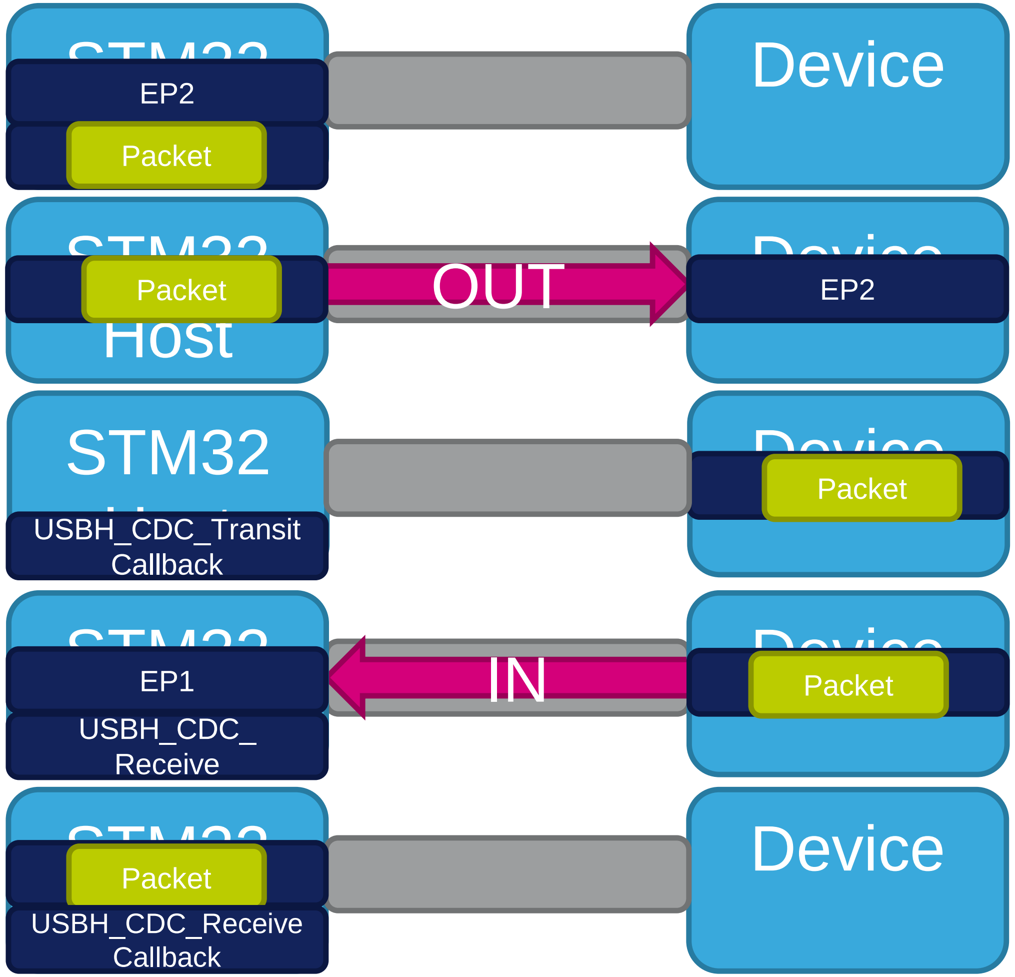
USB VCP Host

USB VCP Host

- The CubeMX CDC host is very easy to handle
  - There is only few functions 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, USBH_CDC_Receive serve for data reception
- Pair of weak callback available for transmit complete notification - USBH_CDC_TransmitCallback and USBH_CDC_ReceiveCallback

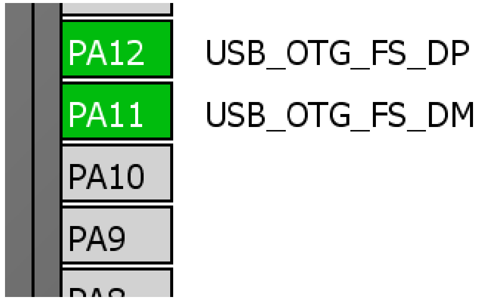
USB VCP Host

- CDC HOST FLOW



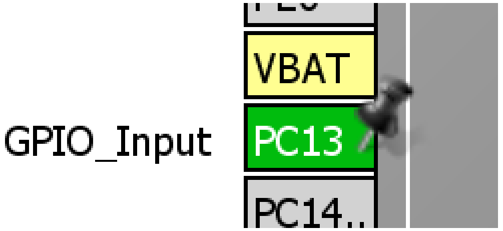
USB VCP Host

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



PA12 USB_OTG_FS_DP

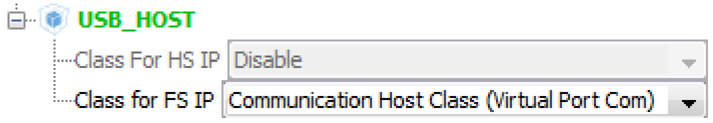
PA11 USB_OTG_FS_DM



GPIO_Input PC13

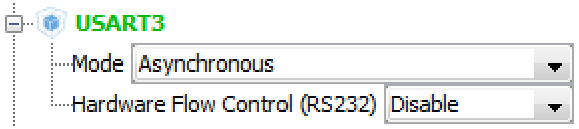
USB VCP Host

- Set GPIOs connected to LEDs as GPIO output – PB0, PB7 and PB14
- Select Communication host class in MiddleWares

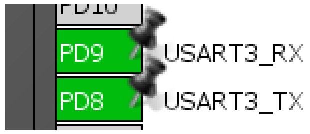


USB_HOST
Class For HS IP: Disable
Class for FS IP: Communication Host Class (Virtual Port Com)

- Configure RCC clocks
  - Set 8 MHz HSE as PLL input and HCLK frequency 168 MHz
- Add USART3 for debug purposes
  - USART3 is connected to STlink virtual COMport functionality
  - PD9 – USART3_RX
  - PD8 – USART3_TX
  - For easier handling more convenient DMA implementation is not used

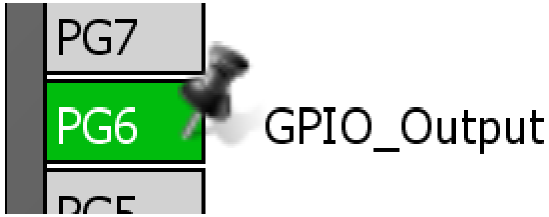


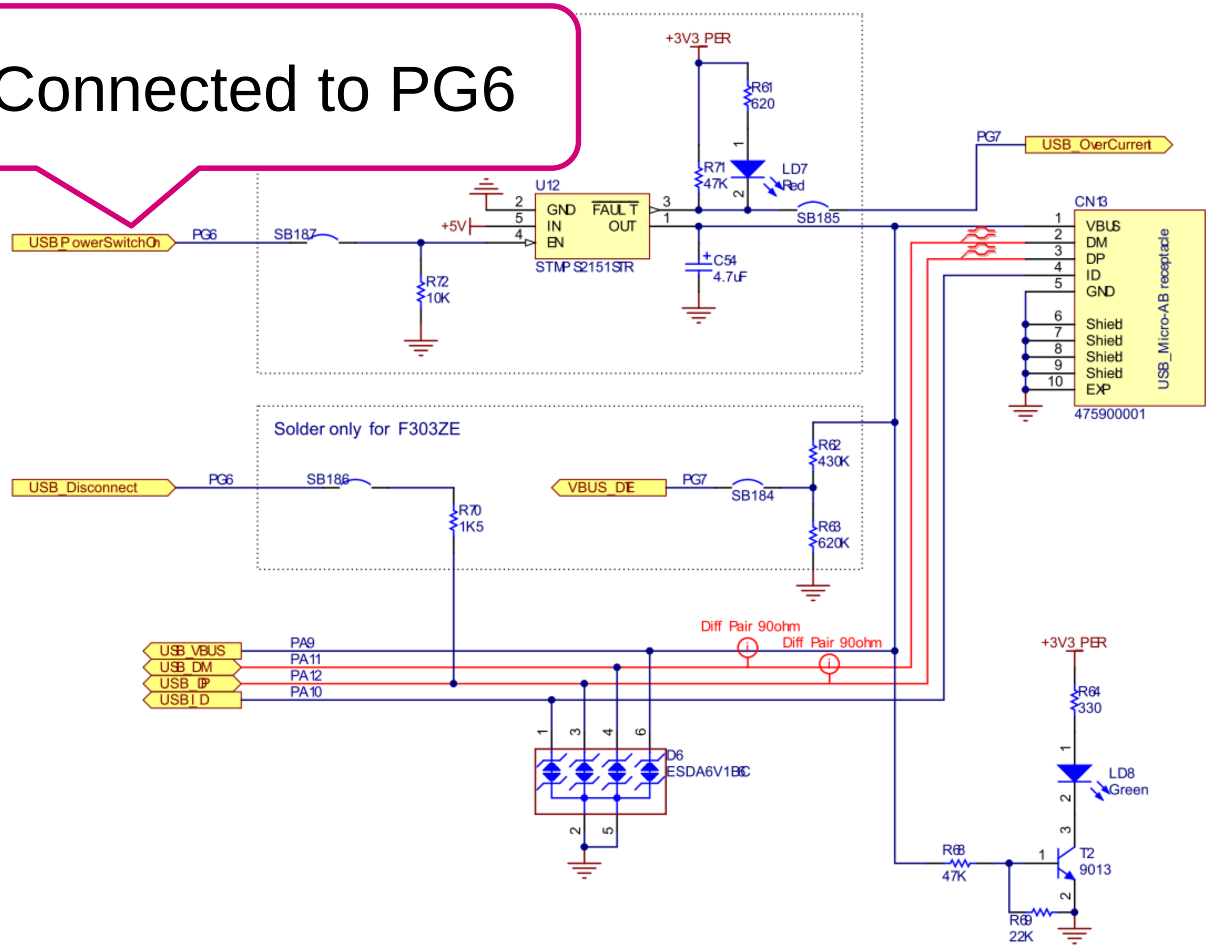
USART3
Mode: Asynchronous
Hardware Flow Control (RS232): Disable



