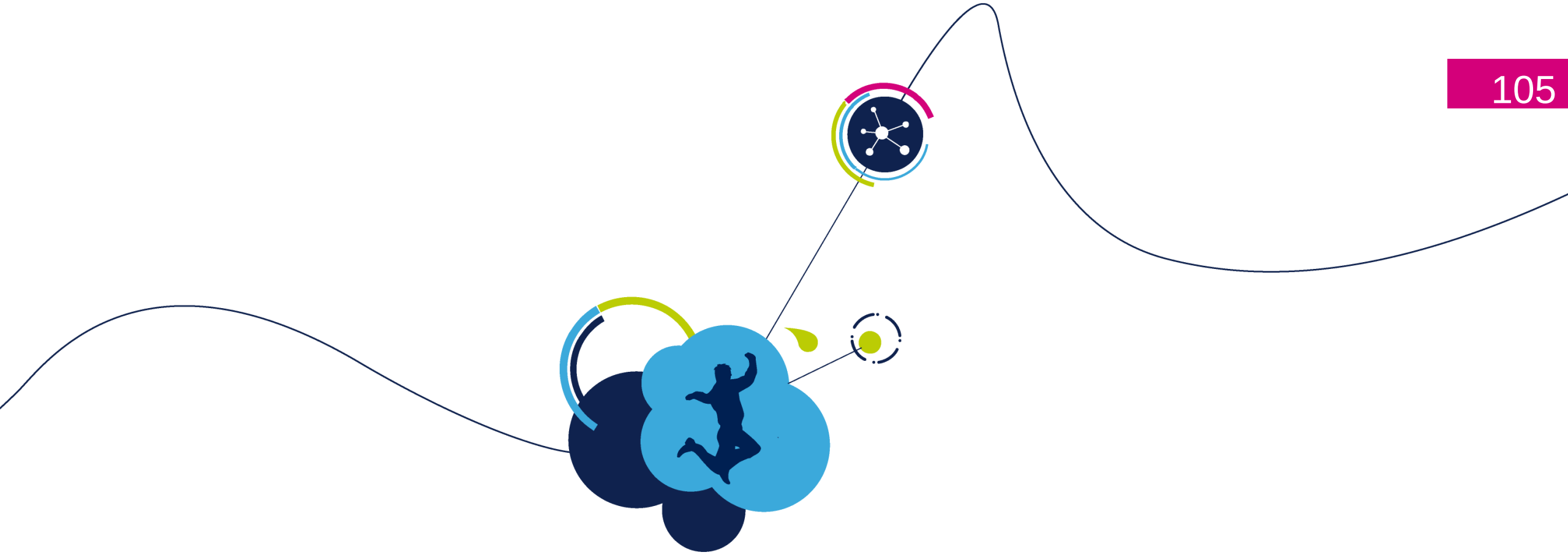


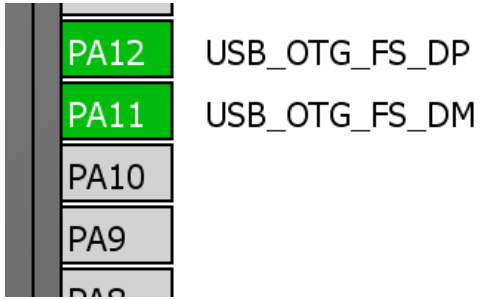
USB Download Firmware Update Device lab

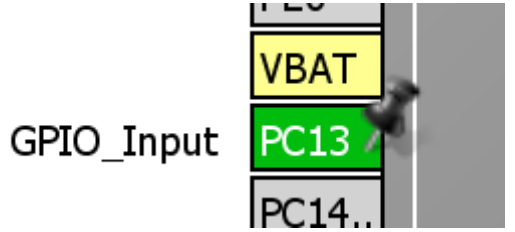
USB DFU Device lab

- Class for firmware update through USB
- Use STtub30 – driver from ST
- In the example internal SRAM is used for as storage for USB mass storage
- Read Out Protection may be active with user DFU bootloader

USB DFU Device lab

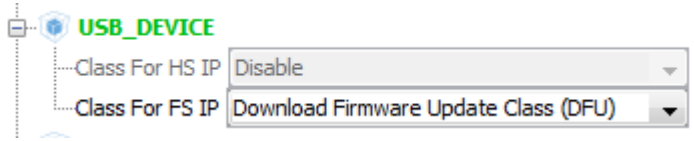
- Create project in CubeMX
- Menu > File > New Project
  - Select STM32F4 > STM32F446 > LQFP144 > STM32F446ZETx
- Select USB FS OTG in device mode
- Select HSE clock
  - (Bypass HSE from STlink)
- Configure PC13 as input – key button





USB DFU Device lab

- Configure GPIOs connected to LEDs as GPIO output – PB0, PB7 and PB14
- Select DFU class in MiddleWares



- Configure RCC clocks
  - Set 8 MHz HSE as PLL input
  - Set HCLK frequency 168 MHz
  - PLL parameters will be computed automatically

USB DFU Device lab

- In Configuration -> USB_DEVICE two parameters needs to be changed
  - User application start on address 0x08008000 – Sector 2

```
USBD_DFU_APP_DEFAULT_ADD = 0x08008000
```

- and USBD_DFU_MEDIA Interface
  - This line is part of descriptor showing host possible memory location inside MCU
  - Meaning of symbols inside can be

```
#define FLASH_DESC_STR      "@Internal Flash /0x08000000/02*016Ka,02*016Kg,01*064Kg,03*128Kg"
```

