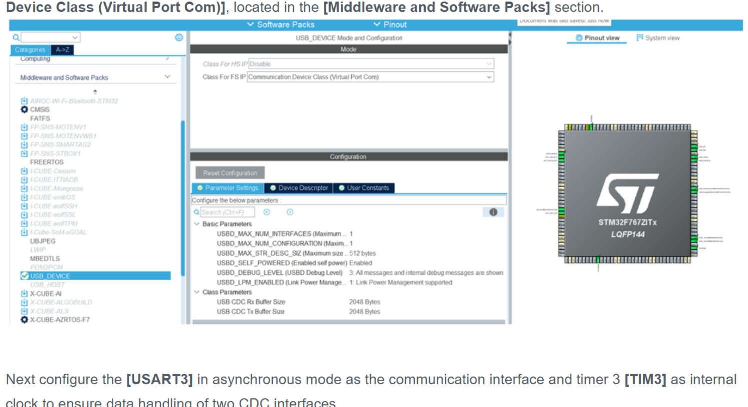
Add naming of the GPIO pins as shown in the image below:



GPIO Mode and Configuration

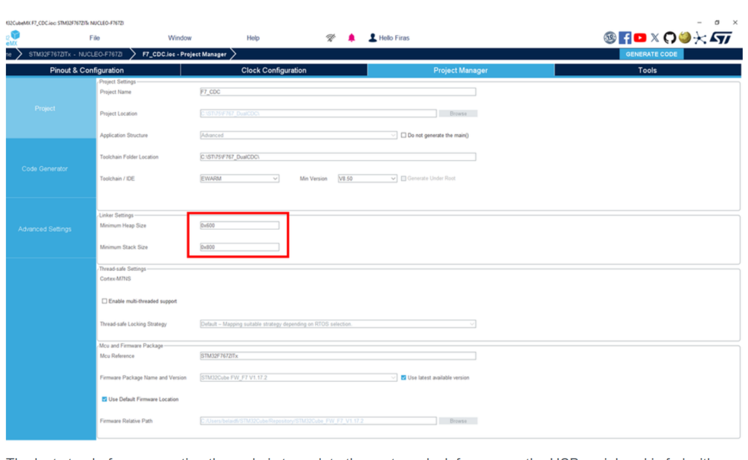
Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO Pull-...	Maximum ...	Fast Mode	User Label	Mo...
PA8	USB_OTG_FS_SOF	n/a	Alternate ...	No pull-up ...	Very High	n/a	USB_SOF	✓
PA9	USB_OTG_FS_VBUS	n/a	Input mode	No pull-up ...	n/a	n/a	USB_VBUS	✓
PA11	USB_OTG_FS_DM	n/a	Alternate ...	No pull-up ...	Very High	n/a	USB_DM	✓
PA12	USB_OTG_FS_DP	n/a	Alternate ...	No pull-up ...	Very High	n/a	USB_DP	✓

Next, configure the USB device middleware **[USB_DEVICE]** in CDC class for FS IP mode **[Communication Device Class (Virtual Port Com)]**, located in the **[Middleware and Software Packs]** section.

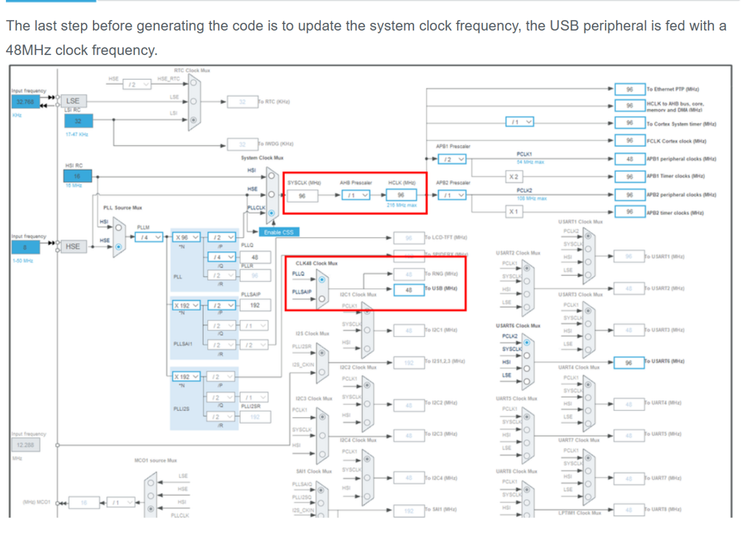


Next configure the **[USART3]** in asynchronous mode as the communication interface and timer 3 **[TIM3]** as internal clock to ensure data handling of two CDC interfaces.