USB VCP Host

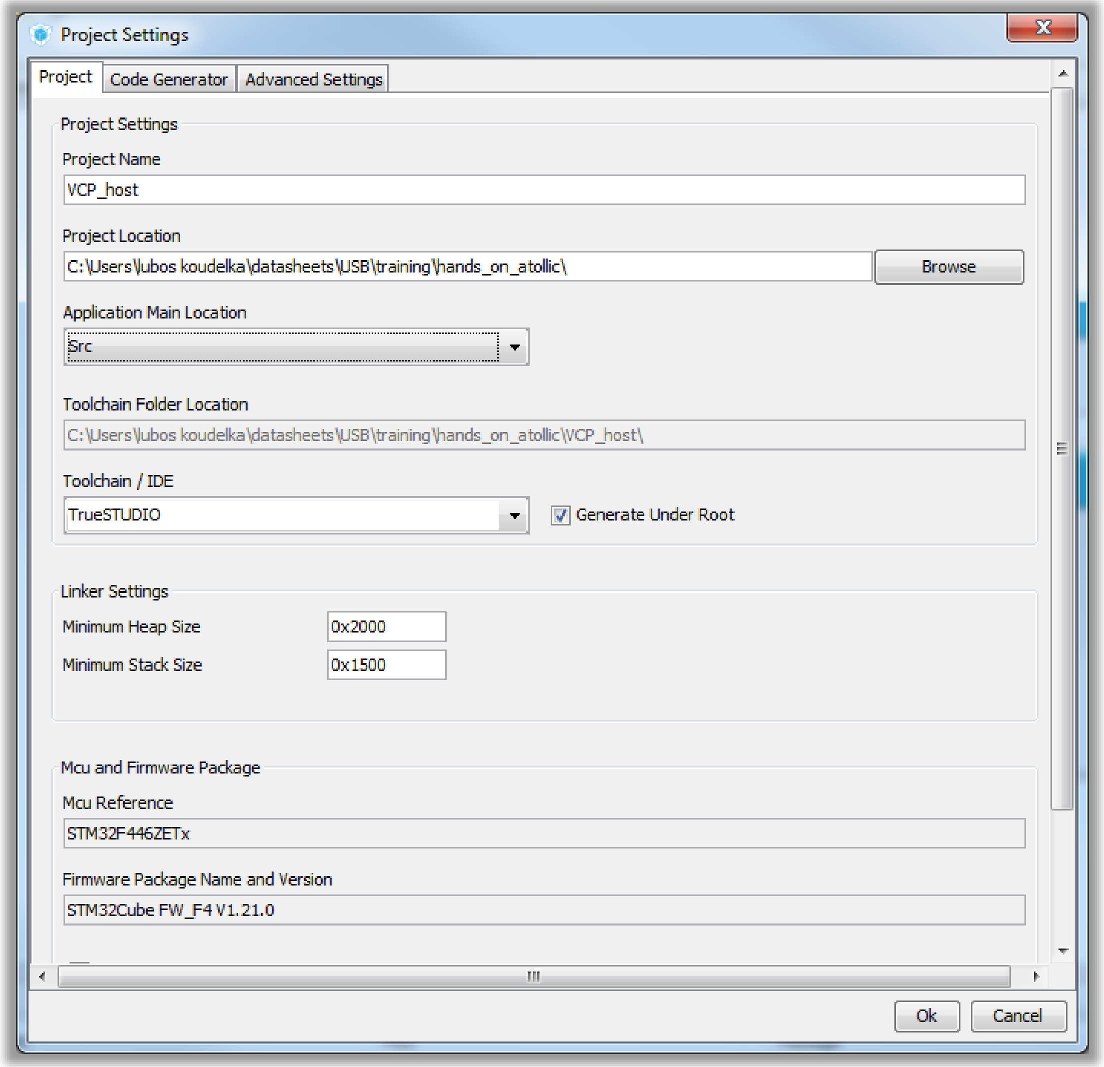
- HOST must also power the device -> we need to enable voltage regulator connected to VBUS line
- Set PG6 as GPIO output




Connected to PG6

USB VCP Host

- 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 0x2000
  - Increase Stack size to 0x1500
- Now we can Generate Code
  - Menu > Project > Generate Code



Project Settings

Project Name: VCP_host

Project Location: C:\Users\lubos koudelka\datasheets\USB\training\hands_on_atollic\

Application Main Location: Src

Toolchain Folder Location: C:\Users\lubos koudelka\datasheets\USB\training\hands_on_atollic\VCP_host\