USB DFU Device lab

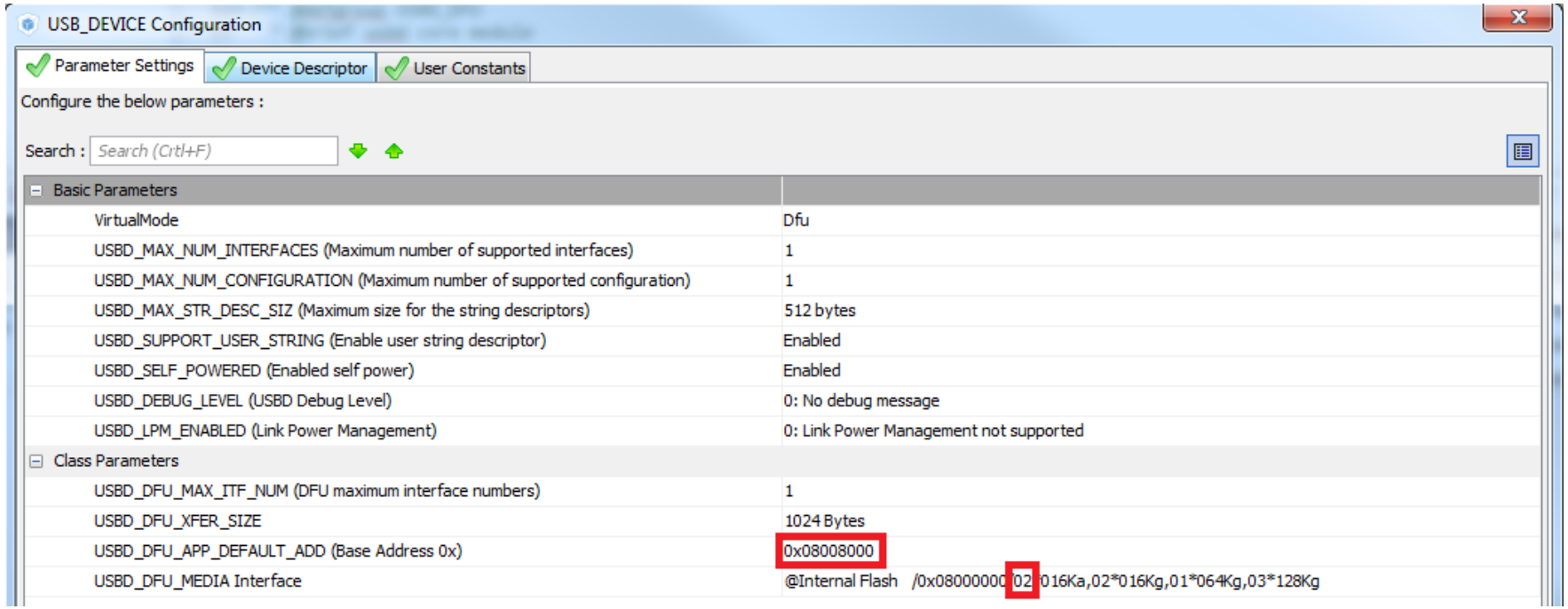
- Decoding of the right mapping for the selected device
  - @: To detect that this is a special mapping descriptor (to avoid decoding standard descriptor)
  - /: for separator between zones
  - Maximum 8 digits per address starting by “0x”
  - /: for separator between zones
  - Maximum of 2 digits for the number of sectors
  - *: For separator between number of sectors and sector size
  - Maximum 3 digits for sector size between 0 and 999
  - 1 digit for the sector size multiplier. Valid entries are: B (byte), K (Kilo), M (Mega)
  - …

USB DFU Device lab

- …
- 1 digit for the sector type as follows:
  - a (0x41): Readable
  - b (0x42): Erasable
  - c (0x43): Readable and Erasabled (0x44): Writeable
  - e (0x45): Readable and Writeable
  - f (0x46): Erasable and Writeable
  - g (0x47): Readable, Erasable and Writeable