After that, increase the amount of heap and stack size of the project as indicated in the image below, this action is done in the **[Project Manager]** tab. Then, under [**Code Generator**], copy all used libraries into the project folder.



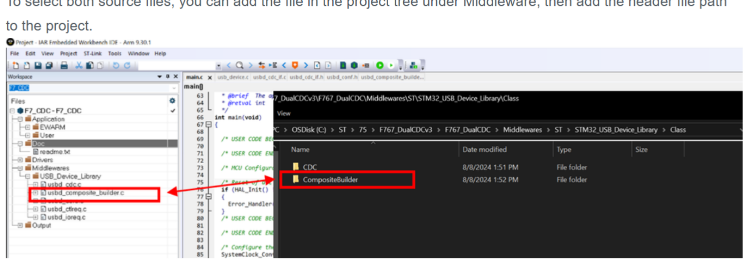
The last step before generating the code is to update the system clock frequency, the USB peripheral is fed with a 48MHz clock frequency.



Doing that, you have all set to proceed with the code generation.

1.2. Project tree configuration

Copy manually files and folders of the composite builder class from **STM32Cube\Repository\STM32Cube_FW_F7_V1.17.2\Middlewares\ST\STM32_USB_Device_Library\Class\CompositeBuilder** to the folder of your project: ***YourProjectName\Middlewares\ST\STM32_USB_Device_Library\Class\CompositeBuilder***.
To select both source files, you can add the file in the project tree under Middleware, then add the header file path to the project.



After doing that, we add the header file under project **[Options->C/C++ Compiler->Preprocessor->Additional Include directories]**

Next step, to facilitate the use of this article, you can simply replace the existent **USB_DEVICE** folder including **App** and **Target** with the one in attached project.

Scroll down to the definition of MX_USB_DEVICE_Init() function and you can find the following code to initialize the USB peripheral stack and register all the used classes:

```

  /* Init Device Library, add supported class and start the library. */
  if (USBD_Init(&hUsbDeviceFS, &COMPOSITE_Desc, DEVICE_FS) != USBD_OK)
  {
    Error_Handler();
  }

  /* Register CDC class first instance */
  USBD_RegisterClassComposite(&hUsbDeviceFS, USBD_CDC_CLASS, CLASS_TYPE_CDC, CDC_EpAdd_Inst1);

  /* Register CDC class second instance */
  USBD_RegisterClassComposite(&hUsbDeviceFS, USBD_CDC_CLASS, CLASS_TYPE_CDC, CDC_EpAdd_Inst2);

  /* Add CDC Interface Class */
  if (USBD_CMPSIT_SetClassID(&hUsbDeviceFS, CLASS_TYPE_CDC, 0) != 0xFF)
  {
    USBD_CDC_RegisterInterface(&hUsbDeviceFS, &USBD_Interface_fops_FS);
  }

  /* Add CDC Interface Class */
  if (USBD_CMPSIT_SetClassID(&hUsbDeviceFS, CLASS_TYPE_CDC, 1) != 0xFF)
  {
    USBD_CDC_RegisterInterface(&hUsbDeviceFS, &USBD_Interface_fops_FS);
  }
  if (USBD_Start(&hUsbDeviceFS) != USBD_OK)
  {
    Error_Handler();
  }
```

The variables that are declared in the code section below, are used to store the addresses of the classes (CDC_EpAdd_Inst1 and CDC_EpAdd_Inst2). Each CDC instance uses three endpoints (CMD, IN, OUT).

```
/* USER CODE BEGIN 0 */

uint8_t CDC_EpAdd_Inst1[3] = {CDC_IN_EP, CDC_OUT_EP, CDC_CMD_EP}; /* CDC Endpoint Adress First Insta
uint8_t CDC_EpAdd_Inst2[3] = {SECOND_CDC_IN_EP, SECOND_CDC_OUT_EP, SECOND_CDC_CMD_EP}; /* CDC Endpoi

/* USER CODE END 0 */
```