Toolchain / IDE: TrueSTUDIO ☑ Generate Under Root

Linker Settings

Minimum Heap Size: 0x2000

Minimum Stack Size: 0x1500

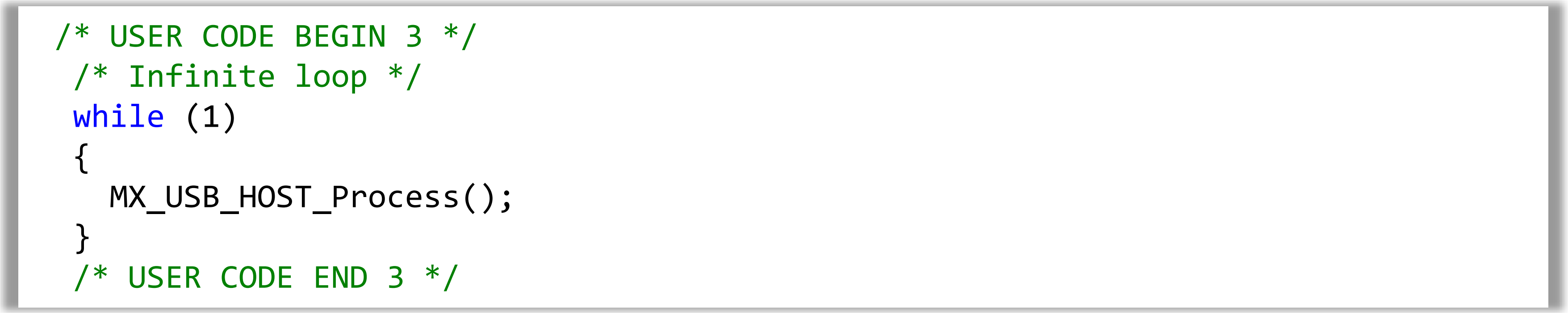
Mcu and Firmware Package

Mcu Reference: STM32F446ZETx

Firmware Package Name and Version: STM32Cube FW_F4 V1.21.0

USB VCP Host

- 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 */
```

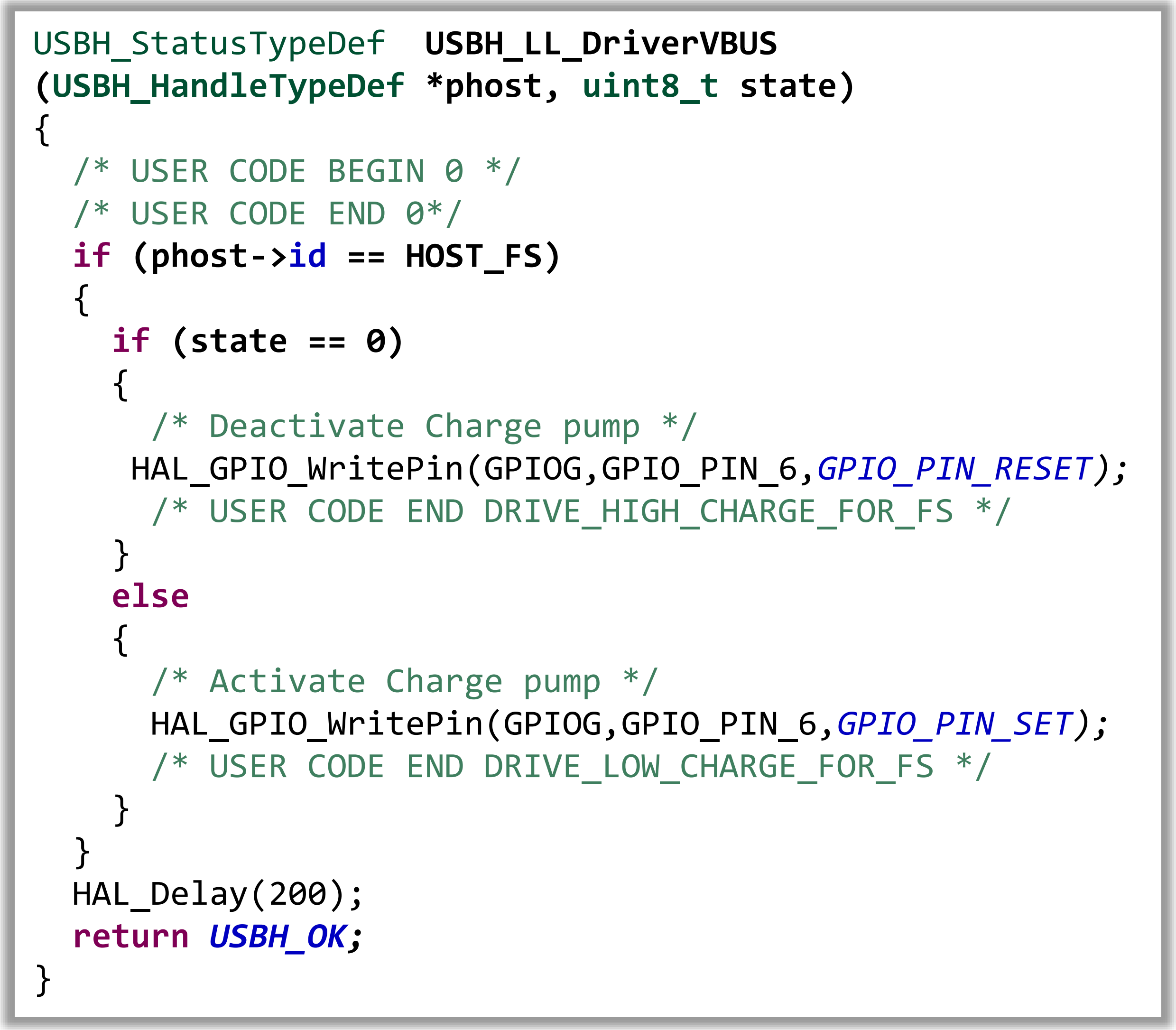
- CubeMX generate it in infinite loop

- But it’s more recommend to handle it by interrupt or in RTOS put it into task
  - When FreeRTOS option is used in CubeMX, Host_process function is called in USB task

USB VCP Host

- In usbh_conf.c is function for handling USB VBUS voltage level - USBH_LL_DriverVBUS

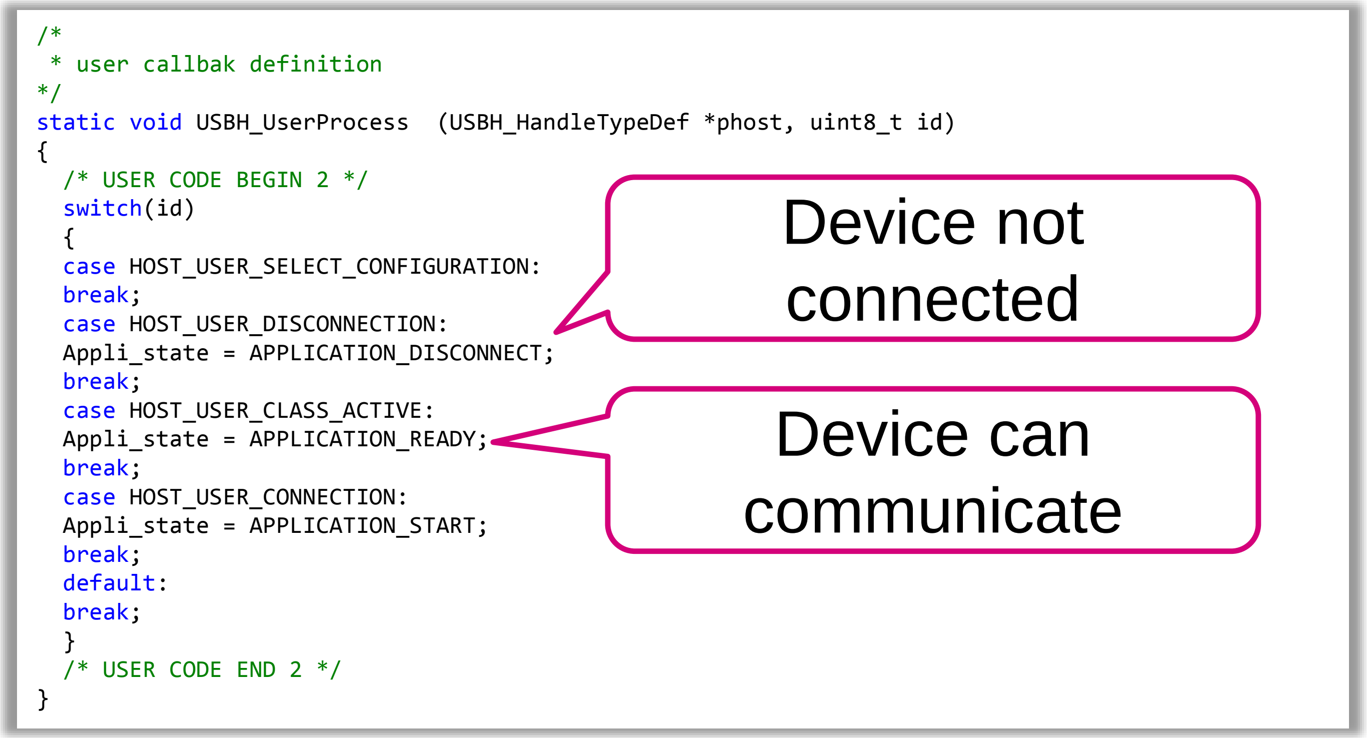
- Pin PG6 controls power source for USB VBUS



```
USBH_StatusTypeDef  USBH_LL_DriverVBUS
(USBH_HandleTypeDef *phost, uint8_t state)
{
  /* USER CODE BEGIN 0 */
  /* USER CODE END 0*/
  if (phost->id == HOST_FS)
  {
    if (state == 0)
    {
      /* Deactivate Charge pump */
     HAL_GPIO_WritePin(GPIOG,GPIO_PIN_6,GPIO_PIN_RESET);
      /* USER CODE END DRIVE_HIGH_CHARGE_FOR_FS */
    }
    else
    {
      /* Activate Charge pump */
      HAL_GPIO_WritePin(GPIOG,GPIO_PIN_6,GPIO_PIN_SET);
      /* USER CODE END DRIVE_LOW_CHARGE_FOR_FS */
    }
  }
  HAL_Delay(200);
  return USBH_OK;
}
```

USB VCP Host

- 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 connected device



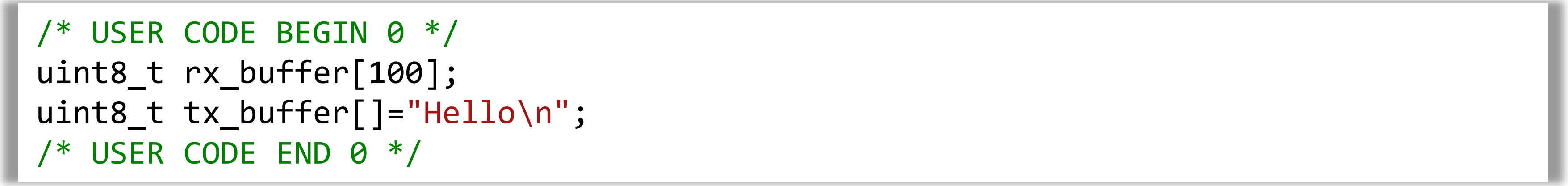
```
/*
 * user callbak definition
*/
static void USBH_UserProcess  (USBH_HandleTypeDef *phost, uint8_t id)
{
  /* USER CODE BEGIN 2 */
  switch(id)
  {
  case HOST_USER_SELECT_CONFIGURATION:
  break;
  case HOST_USER_DISCONNECTION:
  Appli_state = APPLICATION_DISCONNECT;
  break;
  case HOST_USER_CLASS_ACTIVE: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