- Complete description of this descriptor part is inside UM0424, page 72

USB DFU Device lab



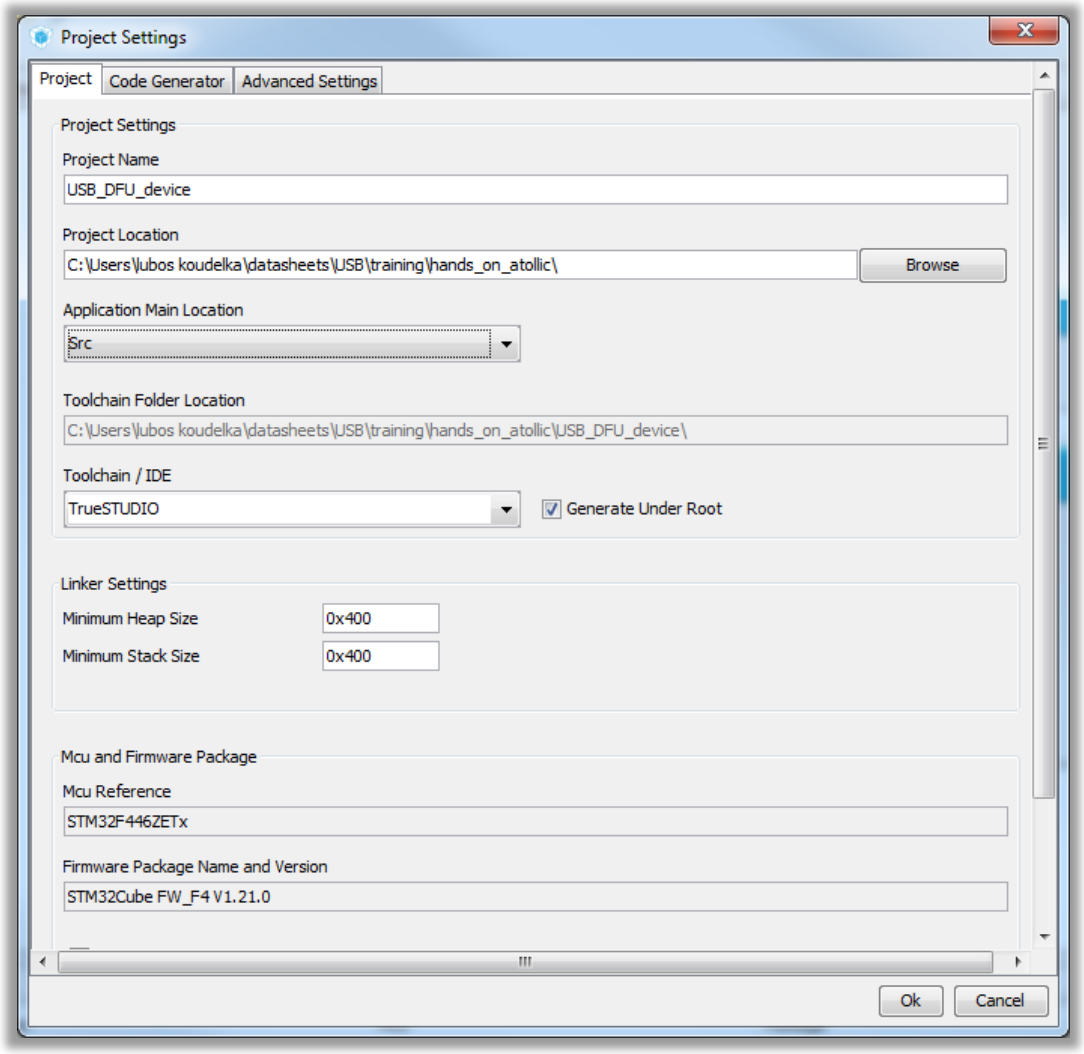
USB DFU Device lab

- In Configuration -> System -> NVIC decrease USB On The Go FS global interrupt priority
- Systick delay is used in USB interrupt routine – Systick priority needs to be higher than USB

Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
USB On The Go FS global interrupt	☑	1	0
FPU global interrupt	☐	0	0

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
- Linker Settings
  - Increase heap size to 0x400
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

USB DFU Device lab

- CubeMX create for us file Usbd_dfu_if.c
- This file handling reading and writing into memory
- MEM_If_Init_FS
  - Initialize programing, called on programing start
- MEM_If_DeInit_FS
  - Deinitialize programing, called on programing end
- MEM_If_Erase_FS
  - Erase selected part of memory
- MEM_If_Write_FS
  - Write into selected memory
- MEM_If_Read_FS
  - Read from selected memory
- MEM_If_GetStatus_FS
  - Return state of programing
  - Busy or ready

USB DFU Device lab

- We need to modify the usbd_dfu_it.c file
- Add flash memory programming parameters:

```
/* USER CODE BEGIN PRIVATE_DEFINES */
#define FLASH_ERASE_TIME    (uint16_t)50
#define FLASH_PROGRAM_TIME  (uint16_t)50
/* USER CODE END PRIVATE_DEFINES */
```

- Modify memory handling functions

```
uint16_t MEM_If_Init_FS(void)
{
  /* USER CODE BEGIN 0 */
  HAL_FLASH_Unlock();
  return (USBD_OK);
  /* USER CODE END 0 */
}
```

```
uint16_t MEM_If_DeInit_FS(void)
{
  /* USER CODE BEGIN 1 */
  HAL_FLASH_Lock();
  return (USBD_OK);
  /* USER CODE END 1 */
}
```

USB DFU Device lab


```
uint16_t MEM_If_Erase_FS(uint32_t Add)
{
  /* USER CODE BEGIN 2 */
  uint32_t startsector = 0, sectorerror = 0;

  /* Variable contains Flash operation status */
  HAL_StatusTypeDef status;
  FLASH_EraseInitTypeDef eraseinitstruct;

  /* Get the number of sector */
  startsector = GetSector(Add);
  eraseinitstruct.TypeErase = FLASH_TYPEERASE_SECTORS;
  eraseinitstruct.Banks = GetBank(Add);
  eraseinitstruct.Sector = startsector;
  eraseinitstruct.NbSectors = 1;
  eraseinitstruct.VoltageRange = FLASH_VOLTAGE_RANGE_3;
  status = HAL_FLASHEx_Erase(&eraseinitstruct, &sectorerror);

  if (status != HAL_OK)
  {
    return (USBD_BUSY);
  }
  return (USBD_OK);
  /* USER CODE END 2 */
}
```

