پورت سریال STM32 یا UART با توابع HAL - ارسال به روش BLOCKING

تاریخ انتشار۲۰ بهمن, ۱۴۰۲ توسط سید حسین سلطانی

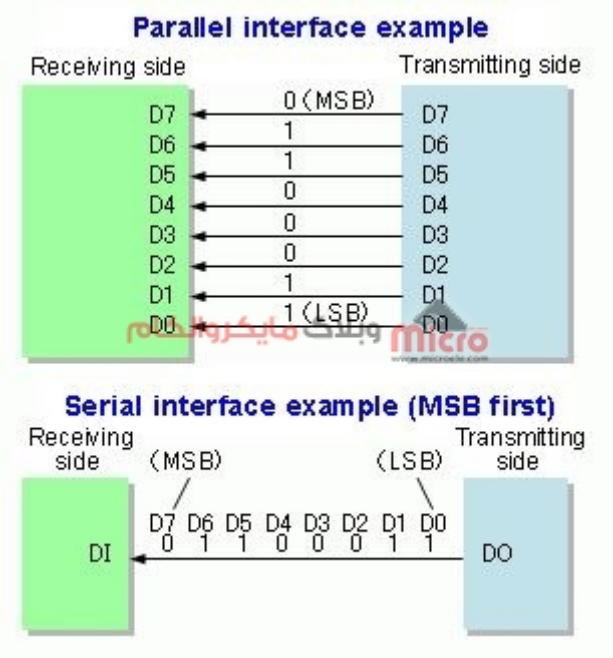
سلام خدمت همه شما مایکروالکامی ها. در مطلب قبلی از سری مطالب معرفی قطعات به بررسی خازن های X و Y پراخته شد. در این مطلب به بررسی ارتباط پورت سریال (UART/USART) و ارسال دیتا از طریق سریال با استفاده از توابع HAL در میکروکنترلر STM32 پرداخته خواهد شد. پس با من تا انتهای مطلب همراه باشید. همچنین شما میتونید سایر مطالب من رو از این لینک مطالعه و بررسی کنید.

مقدمه

ارتباط سریال (UART/USART) یکی از پریفرال های مطرح و پر کاربرد در الکترونیک و راه ارتباطی برای ارسال و دریافت دیتا بین میکروکنترلر ها (مانند STM32 و AVR) و ماژول هایی همانند GSM، WiFi و... می‌باشد. مهمترین عامل جهت ارتباط درست و موثر در ارتباط سریال دانستن نیاز های پروژه و داشتن نرخ تبادل دیتا (baud rate) برابر بین دو دستگاه می‌باشد. در این مطلب نحوه ارسال دیتا در بستر ارتباط سریال UART بررسی شده در مطلب بعدی دریافت دیتا مورد بررسی خواهد گرفت

روش ارتباط موازی (Parallel) و سریال (Serial)

در ارسال و دریافت دیتا دو روش موازی و سریال وجود دارد. در روش موازی فرآیند ارتباط از طریق چند کانال ارتباطی بصورت همزمان انجام می‌شود. در نتیجه سرعت تبادل دیتا زیاد بوده اما هزینه کابل کشی و نویز پذیری آن بالا خواهد بود. در روش سریال از یک مسیر (سیم) برای تبادل دیتا استفاده می‌شود. در این بستر، دیتا یکی یکی برروی کانال ارتباطی ارسال/دریافت می‌شود. از این رو این ارتباط برای مسافت های طولانی تر مناسب می‌باشد.



تبادیل دیتا به روش موازی و سریال

ارتباط سریال (USART/UART)

در ارتباط سریال دو عنوان UART به معنی ارتباط غیر همزمان (Universal Asynchronous Received/Transmitter) و USART به معنی ارتباط همزمان (Universal Synchronous Asynchronous Received/Transmitter) وجود دارد. در جدول زیر می‌توان بصورت خلاصه تفاوت این دو نوع ارتباط را بررسی کرد.