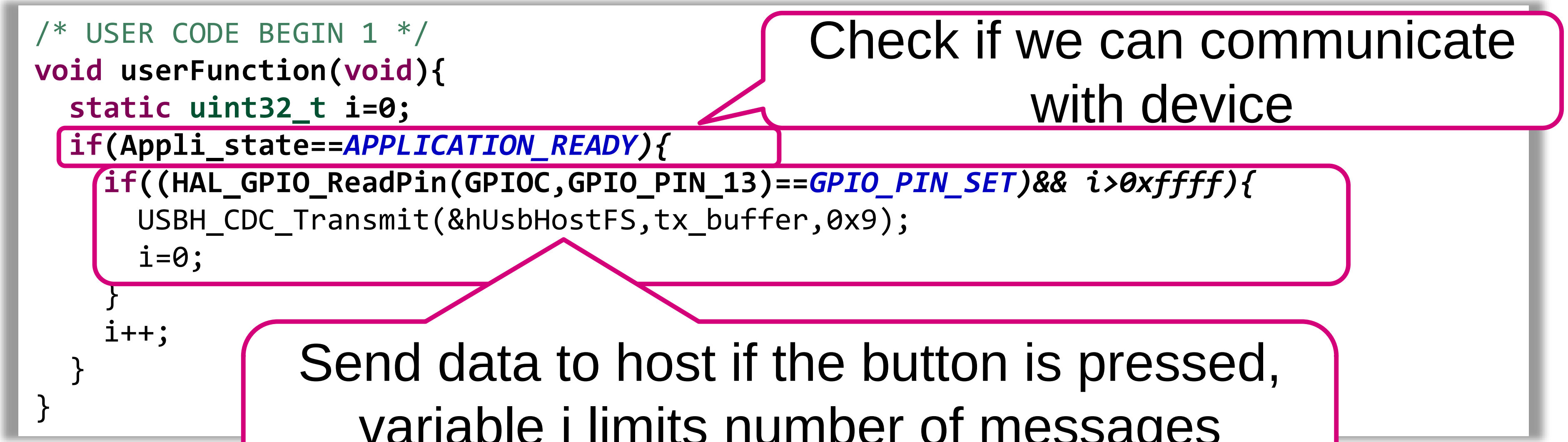
USB VCP Host

- 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



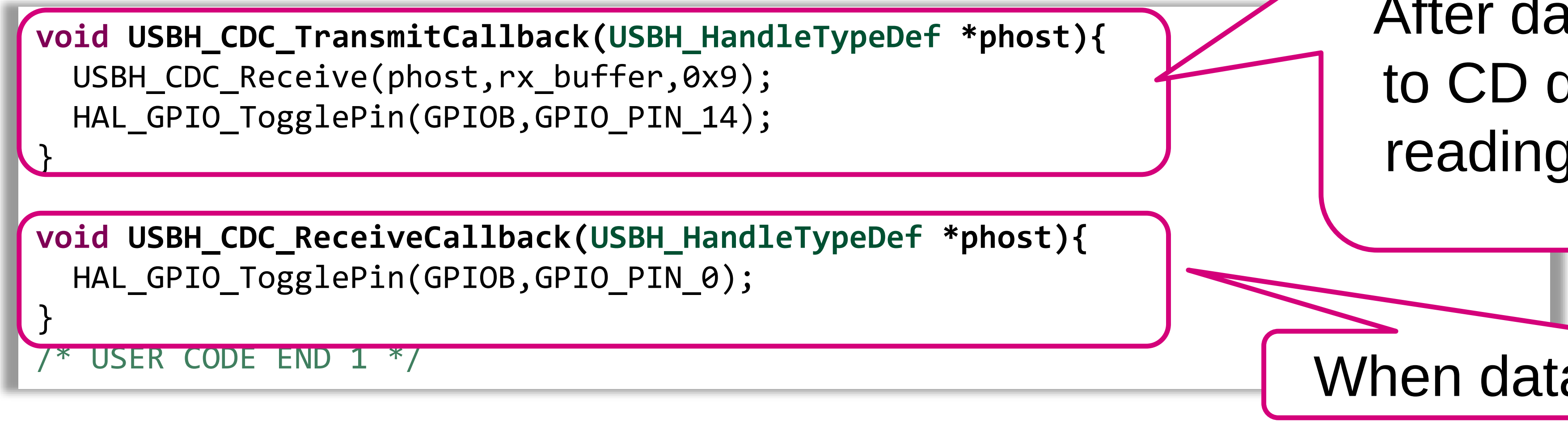
```
/* USER CODE BEGIN 1 */
void userFunction(void){
  static uint32_t i=0;
  if(Appli_state==APPLICATION_READY){
    if((HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13)==GPIO_PIN_SET)&& i>0xffff){
      USBH_CDC_Transmit(&hUsbHostFS,tx_buffer,0x9);
      i=0;
    }
    i++;
  }
}
```

Check if we can communicate with device

Send data to host if the button is pressed, variable i limits number of messages
We send tx_buffer 9bytes long

USB VCP Host

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



```
void USBH_CDC_TransmitCallback(USBH_HandleTypeDef *phost){
  USBH_CDC_Receive(phost,rx_buffer,0x9);
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
}

void USBH_CDC_ReceiveCallback(USBH_HandleTypeDef *phost){
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_0);
}
/* USER CODE END 1 */
```

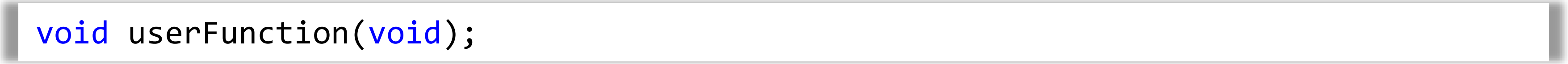
After data was transmitted to CD device we Request reading from CDC device

When data was toggle LED

USB VCP Host

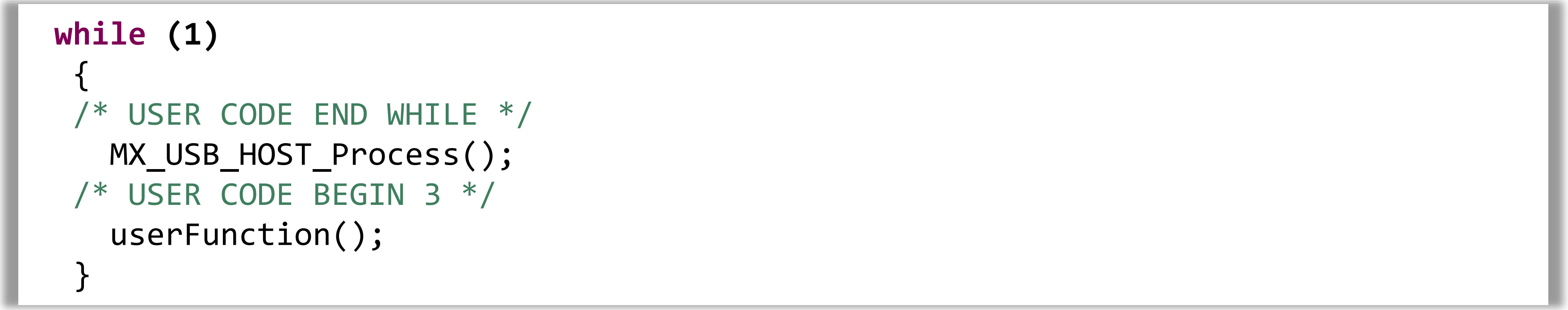
- Now only thing what is missing is call userFunction which will send data after button press

- User function declaration need to added into usb_host.h



```
void userFunction(void);
```

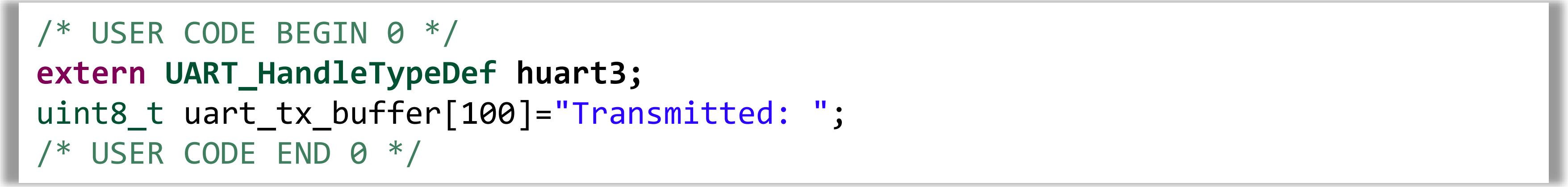
- And put userFunction in while loop



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

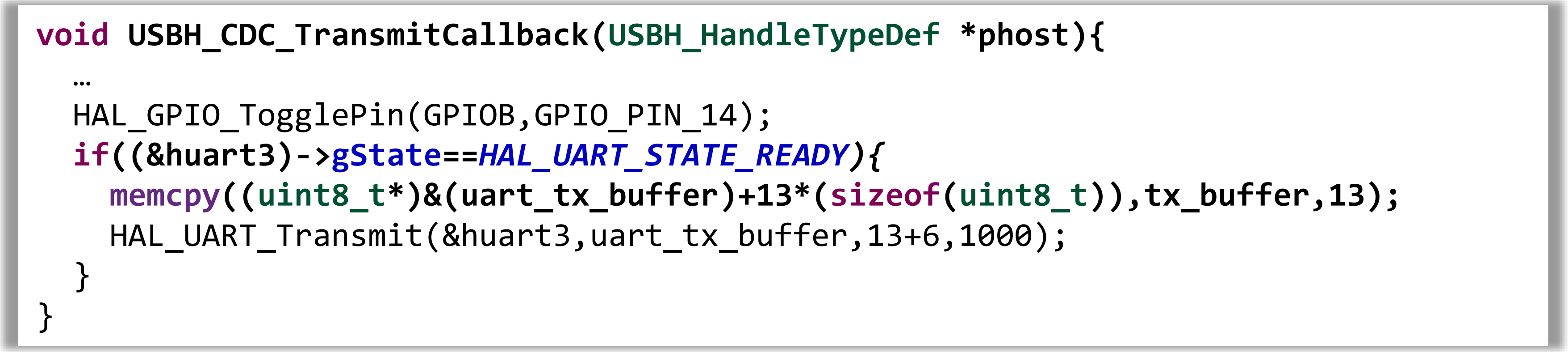
USB VCP Host

- For function demonstration we can add debug messages print using STlink virtual COM port connected to USART3
    - To usb_host.c add



```
/* USER CODE BEGIN 0 */
extern UART_HandleTypeDef huart3;
uint8_t uart_tx_buffer[100]="Transmitted: ";
/* USER CODE END 0 */
```

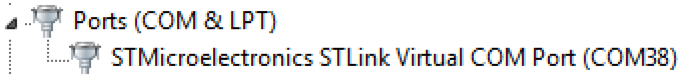
- Message will be send only if no other message is already being sent
    - More convenient FIFO/DMA approach is not used due to higher complexity of the code



```
void USBH_CDC_TransmitCallback(USBH_HandleTypeDef *phost){
  …
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
  if((&huart3)->gState==HAL_UART_STATE_READY){
    memcpy((uint8_t*)&(uart_tx_buffer)+13*(sizeof(uint8_t)),tx_buffer,13);
    HAL_UART_Transmit(&huart3,uart_tx_buffer,13+6,1000);
  }
}
```