USB DFU Device lab


```
uint16_t MEM_If_Write_FS(uint8_t *src, uint8_t *dest, uint32_t Len)
{
  /* USER CODE BEGIN 3 */
uint32_t i = 0;
  for(i = 0; i < Len; i+=4)
  {
/* Device voltage range supposed to be [2.7V to 3.6V], the operation
will be done by byte */
if(HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, (uint32_t)(dest+i),
*(uint32_t*)(src+i)) == HAL_OK)
{
 /* Check the written value */
  if(*(uint32_t *)(src + i) != *(uint32_t*)(dest+i))
  {/* Flash content doesn't match SRAM content */
      return (USBD_FAIL);
  }
}
else
{/* Error occurred while writing data in Flash memory */
   return (USBD_BUSY);
}
  }
  return (USBD_OK);
  /* USER CODE END 3 */
}
```

USB DFU Device lab


```
uint8_t *MEM_If_Read_FS (uint8_t *src, uint8_t *dest, uint32_t Len)
{
  /* Return a valid address to avoid HardFault */
  /* USER CODE BEGIN 4 */
  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 4 */
}
```

USB DFU Device lab


```
uint16_t MEM_If_GetStatus_FS (uint32_t Add, uint8_t Cmd, uint8_t *buffer)
{
  /* USER CODE BEGIN 5 */
switch(Cmd)
  {
  case DFU_MEDIA_PROGRAM:
buffer[1] = (uint8_t)FLASH_PROGRAM_TIME;
buffer[2] = (uint8_t)(FLASH_PROGRAM_TIME << 8);
buffer[3] = 0;
break;

  case DFU_MEDIA_ERASE:
  default:
buffer[1] = (uint8_t)FLASH_ERASE_TIME;
buffer[2] = (uint8_t)(FLASH_ERASE_TIME << 8);
buffer[3] = 0;
break;
  }
  return  (USBD_OK);
  /* USER CODE END 5 */
}
```

USB DFU Device lab

- Two additional functions needs to be added

```
/* USER CODE BEGIN PRIVATE_FUNCTIONS_DECLARATION */
static uint32_t GetSector(uint32_t Address);
static uint32_t GetBank(uint32_t Addr);
/* USER CODE END PRIVATE_FUNCTIONS_DECLARATION */
```

USB DFU Device lab

- We need to add function to get sector number from address

```
/* USER CODE BEGIN PRIVATE_FUNCTIONS_IMPLEMENTATION */
static uint32_t GetSector(uint32_t Address)
{
  uint32_t sector = 0;

 if((Address < ADDR_FLASH_SECTOR_1) && (Address >= ADDR_FLASH_SECTOR_0))
  {
    sector = FLASH_SECTOR_0;
  }
  else if((Address < ADDR_FLASH_SECTOR_2) && (Address >=
ADDR_FLASH_SECTOR_1))
  {
    sector = FLASH_SECTOR_1;
  }
  else if((Address < ADDR_FLASH_SECTOR_3) && (Address >=
ADDR_FLASH_SECTOR_2))
  {
    sector = FLASH_SECTOR_2;
  }
```

USB DFU Device lab


```
else if((Address < ADDR_FLASH_SECTOR_4) && (Address >= ADDR_FLASH_SECTOR_3))
  {
    sector = FLASH_SECTOR_3;
  }
else if((Address < ADDR_FLASH_SECTOR_5) && (Address >= ADDR_FLASH_SECTOR_4))
  {
    sector = FLASH_SECTOR_4;
  }
  else if((Address < ADDR_FLASH_SECTOR_6) && (Address >=
ADDR_FLASH_SECTOR_5))
  {
    sector = FLASH_SECTOR_5;
  }
  else if((Address < ADDR_FLASH_SECTOR_7) && (Address >=
ADDR_FLASH_SECTOR_6))
  {
    sector = FLASH_SECTOR_6;
  }
  else
  {
    sector = FLASH_SECTOR_7;
  }
  return sector;
}
```

USB DFU Device lab

- And simple function to return bank number – to keep the code universal

```
static uint32_t GetBank(uint32_t Addr)
{
  uint32_t bank = 0;

  /* Sector in bank 1 */
  bank = FLASH_BANK_1;
  return bank;
}

/* USER CODE END PRIVATE_FUNCTIONS_IMPLEMENTATION */
```

USB DFU Device lab

- To file usbd_dfu_if.h flash address sector range need to be added

```
/* USER CODE BEGIN EXPORTED_DEFINES */
#define ADDR_FLASH_SECTOR_0     ((uint32_t)0x08000000) /* Base @ of Sector 0, 16 Kbytes */
#define ADDR_FLASH_SECTOR_1     ((uint32_t)0x08004000) /* Base @ of Sector 1, 16 Kbytes */
#define ADDR_FLASH_SECTOR_2     ((uint32_t)0x08008000) /* Base @ of Sector 2, 16 Kbytes */
#define ADDR_FLASH_SECTOR_3     ((uint32_t)0x0800C000) /* Base @ of Sector 3, 16 Kbytes */
#define ADDR_FLASH_SECTOR_4     ((uint32_t)0x08010000) /* Base @ of Sector 4, 64 Kbytes */
#define ADDR_FLASH_SECTOR_5     ((uint32_t)0x08020000) /* Base @ of Sector 5, 128 Kbytes */
#define ADDR_FLASH_SECTOR_6     ((uint32_t)0x08040000) /* Base @ of Sector 6, 128 Kbytes */
#define ADDR_FLASH_SECTOR_7     ((uint32_t)0x08060000) /* Base @ of Sector 7, 128 Kbytes */

/* USER CODE END EXPORTED_DEFINES */
```