Continuing in the file **usbd_conf.h**, you can also find defines as presented below. The created configuration was made to match the requirements for CDC classes.

```
/* Activate the IAD option */
#define USBD_COMPOSITE_USE_IAD                      1U

/* Activate the composite builder */
#define USE_USBD_COMPOSITE

/* Activate CDC class in composite builder */
#define USBD_CMPSIT_ACTIVATE_CDC                    1U

/* The definition of endpoint numbers must respect the order of classes instantiation*/
#define CDC_OUT_EP                          0x01U  /* EP1 for CDC data OUT First Instance */
#define CDC_IN_EP                           0x81U  /* EP1 for CDC data IN First Instance */
#define CDC_CMD_EP                          0x82U  /* EP2 for CDC commands First Instance */

#define SECOND_CDC_OUT_EP                   0x02  /* EP2 for CDC data OUT Second Instance */
#define SECOND_CDC_IN_EP                    0x83  /* EP4 for CDC data IN Second Instance */
#define SECOND_CDC_CMD_EP                   0x84  /* EP5 for CDC commands Second Instance */

/* Common Config */
#define USBD_MAX_NUM_INTERFACES             6U
#define USBD_MAX_NUM_CONFIGURATION          1U
#define USBD_MAX_STR_DESC_SIZ               0x100U
#define USBD_SUPPORT_USER_STRING            1U
#define USBD_SUPPORT_USER_STRING_DESC       1U
#define USBD_SELF_POWERED                   1U
#define USBD_DEBUG_LEVEL                    0U
#define USBD_CONFIG_STR_DESC_IDX                    4U
#define USBD_CONFIG_BMATTRIBUTES                    0xC0U
#define USBD_CONFIG_MAXPOWER                        0x32U
```

To use the CDC class, these settings are enough. Otherwise, when other classes are needed, you can edit the files following your application requirements.

Within **usbd_conf.c**, we need to link the HAL PCD drivers with the USB stack.

```
USBD_StatusTypeDef USBD_LL_Init(USBD_HandleTypeDef *pdev)
{
         /* Init USB Ip. */
  if (pdev->id == DEVICE_FS) {
  /* Link the driver to the stack. */

  hpcd_USB_OTG_FS.pData = pdev;
  pdev->pData = &hpcd_USB_OTG_FS;
  hpcd_USB_OTG_FS.Instance = USB_OTG_FS;
  hpcd_USB_OTG_FS.Init.dev_endpoints = 6;
  hpcd_USB_OTG_FS.Init.speed = PCD_SPEED_FULL;
  hpcd_USB_OTG_FS.Init.dma_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.phy_itface = PCD_PHY_EMBEDDED;
  hpcd_USB_OTG_FS.Init.Sof_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.low_power_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.lpm_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.vbus_sensing_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.use_dedicated_ep1 = DISABLE;

  if (HAL_PCD_Init(&hpcd_USB_OTG_FS) != HAL_OK)
  {
    Error_Handler( );
  }

  (void)HAL_PCDEx_SetRxFiFo(&hpcd_USB_OTG_FS, 0x80);
  (void)HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_FS, 0, 0x15);
  (void)HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_FS, 1, 0x15);
  (void)HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_FS, 2, 0x30);
  (void)HAL_PCDEx_SetTxFiFo(&hpcd_USB_OTG_FS, 3, 0x15);
  (void)HAL_PCDEx_SetTxFiFo(&hpcd_USB_O
```

View more

Now, we can move forward to the **usbd_desc.c** file. In this file, rename the USBD_DescriptorsTypeDef to Class_Desc. The product ID used for the F7 series is **0xC9**.