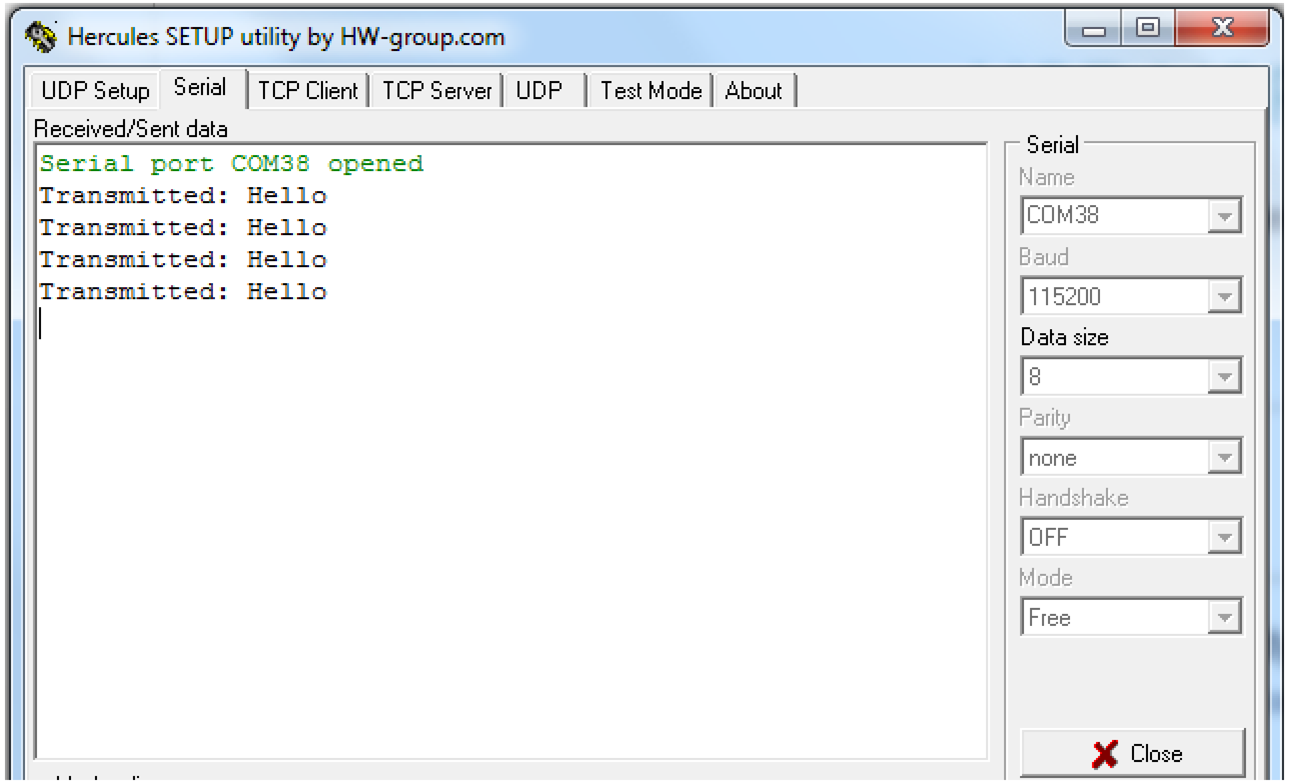
USB VCP Host

- Then in device manager find COM port number of connected host board STlink



Ports (COM & LPT)
STMicroelectronics STLink Virtual COM Port (COM38)

- Debug output can be view in any COM port terminal application

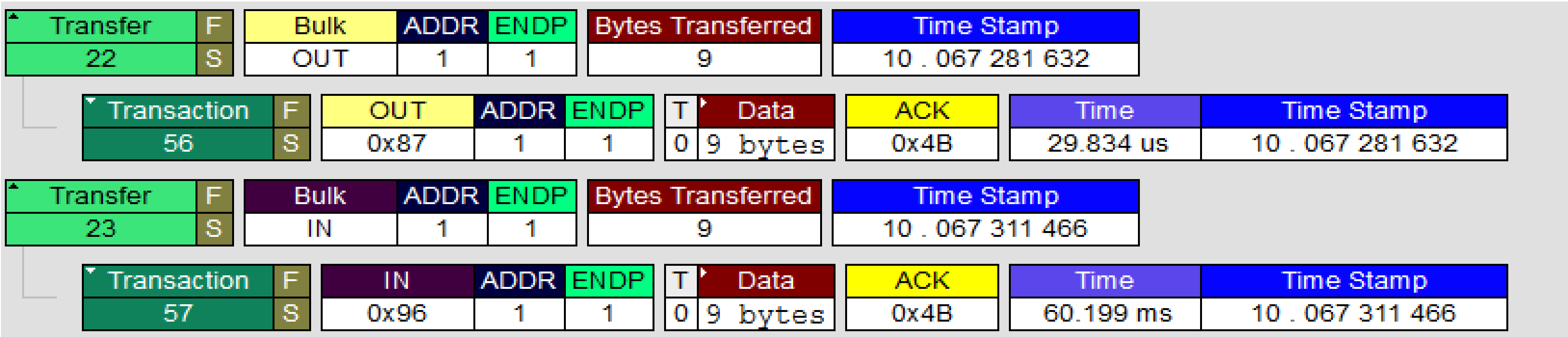


```
Serial port COM38 opened
Transmitted: Hello
Transmitted: Hello
Transmitted: Hello
Transmitted: Hello
```

Serial
Name: COM38
Baud: 115200
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

USB VCP Host

- For demonstration we use CDC device project from VCP hands on
- Data are send from host to device when button is pressed, device immediately send the same message back



Transfer	F	Bulk	ADDR	ENDP	Bytes Transferred	Time Stamp
22	S	OUT	1	1	9	10 . 067 281 632

Transaction	F	OUT	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
56	S	0x87	1	1	0	9 bytes	0x4B	29.834 us	10 . 067 281 632

Transfer	F	Bulk	ADDR	ENDP	Bytes Transferred	Time Stamp
23	S	IN	1	1	9	10 . 067 311 466

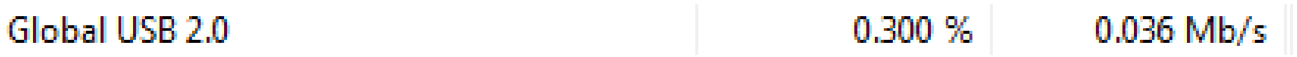
Transaction	F	IN	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
57	S	0x96	1	1	0	9 bytes	0x4B	60.199 ms	10 . 067 311 466

VCP_host.usb

USB VCP Host

- Bus load in this scenario is really low – there are no IN packet pending, device is answering immediately

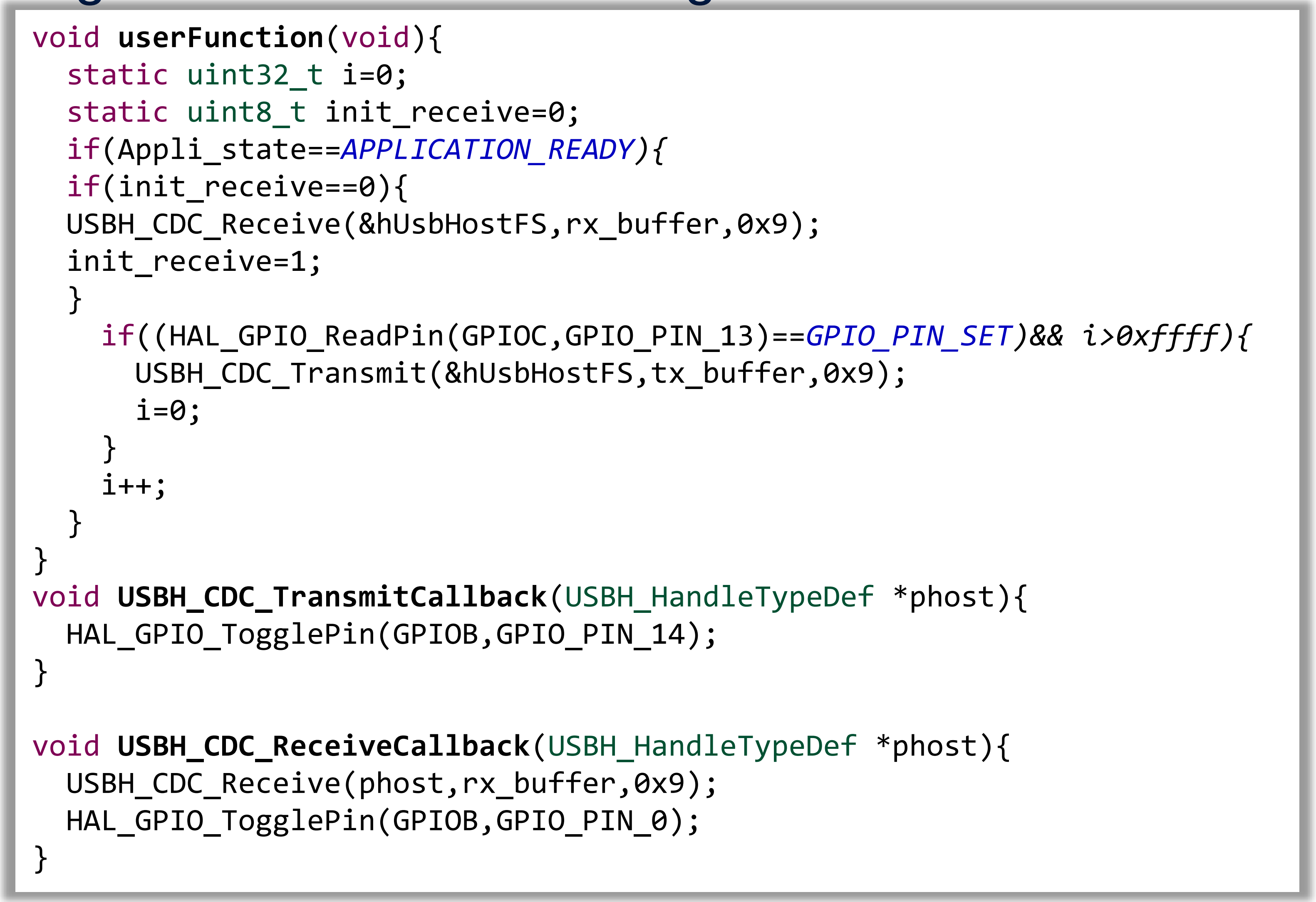
- Host can receive from device only one packet just after transmit
  - No other data are received by host