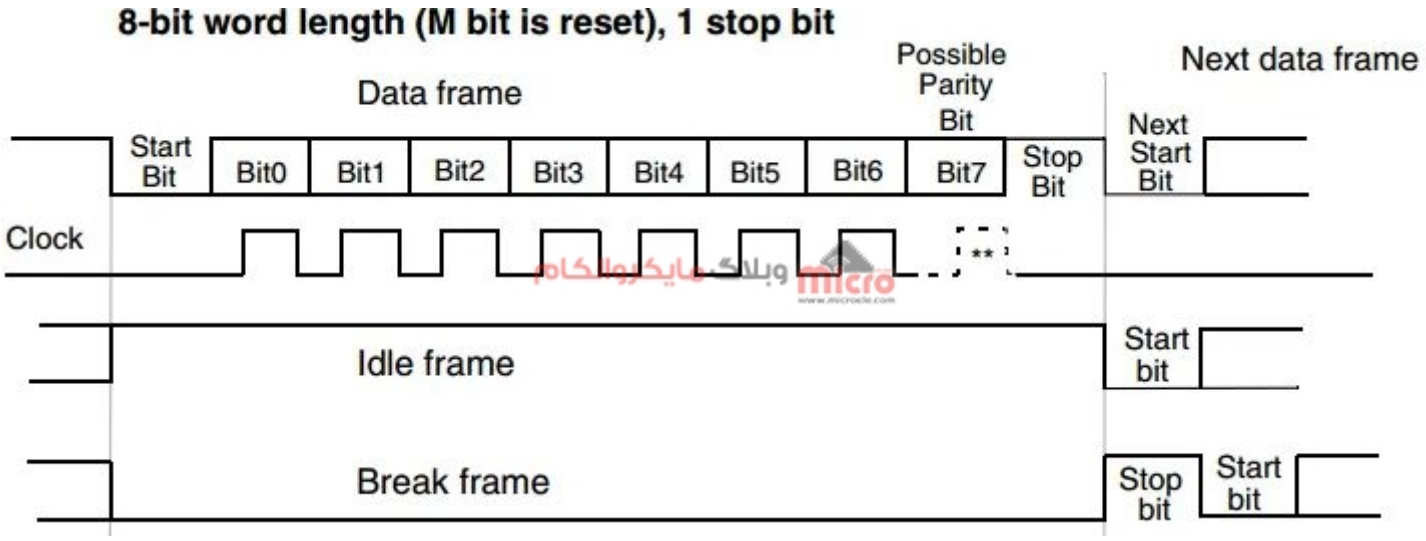
	UART	USART
حالت ارتباط	تمام دو طرفه (Full duplex)	نیم دو طرفه (Half duplex)
سیگنال/پایه های مورد نیاز	2 (Tx و Rx)	4 (شامل Tx، Rx، XCK یا clock و XDIR یا direction جهت)
نرخ تبادل دیتا	قابل تعریف بین فرستنده و گیرنده	قابل تعریف برروی کلاک پالس روی پایه XCK

برای برقراری ارتباط سریال به دو پایه از میکروکنترلر (AVR، STM32 و ...) به نام Rx و Tx نیاز داریم. نحوه اتصال این

پایه ها بین گیرنده و فرستنده بصورت ضربدری می‌باشد. یعنی پایه Tx فرستنده به Rx گیرنده و Rx فرستنده به Tx گیرنده باید متصل شود. علاوه بر این باید باودریت گیرنده و فرستنده نیز برابر بوده تا ارتباط سریال به درستی انجام پذیرد.

پکت بندی دیتا در ارتباط سریال

ارتباط دیتا در سریال می‌تواند بصورت 8 یا 9 بیتی باشد. برای مشخص کردن نوع 8 یا 9 بیتی بودن باید در رجیستر مربوطه این کار را انجام داد. یک پکت دیتا برای تبادل مشابه تصویر زیر می‌باشد. در ابتدای آن یک بیت به عنوان بیت آغاز (Start bit)، 8 بیت به عنوان دیتا مورد نظر، یک بیت به عنوان Parity، یک تا 2 بیت برای بیت پایان (Stop bit) در نظر گرفته شده است.



