Topic:	Processing Touch UART CubeIDE
Author:	PARA
Date:	30.11.2021

0. Before you start

This document will give you an overview of the source code for the example package Touch UART. Before you can work with it you need to set up your working environment as explained in the document “examplepackage_cubeide_gettingstarted”. Make sure you have read this document beforehand and executed all the steps to configure your STM32CubeIDE.

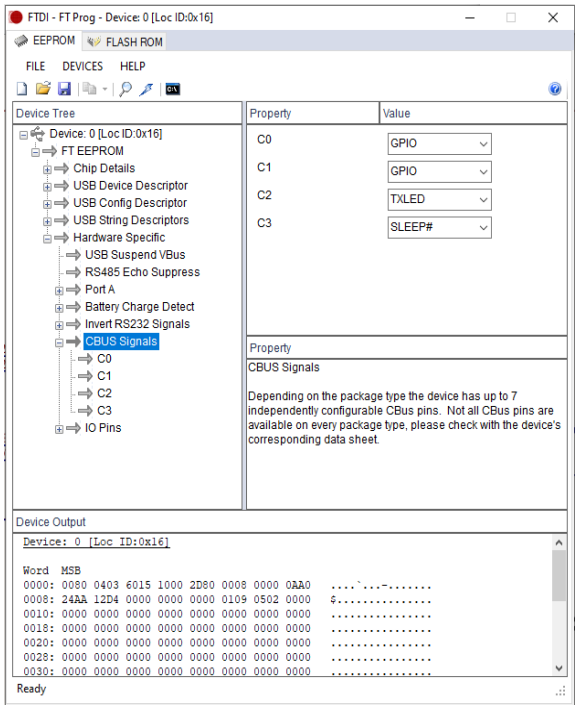
For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) by STMicroelectronics.

1. Introduction

In this example package we will work with the touch panel of our display module, which is mounted on top of the LCD-module. The touch panel has its own control board which is connected to the microcontroller via I²C. When you touch the display, you can get the position of this touch event. The same applies to a release event. This information, the position and the type of event, is then sent over UART to the on-board FTDI-chip, which converts the data input to the USB-format and sends the data through the USB-Type-B socket to a connected PC.

2. Additional Required Software

2.1 FT_PROG

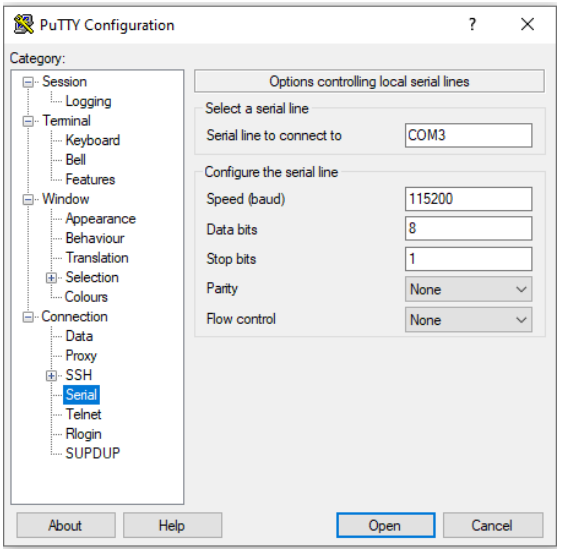


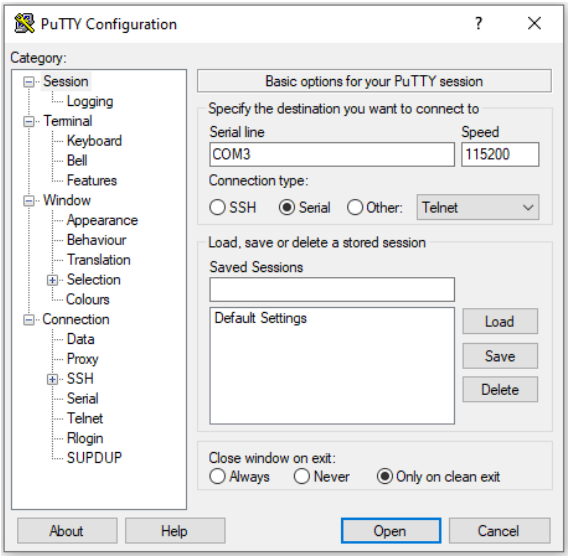
You may or may not need to configure the FTDI-chip. To make sure, that the chip is configured correctly you should download the free EEPROM programming utility FT_PROG by FTDI (https://ftdichip.com/utilities/). Connect your computer with a USB-Type-A to USB-Type-B cable to the display board and make sure that the board is connected to its power supply. Start FT_PROG and click on the button with the magnifying glass. This will start the scan process for connected devices. Then select *Hardware Specific -> CBUS Signals*. Now check if values for the properties C0 and C1 on the right panel are set to *GPIO*. If not, change this. Afterwards click on the lightning button and click *Program* to start the programming process. After the programming has ended successfully, turn the power off, wait a few seconds and turn it on again. The FTDI-chip should now be configured properly.

2.2 PuTTY

To read the data, that is transmitted by the microcontroller to the PC, you need a program to establish the required serial connection. We recommend the free software PuTTY for this. When you opened the program, you first need to configure the serial connection. Click on the

tab *Serial* and configure the serial line with the values you see in the left image below. They should match with the values we used to initialize the UART interface of the microcontroller (see */src/main.c* -> `static void MX_USART3_UART_Init(void)`). Next, click on the tab *Session* and choose the connection type *Serial*. The other parameters will change automatically. Now, you can press *Open* to start the connection. A terminal window should pop up.