Global USB 2.0	0.300 %	0.036 Mb/s

USB VCP Host

- With few changes in userFunction we get more realistic behavior

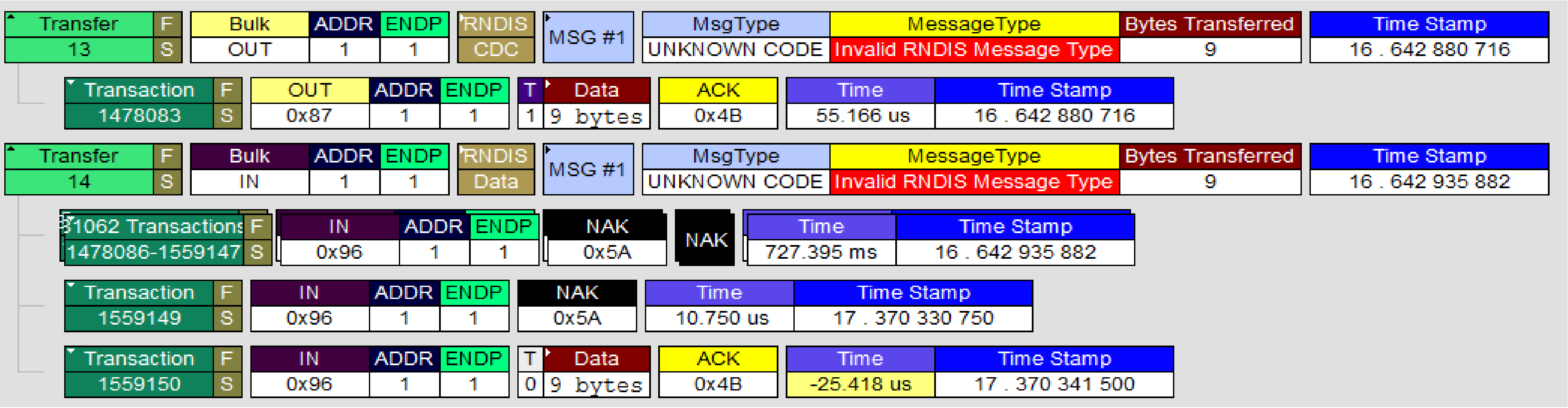


```
void userFunction(void){
  static uint32_t i=0;
  static uint8_t init_receive=0;
  if(Appli_state==APPLICATION_READY){
  if(init_receive==0){
  USBH_CDC_Receive(&hUsbHostFS,rx_buffer,0x9);
  init_receive=1;
  }
    if((HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13)==GPIO_PIN_SET)&& i>0xffff){
      USBH_CDC_Transmit(&hUsbHostFS,tx_buffer,0x9);
      i=0;
    }
    i++;
  }
}
void USBH_CDC_TransmitCallback(USBH_HandleTypeDef *phost){
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_14);
}

void USBH_CDC_ReceiveCallback(USBH_HandleTypeDef *phost){
  USBH_CDC_Receive(phost,rx_buffer,0x9);
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_0);
}
```

USB VCP Host

- Now the host is sending IN request each time a NAK is received to IN request



Transfer	F	Bulk	ADDR	ENDP	RNDIS	MSG #1	MsgType	MessageType	Bytes Transferred	Time Stamp
13	S	OUT	1	1	CDC		UNKNOWN CODE	Invalid RNDIS Message Type	9	16 . 642 880 716

Transaction	F	OUT	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
1478083	S	0x87	1	1	1	9 bytes	0x4B	55.166 us	16 . 642 880 716

Transfer	F	Bulk	ADDR	ENDP	RNDIS	MSG #1	MsgType	MessageType	Bytes Transferred	Time Stamp
14	S	IN	1	1	Data		UNKNOWN CODE	Invalid RNDIS Message Type	9	16 . 642 935 882

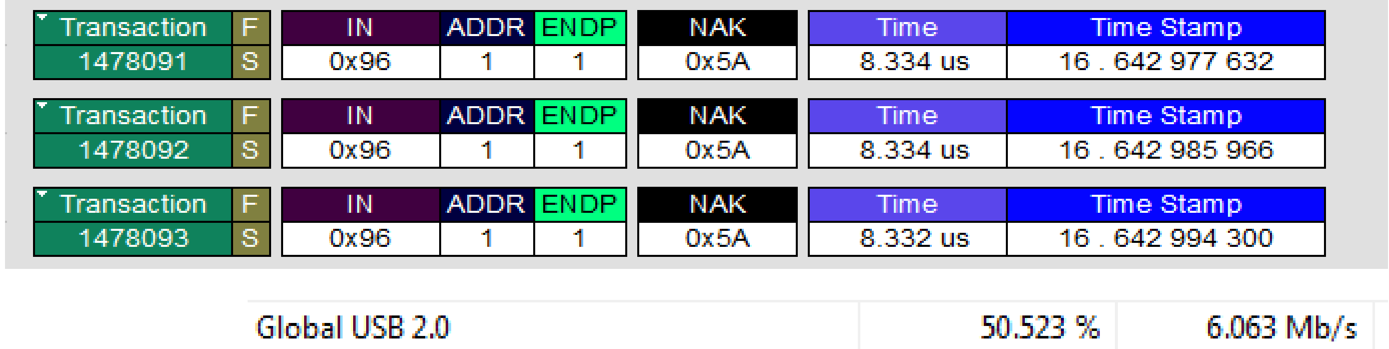
81062 Transactions	F	IN	ADDR	ENDP	NAK	NAK	Time	Time Stamp
1478086-1559147	S	0x96	1	1	0x5A		727.395 ms	16 . 642 935 882

Transaction	F	IN	ADDR	ENDP	NAK	Time	Time Stamp
1559149	S	0x96	1	1	0x5A	10.750 us	17 . 370 330 750

Transaction	F	IN	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
1559150	S	0x96	1	1	0	9 bytes	0x4B	-25.418 us	17 . 370 341 500

VCP_host_NAK.usb

- Host is ready to receive packet from the device anytime, but load on the bus is really big – there is a NAK each ~8.3 us



Transaction	F	IN	ADDR	ENDP	NAK	Time	Time Stamp
1478091	S	0x96	1	1	0x5A	8.334 us	16 . 642 977 632

Transaction	F	IN	ADDR	ENDP	NAK	Time	Time Stamp
1478092	S	0x96	1	1	0x5A	8.334 us	16 . 642 985 966

Transaction	F	IN	ADDR	ENDP	NAK	Time	Time Stamp
1478093	S	0x96	1	1	0x5A	8.332 us	16 . 642 994 300

Global USB 2.0	50.523 %	6.063 Mb/s

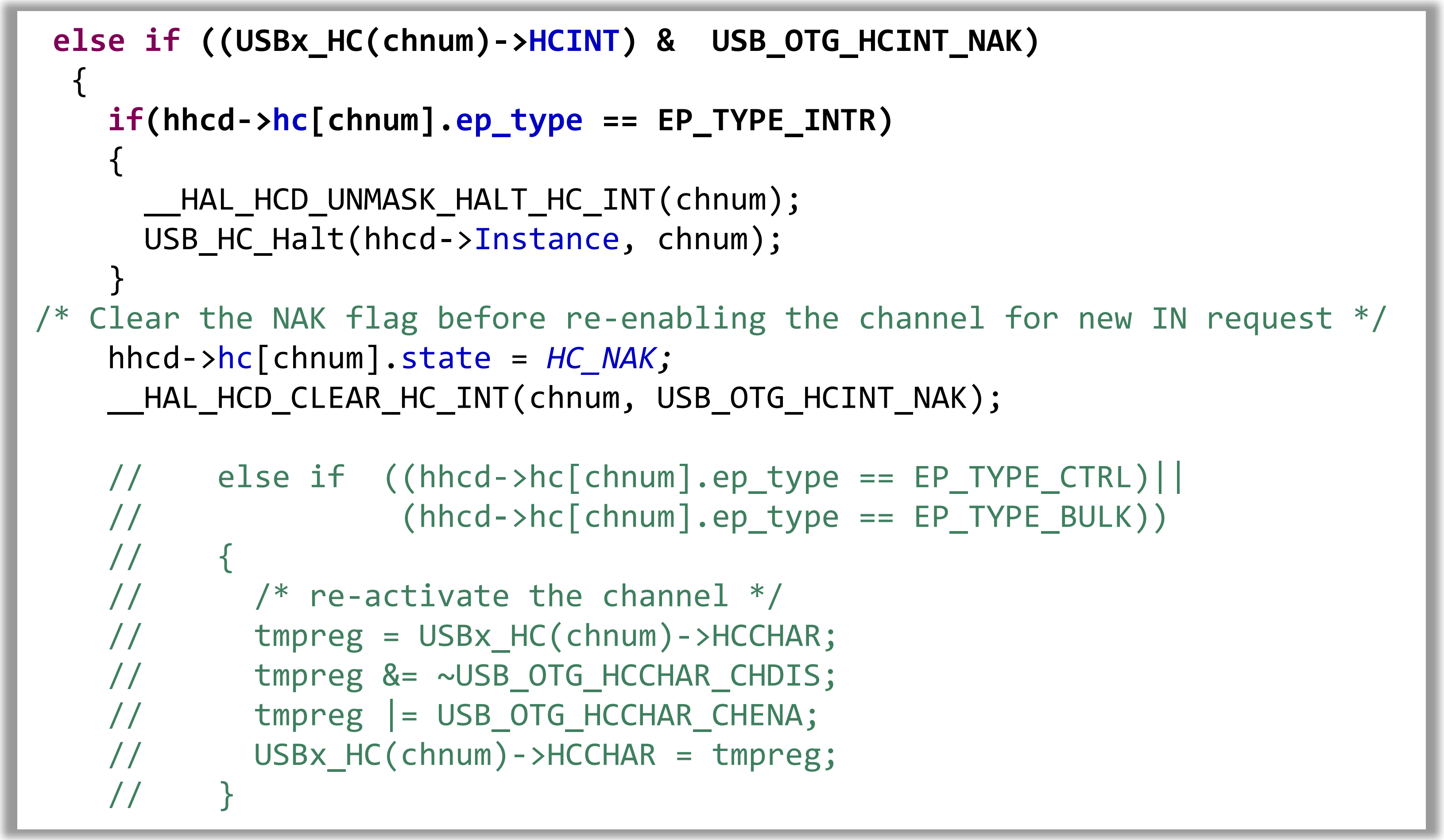
life.augmented

USB VCP Host

- High load on the bus mean also high interrupt count on the MCU – interrupt is received with each NAK, and in the interrupt another IN transfer is set.
- This can be modified in the library – no more activating of IN request after NAK-> for each call of USBH_CDC_Receive function only one packet or NAK is received