USB DFU Device lab

- To main.c add function for jump into user application if user button is not pressed

```
/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
typedef  void (*pFunction)(void);
pFunction JumpToApplication;
uint32_t JumpAddress;
/* USER CODE END PFP */
```

- LED toggle for signaling what part of program is executed

```
while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_7);
  HAL_Delay(500);
  }
  /* USER CODE END 3 */
```

USB DFU Device lab

- Functionality to switch program execution to different address and change MX_USB_DEVICE_Init() position in the code

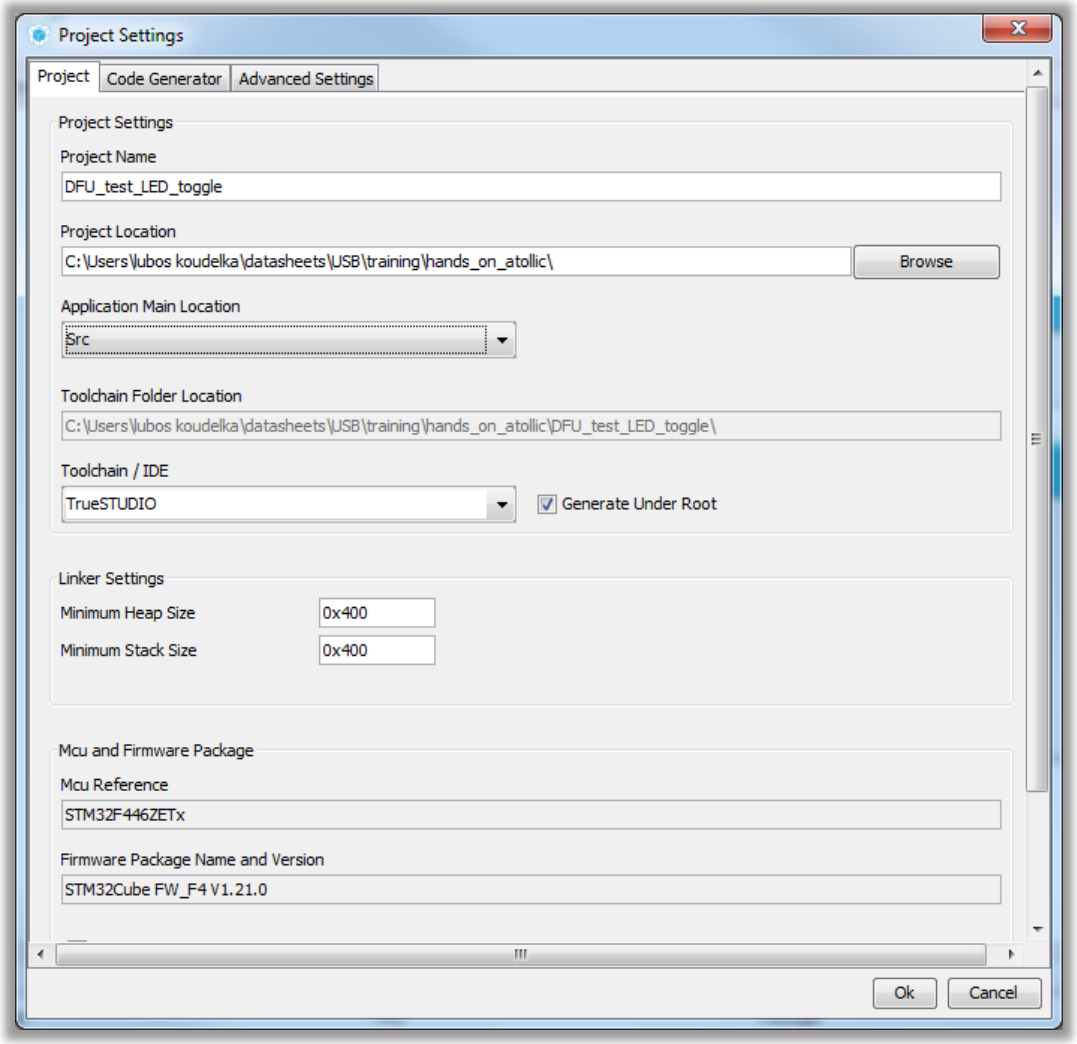
```
/* Initialize all configured peripherals */
 MX_GPIO_Init();

 /* USER CODE BEGIN 2 */
 if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13) == 0x00)
 {
   /* Test if user code is programmed starting from USBD_DFU_APP_DEFAULT_ADD address */
   if(((*(__IO uint32_t*)USBD_DFU_APP_DEFAULT_ADD) & 0x2FF00000 ) == 0x20000000)
   {
     /* Jump to user application */
     JumpAddress = *(__IO uint32_t*) (USBD_DFU_APP_DEFAULT_ADD + 4);
     JumpToApplication = (pFunction) JumpAddress;
     /* Initialize user application's Stack Pointer */
     __set_MSP((*(__IO uint32_t*) USBD_DFU_APP_DEFAULT_ADD ));
     JumpToApplication();
   }
 }
 MX_USB_DEVICE_Init();
 /* USER CODE END 2 */
```