Then proceed opening the usbd_desc.c file and edit the following defines for your application:

```
/* Private define ------------------------------------------------------------*/
#define USBD_VID                         0x0483
#define USBD_PID_FS                      0xC9
#define USBD_LANGID_STRING               0x409
#define USBD_MANUFACTURER_STRING         "STMicroelectronics"
#define USBD_PRODUCT_HS_STRING           "Composite DualCDC in HS Mode"
#define USBD_PRODUCT_FS_STRING           "Composite DualCDC in FS Mode"
#define USBD_CONFIGURATION_STRING_HS     "Composite DualCDC Config in HS"
#define USBD_INTERFACE_STRING_HS         "Composite DualCDC Interface i HS"
#define USBD_CONFIGURATION_STRING_FS     "Composite DualCDC Config in FS"
#define USBD_INTERFACE_STRING_FS         "Composite DualCDC Interface in FS"
```

You can rename all the functions available in the **usbd_cdc_if.c** from to the most convenient name for your application. This file was created in this way to allow you to handle more than one CDC.

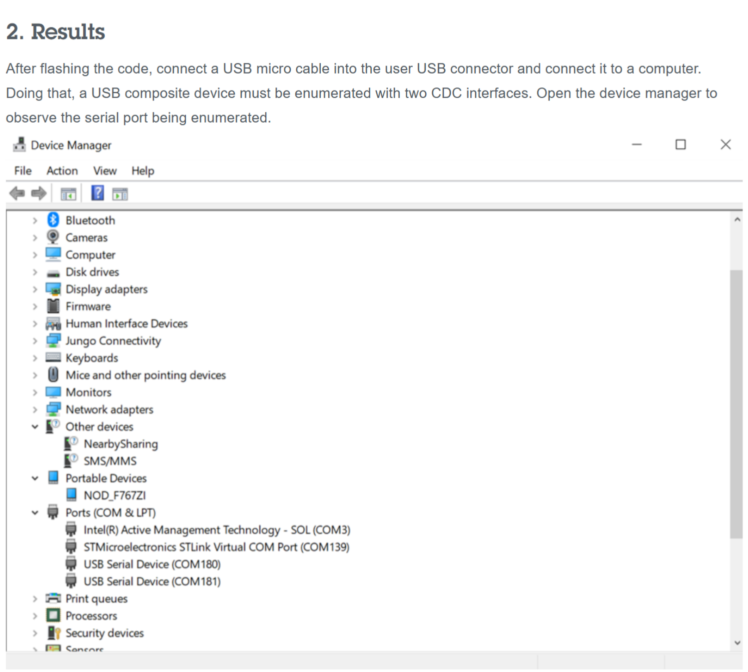
Then, open the **main.c** file to start creating the application. The first step in this file is to add the following includes:

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "usbd_core.h"
#include "usbd_cdc.h"
#include "usbd_cdc_if.h"
#include "usbd_desc.h"
#include "usbd_composite_builder.h"
/* USER CODE END Includes */
```

Here all the code implementation is done! You can build and flash the code into your device and try it!

2. Results

After flashing the code, connect a USB micro cable into the user USB connector and connect it to a computer. Doing that, a USB composite device must be enumerated with two CDC interfaces. Open the device manager to observe the serial port being enumerated.



Conclusion

And that concludes our article. Now, you have the needed knowledge to implement a composite class in the STM32 using our legacy USB middleware. Here we presented the step-by-step to construct dual CDC, but the steps for opening other classes should be similar.

For details on how to use/implement the other device classes, please refer to GitHub - STMicroelectronics/STM32CubeH7 at dev/usb/composite branch. Within that you may find a large set of examples showing the usage of the most classes available and composite examples.

Related links

Here are some useful links that contain material, which was used to build this article and can be helpful in your developments.

- GitHub - STMicroelectronics/stm32_mw_usb_device
- ST Wiki - Introduction to USB with STM32
- MOOC - STM32 USB Training
- NUCLEO-F767ZI - STM32 Nucleo-144 development board with STM32F767ZI MCU, supports Arduino, ST Zio and morpho connectivity - STMicroelectronics
- STM32 Nucleo-144 boards (MB1137) - User manual

USB

F767_DualCDC.7z

2 Kudos

Version history

Last update:
2025-11-25 6:27 AM

Updated by:
ADMIN Laurids_PETERSEN

Contributors

FBL