USB VCP Host

- Modify HCD_HC_IN_IRQHandler in stm32f4xx_hal_hcd.c to disable automatic NAK retransmit

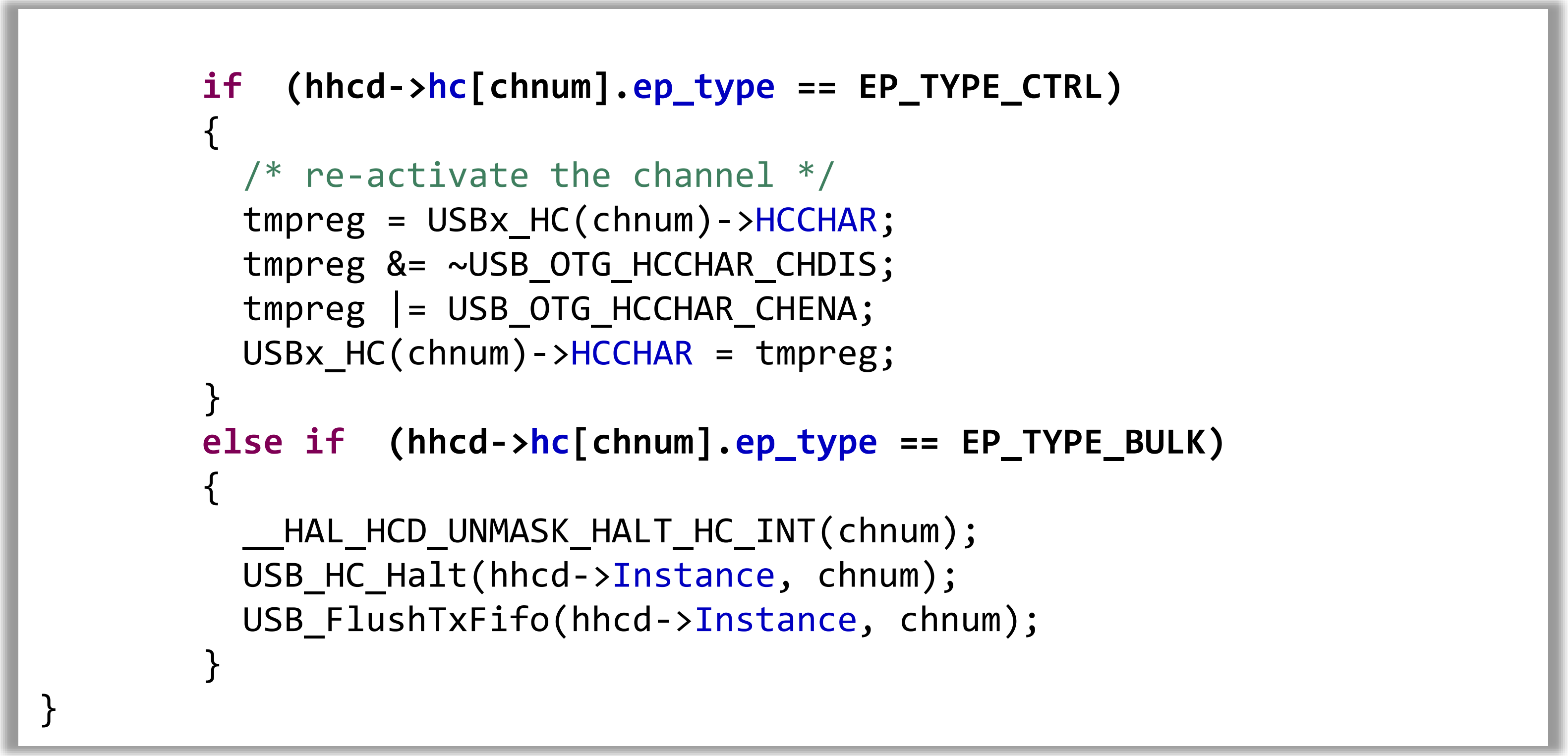


```
 else if ((USBx_HC(chnum)->HCINT) &  USB_OTG_HCINT_NAK)
  {
    if(hhcd->hc[chnum].ep_type == EP_TYPE_INTR)
    {
      __HAL_HCD_UNMASK_HALT_HC_INT(chnum);
      USB_HC_Halt(hhcd->Instance, chnum);
    }
/* Clear the NAK flag before re-enabling the channel for new IN request */
    hhcd->hc[chnum].state = HC_NAK;
    __HAL_HCD_CLEAR_HC_INT(chnum, USB_OTG_HCINT_NAK);

    //    else if  ((hhcd->hc[chnum].ep_type == EP_TYPE_CTRL)||
    //              (hhcd->hc[chnum].ep_type == EP_TYPE_BULK))
    //    {
    //      /* re-activate the channel */
    //      tmpreg = USBx_HC(chnum)->HCCHAR;
    //      tmpreg &= ~USB_OTG_HCCHAR_CHDIS;
    //      tmpreg |= USB_OTG_HCCHAR_CHENA;
    //      USBx_HC(chnum)->HCCHAR = tmpreg;
    //    }
```

USB VCP Host

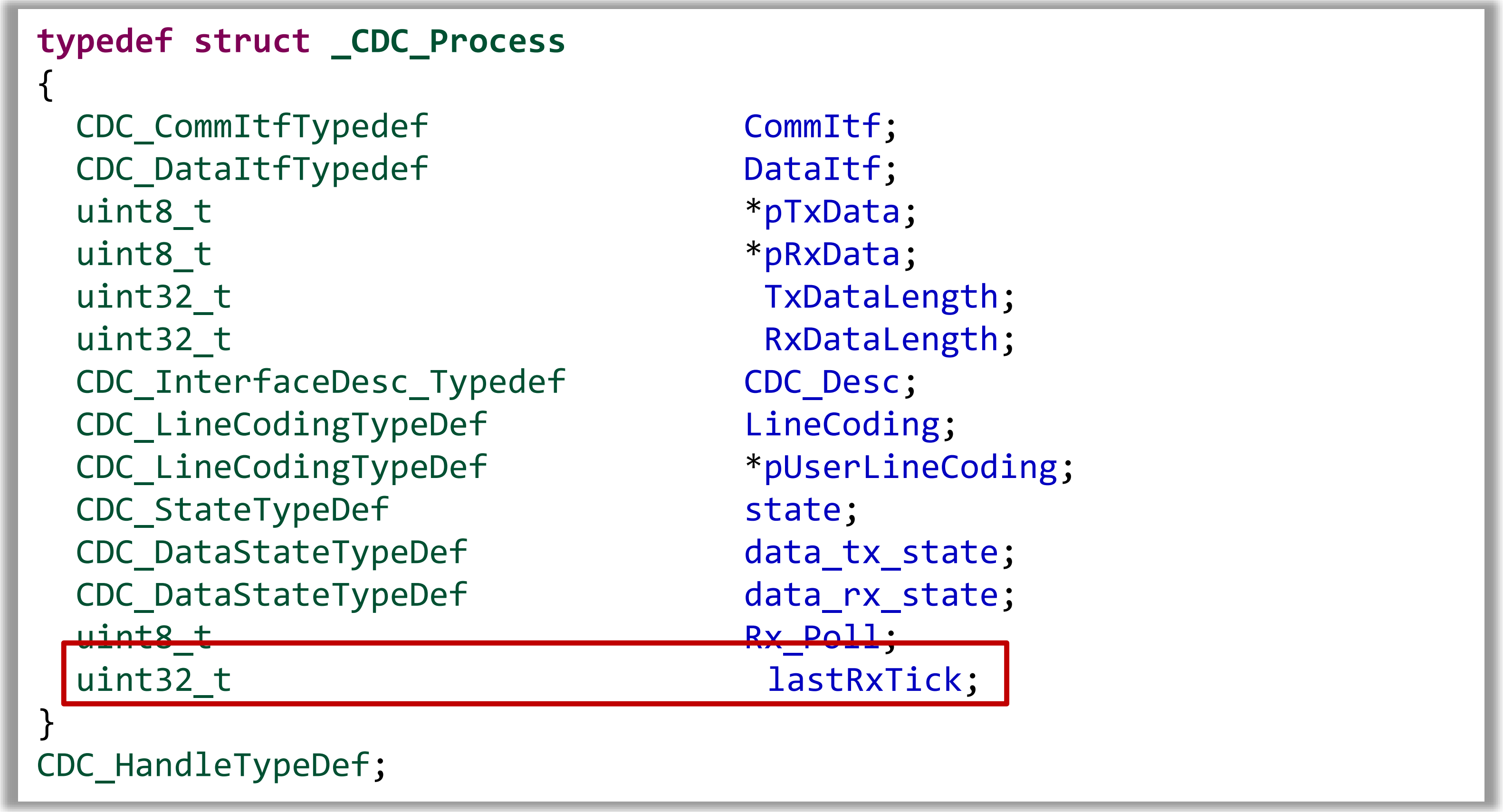
- Modify HCD_HC_IN_IRQHandler in stm32f4xx_hal_hcd.c to disable automatic NAK retransmit



```
        if  (hhcd->hc[chnum].ep_type == EP_TYPE_CTRL)
        {
          /* re-activate the channel */
          tmpreg = USBx_HC(chnum)->HCCHAR;
          tmpreg &= ~USB_OTG_HCCHAR_CHDIS;
          tmpreg |= USB_OTG_HCCHAR_CHENA;
          USBx_HC(chnum)->HCCHAR = tmpreg;
        }
        else if  (hhcd->hc[chnum].ep_type == EP_TYPE_BULK)
        {
          __HAL_HCD_UNMASK_HALT_HC_INT(chnum);
          USB_HC_Halt(hhcd->Instance, chnum);
          USB_FlushTxFifo(hhcd->Instance, chnum);
        }
}
```