USB DFU Device lab

- USB DFU application is ready, for testing DFU functionality we create only simple LED toggling project to be flashed using DFU
- Create project in CubeMX
  - Menu > File > New Project
  - Select STM32F4 > STM32F446 > LQFP144 > STM32F446ZETx
- Configure GPIOs connected to LEDs as GPIO output – PB0, PB7 and PB14

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



USB DFU Device lab

- Application starting address is 0x08008000, this needs to be changed in STM32F446ZETx_FLASH.ld:

```
/* Specify the memory areas */
MEMORY
{
RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 128K
FLASH (rx)      : ORIGIN = 0x08008000, LENGTH = 256K
}
```

- And in stm32f446xx.h – vector table reallocation

```
#define FLASH_BASE            0x08008000U
```

USB DFU Device lab

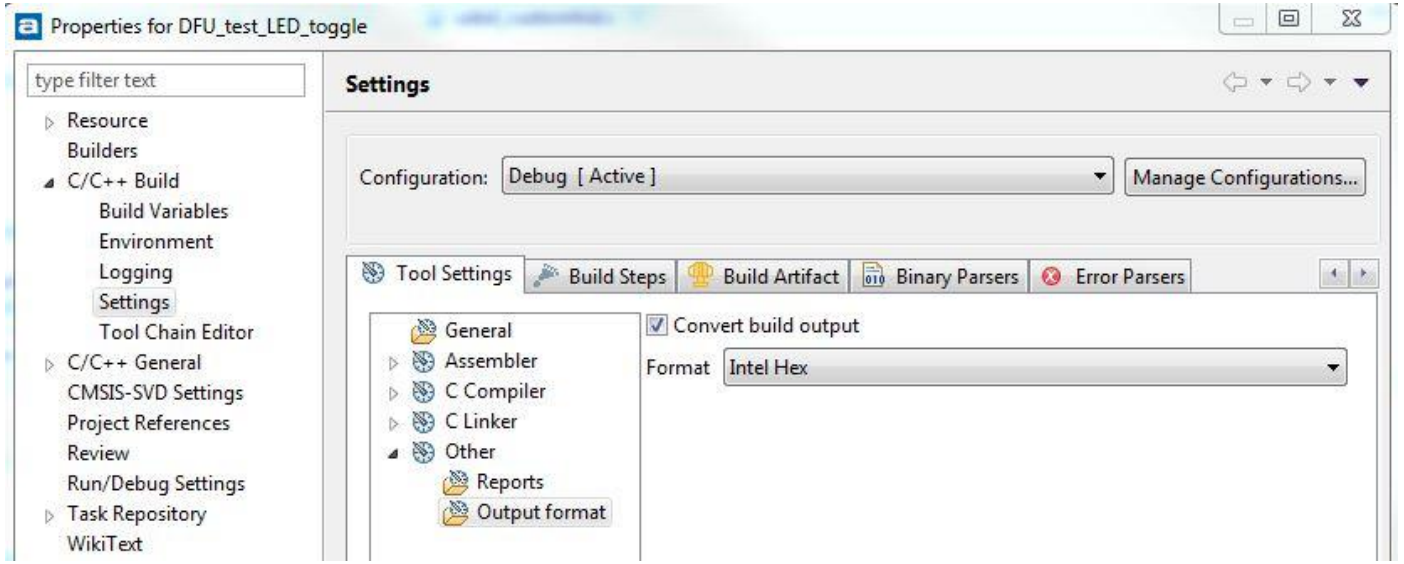
- Add to main.c LED toggling

```
/* USER CODE BEGIN WHILE */
 while (1)
 {
 /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */
 HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
 HAL_Delay(100);
 HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_0);
 HAL_Delay(100);
 }
```

- Now the project is ready to be compiled

USB DFU Device lab

- For DFU tool we need to get .hex file output of the project
- In Atollic project Properties -> C/C++ Build -> Settings -> Tool Settings ->Other -> Output Format select Intel Hex