3. Explanation of Example Code

3.1 Main function

We will start our walk through the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the hardware abstraction layer (HAL) and the system clocks are initialized. Then all the peripherals are initialized. A variable is declared for later use.

The 3.5- and 5.0-inch displays need the pin PH6 to be set to high in order to enable the backlight. Those two displays are recognized by the define BACKLIGHT_EN which is defined in the global_Display_Touch_HAL.h file. After that, the display startup sequence, consisting of filling the buffers/layers with the color white and displaying our 2 logos, is started.

Following the short delay of the display sequence the active display buffer is filled with the color 0xFFFFFFFF (ARGB8888) as background. Now the string “Touch Me!” is drawn to the display. In the main while-loop the application code function:

```
void process_touch_input(Buffer_e touchMeBuffer)
```

will be called continuously. This function will be explained later. Important to know is that the touch point updating procedure is done via interrupt. You can see that in the file *Core/Src/stm32f4xx_it.c* in the function `void EXTI3_IRQHandler(void)`. This is an external interrupt function, which will be called every time the configured pin (PD3) is low. This corresponds to the touch controller pulling the pin low and signaling an incoming touch event.

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USART6_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  /* Declaration of variables */
  Buffer_e activeBuffer;

  /* Backlight enable */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_9, GPIO_PIN_SET);
  if (BACKLIGHT_EN) {
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_SET);
  }
  /* Display the logos and save active buffer */
  activeBuffer = Display_Startup_Sequence();
  DMA2D_Fill_Color(0xFFFFFFFF, Layer_1, activeBuffer, true);
  /* Write "Touch me!" into the current Buffer */
  DMA2D_write_string("Touch Me!", (HDP-9*16)/2, VDP/2-11, 0xFF000000, &courier_new, Layer_1, activeBuffer, true);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* Check if User touched the display and send touch event via UART */
    process_touch_input(activeBuffer);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

3.2 Function: process_touch_input(Buffer_e touchMeBuffer)

```c
void process_touch_input(Buffer_e touchMeBuffer){
  /* Declaration of variables */
  static uint8_t state = RELEASE;
  TouchXY_t* txy;
  static int touch_snd = 0;
  static uint64_t timeout;

  /* Debounce */
  if (touch_snd) {
    if(HAL_GetTick() - timeout > 500)
      return;
    touch_snd = 0;
  }

  /* Get address of touch structure */
  txy = return_touch_struct();
  /* Check for touch event */
  if (txy->event == TOUCH) {
    if (state != TOUCH) {
      state = TOUCH;
      /* Send response via UART */
      send_touch_response(txy);
      touch_snd = 1;
      if(AreaCheck(0, 0, HDP/2, VDP/2)){
        DMA2D_Fill_Color(0xFF00FF00, Layer_1, !touchMeBuffer, true);
        LTDC_Switch_Buffer(!touchMeBuffer);
      }
      else{
        DMA2D_Fill_Color(0xFFFF0000, Layer_1, !touchMeBuffer, true);
        LTDC_Switch_Buffer(!touchMeBuffer);
      }
      timeout = HAL_GetTick();
    }
  }

  /* Check for release event */
  else if (txy->event == RELEASE) {
    if (state != RELEASE) {
      state = RELEASE;
      /* Send response via UART */
      send_touch_response(txy);
      touch_snd = 1;
      timeout = HAL_GetTick();
      LTDC_Switch_Buffer(touchMeBuffer);
    }
  }

  /* Check for error event */
  else if (txy->event == 0xFF) {
    /* Send response via UART */
    send_touch_response(txy);
    timeout = HAL_GetTick();
  }
}
```

This function fulfills the main task of this example code. This function will be called repeatedly and check the touch event structure “txy” for the current touch event. If a touch event occurs it checks if its different from the last one and handles the event. If an event happens, the function will send a string via UART.

The function will also check via AreaCheck if the touch event was inside the upper left quarter of the display. If true it will fill the background buffer with green and display it, otherwise it will fill the buffer with red and display it.

After the user releases the display, the touchMeBuffer will be displayed again. That is the buffer with the string “Touch me!” written into it.

3.3 Function: send_touch_response

```
75  void send_touch_response(TouchXY_t* touch_resp){
76    char output[49];
77
78    if(touch_resp->event == TOUCH){
79      sprintf(output, "X,Y-Coordinates: (%3i,%3i) EVENT-TYPE:   TOUCH\r\n", touch_resp->xPos, touch_resp->yPos);
80    }
81    else if (touch_resp->event == RELEASE){
82      sprintf(output, "X,Y-Coordinates: (%3i,%3i) EVENT-TYPE: RELEASE\r\n", touch_resp->xPos, touch_resp->yPos);
83    }
84    else if (touch_resp->event == 0xFF){
85      sprintf(output, "X,Y-Coordinates: (---,---) EVENT-TYPE:   ERROR\r\n");
86    }
87
88    HAL_UART_Transmit(&huart1, (uint8_t*) output, (uint16_t) strnlen(output, 49), HAL_MAX_DELAY);
89
90  }
```

This function packs the information about the touch/release event, which is stored in the *touch_resp_t* pointer, into a string. This string is then sent via UART to the PC.