USB VCP Host

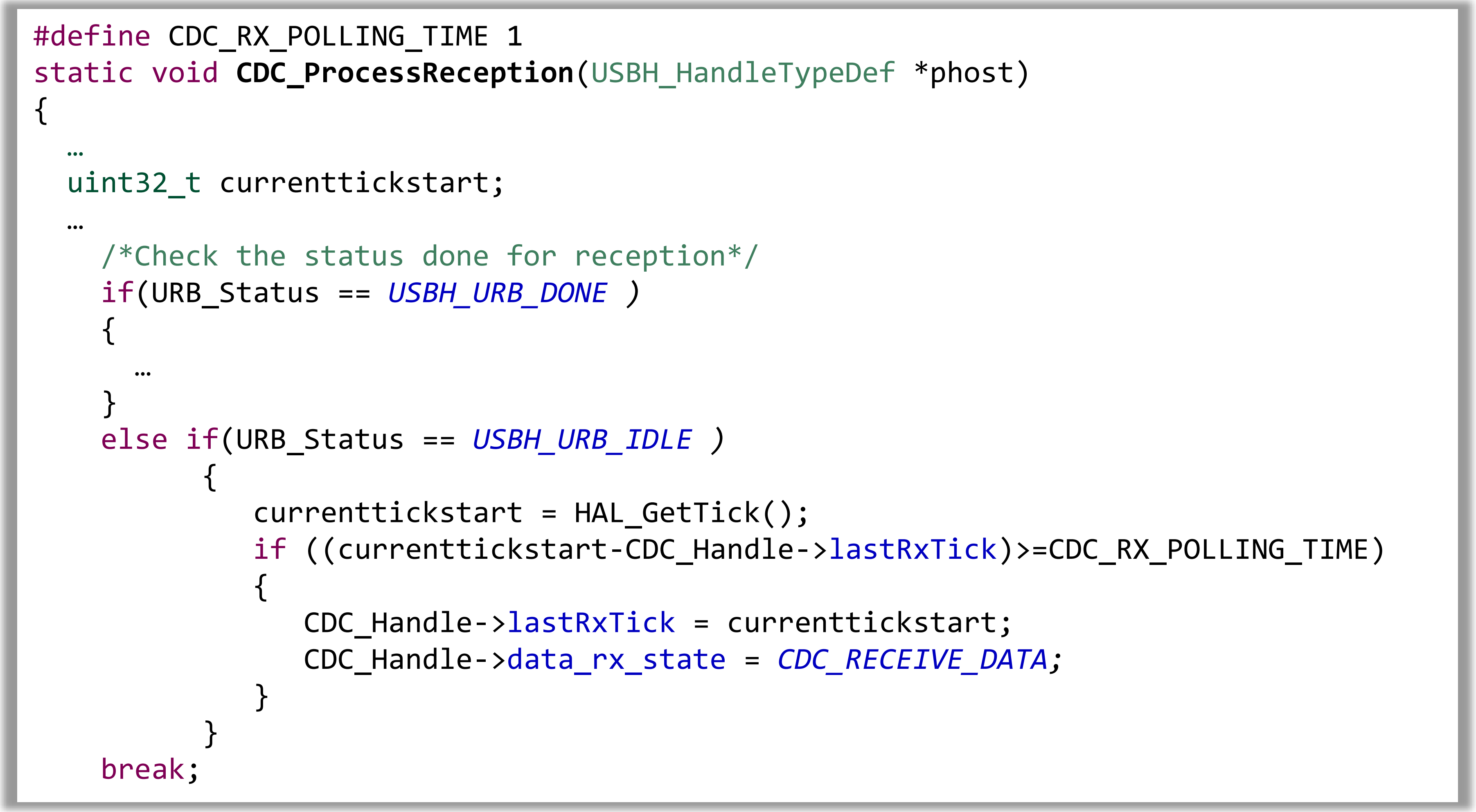
- Now different way how to ask periodically for data need to be added
  - CDC_ProcessReception function will be used, but first CDC_Process structure inside usbh_cdc.h needs to be modified to keep information about last IN request sent



```
typedef struct _CDC_Process
{
  CDC_CommItfTypedef                CommItf;
  CDC_DataItfTypedef                DataItf;
  uint8_t                           *pTxData;
  uint8_t                           *pRxData;
  uint32_t                           TxDataLength;
  uint32_t                           RxDataLength;
  CDC_InterfaceDesc_Typedef         CDC_Desc;
  CDC_LineCodingTypeDef             LineCoding;
  CDC_LineCodingTypeDef             *pUserLineCoding;
  CDC_StateTypeDef                  state;
  CDC_DataStateTypeDef              data_tx_state;
  CDC_DataStateTypeDef              data_rx_state;
  uint8_t                           Rx_Poll;
  uint32_t                           lastRxTick;
}
CDC_HandleTypeDef;
```

USB VCP Host

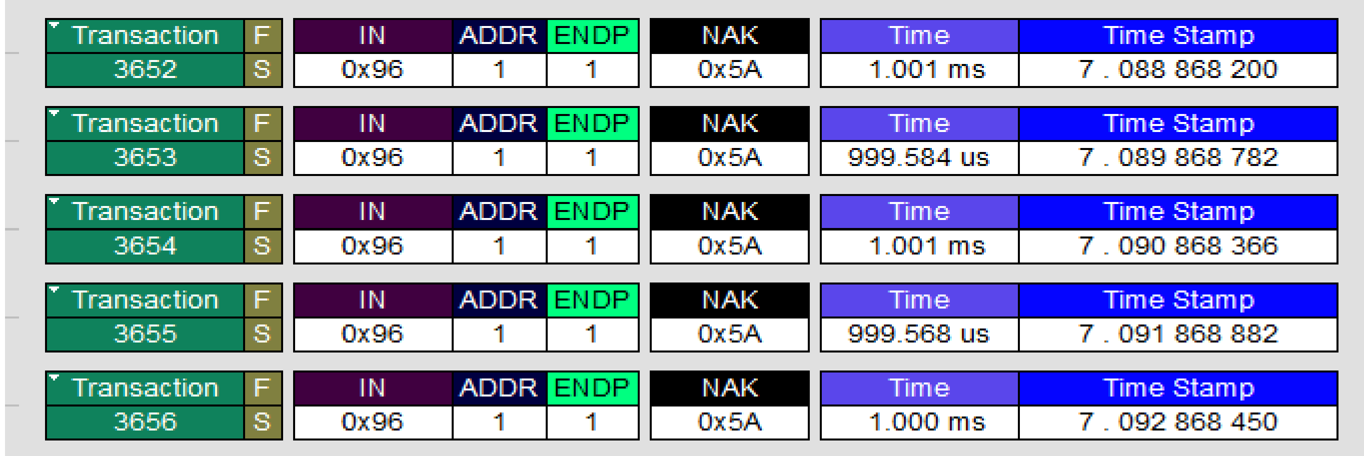
- Now modify CDC_ProcessReception function inside usbh_cdc.c



```
#define CDC_RX_POLLING_TIME 1
static void CDC_ProcessReception(USBH_HandleTypeDef *phost)
{
  …
  uint32_t currenttickstart;
  …
    /*Check the status done for reception*/
    if(URB_Status == USBH_URB_DONE )
    {
      …
    }
    else if(URB_Status == USBH_URB_IDLE )
          {
             currenttickstart = HAL_GetTick();
             if ((currenttickstart-CDC_Handle->lastRxTick)>=CDC_RX_POLLING_TIME)
             {
                CDC_Handle->lastRxTick = currenttickstart;
                CDC_Handle->data_rx_state = CDC_RECEIVE_DATA;
             }
          }
    break;
```

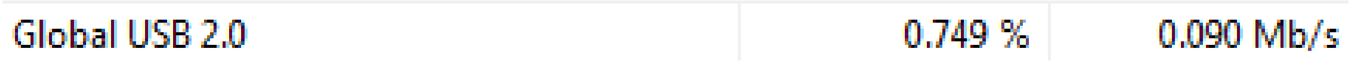
USB VCP Host

- Now the number of IN requests can be set by application
- With lower IN requests frequency throughput of the system is decreasing



Transaction	F S	IN	ADDR	ENDP	NAK	Time	Time Stamp
3652	S	0x96	1	1	0x5A	1.001 ms	7 . 088 868 200
3653	S	0x96	1	1	0x5A	999.584 us	7 . 089 868 782
3654	S	0x96	1	1	0x5A	1.001 ms	7 . 090 868 366
3655	S	0x96	1	1	0x5A	999.568 us	7 . 091 868 882
3656	S	0x96	1	1	0x5A	1.000 ms	7 . 092 868 450

VCP_host_NAK_reduced.usb



Global USB 2.0 0.749 % 0.090 Mb/s