- Rebuild the project, .hex file is created in project structure

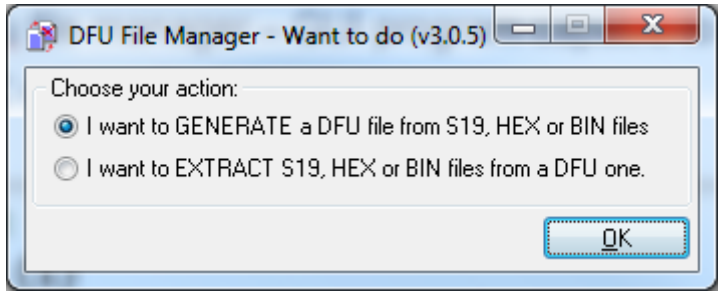
life.augmented

USB DFU Device lab

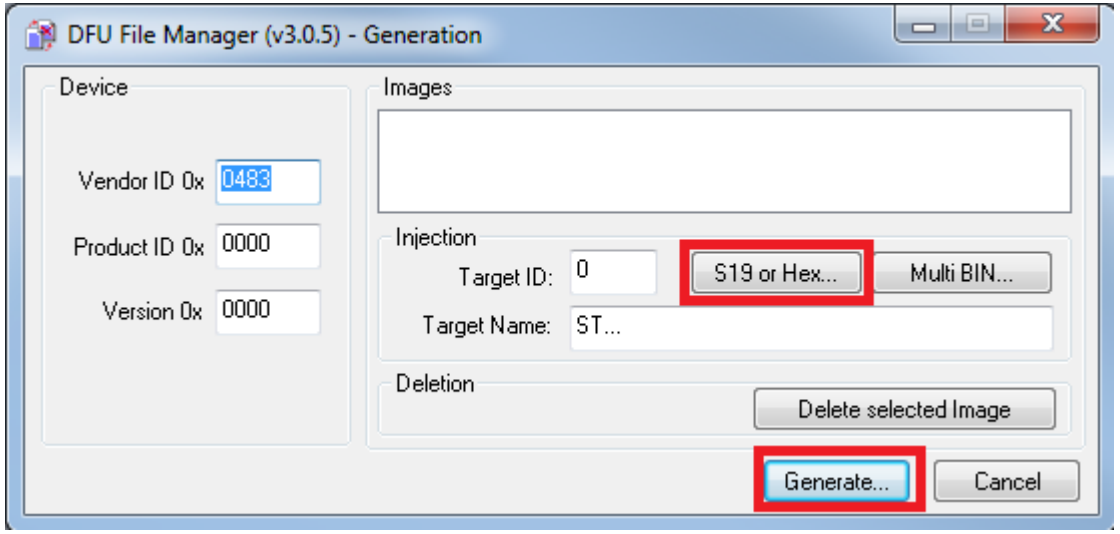
- For DFU programing on Windows PC you can find on st.com DfuSe tool which contains:
  - DfuSeDemo – GUI application for flashing .dfu files into various parts of STM32 MCU in DFU mode – DfuSe is capable to write into user FLASH, OTP, option bytes (if implemented in bootloader firmware). Both data directions are supported, download and upload
  - Dfu file manager – GUI program capable to convert .hex and other format to .dfu file
  - PC drivers for DFU device
  - Source files for DFU tools and documentation

USB DFU Device lab

- Creating a DFU file in DFU file manager from hex file

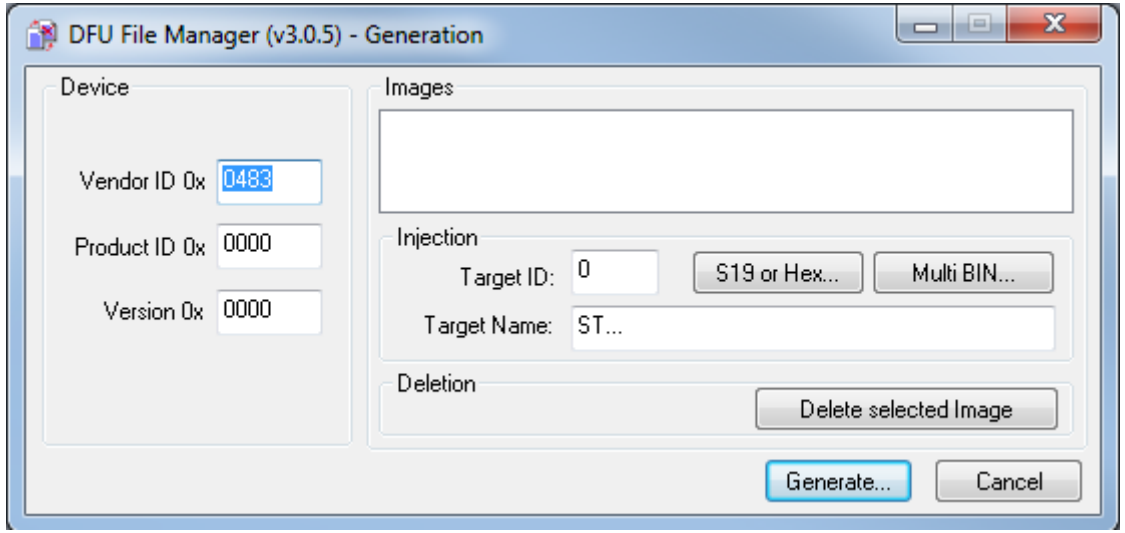


- Choose hex file from LED project and generate dfu file



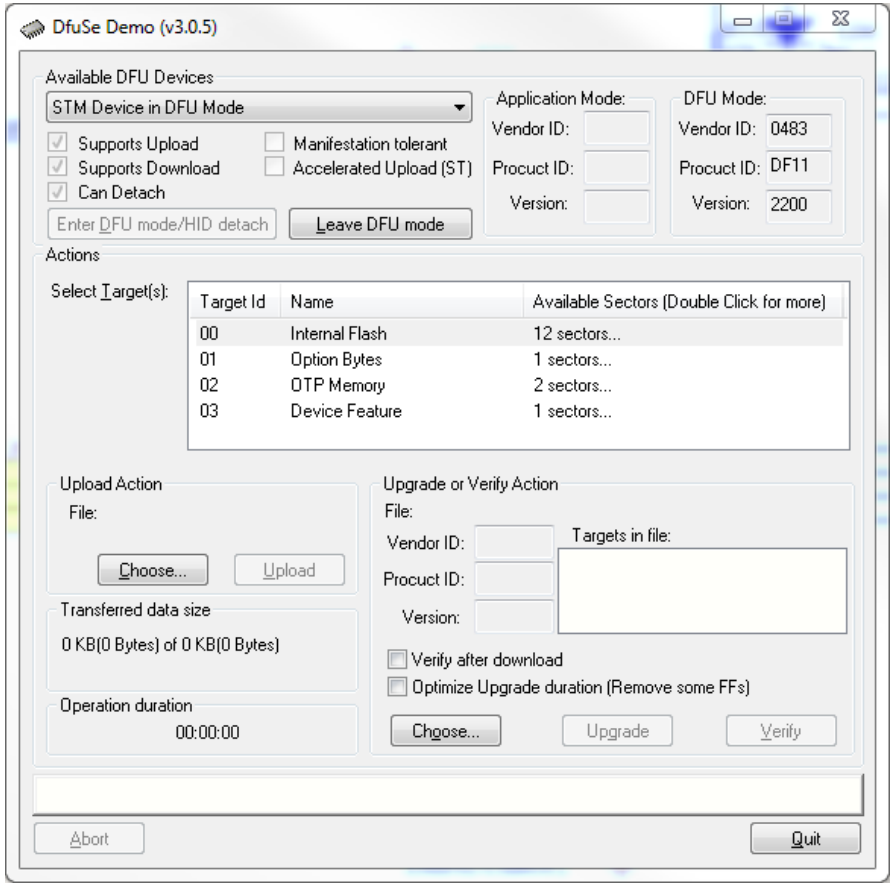
USB DFU Device lab

- Device – information about target device
  - For some application is mandatory to fill completely PID, VID and version
- Target ID – code of target memory
  - 0 – Internal Flash
  - 1 – Option Bytes
  - 2 – OTP Memory
  - 3 – Device Feature



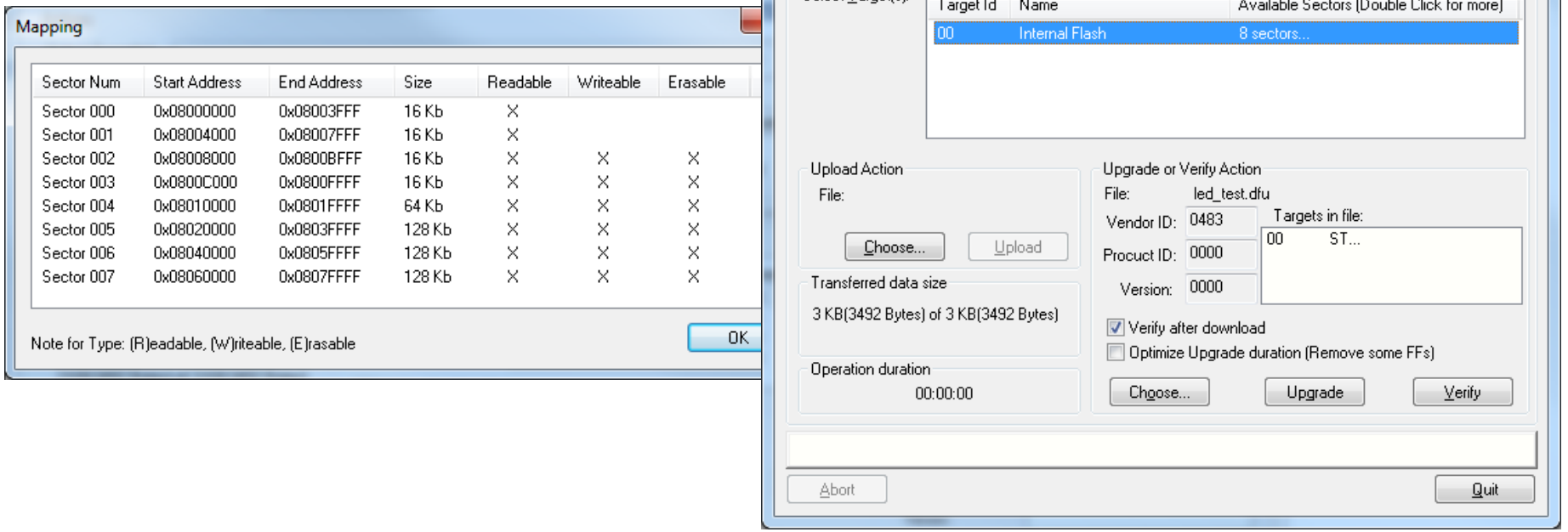
USB DFU Device lab

- Fully filled information about device and target ID are visible when system bootloader DFU is connected to DfuSe Demo application
  - Figure show example of connecting STM32F407 in DFU mode



USB DFU Device lab

- Demonstrated user DFU implementation contain only Internal Flash
  - Choose generated dfu file
  - Upgrade
  - Press Leave DFU mode or reset MCU to enter user application

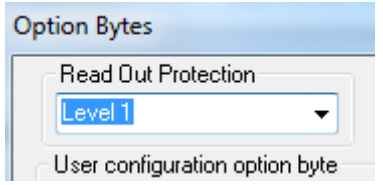


life.augmented

USB DFU Device lab

- Now we can demonstrate, that with user DFU bootloader is possible to work also with Read Out Protection enabled
  - Bootloader in system memory is not able to work with RDP level 1
- For demonstration purposes, do mass erase of the MCU using ST-link utility
  - It’s sure, that no FW is loaded higher address

- Flash again DFU device application using your IDE or ST-link utility
- Enable RDP level 1 in ST-link utility
  - Target -> Option Bytes



USB DFU Device lab

- Remove CN4 jumpers (debugger disconnect) and make Power on Reset
  - ST-link is reading information from MCU flash during start-up, while RDP level 1 is enabled, this will cause that MCU won’t response
- Load FW update using DfuSe Demo like in previous demo without RDP
- Read out protection don’t impact functionality of user DFU bootloader