پکت دیتا در ارتباط سریال

پکت دیتا در ارتباط سریال

فریم پکت سریال

یک فریم دیتا شامل دیتا اصلی جهت ارسال بوده و در صورت استفاده از بیت Parity (برابری) می‌تواند بین 5 تا 8 بیت باشد. در صورت عدم استفاده از بیت Parity، می‌تواند فریم دیتا تا 9 بیت باشد. بیت برابری مشخص کننده فرد یا زوج بودن است. از این بیت برای اطمینان از دریافت صحیح دیتا در ارتباط سریال (UART/USART) استفاده می‌شود.

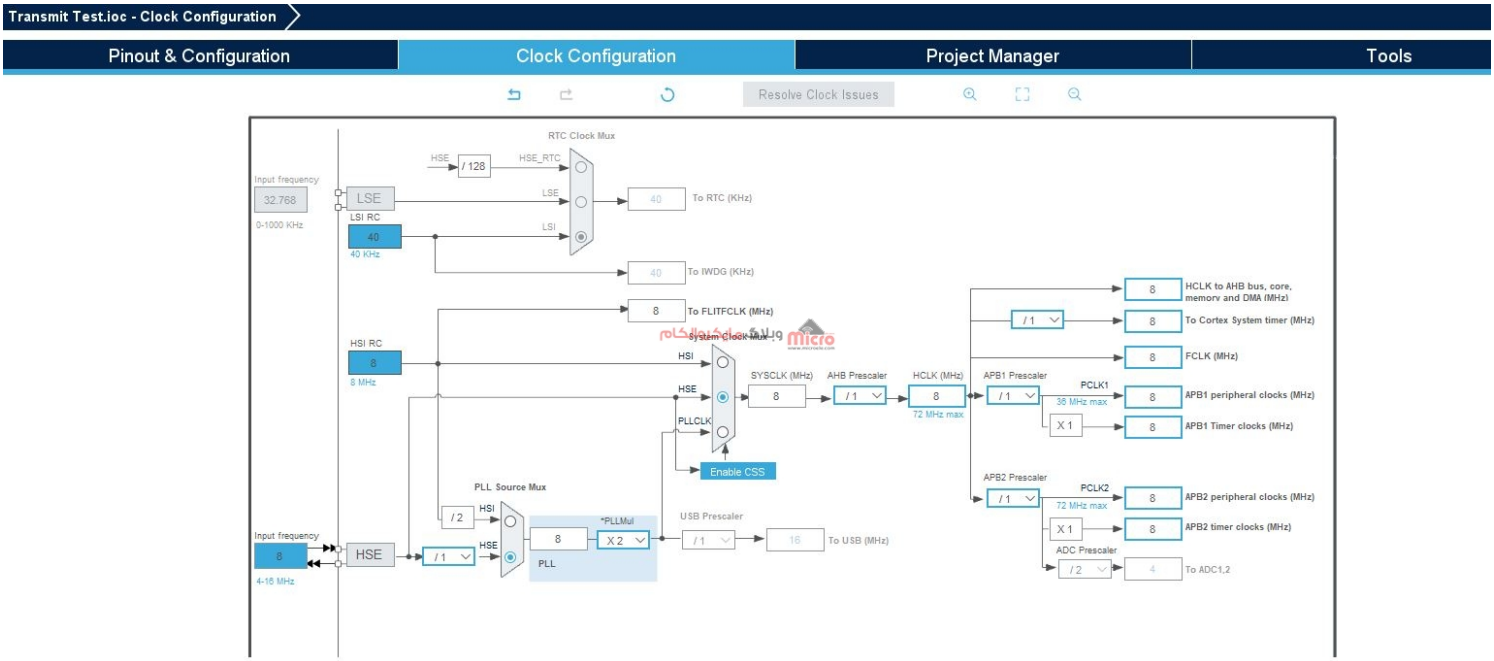
پس از دریافت دیتا، تعداد بیت هایی که منطق 1 داشته اند شمارش می‌شود. سپس بررسی می‌شود تعداد آن زوج است یا فرد. درصورتیکه بیت Parity صفر تنظیم شده باشد به معنی برابری یا تعادل زوج و اگر 1 باشد به معنی برابری یا تعادل فرد می‌باشد. حال اگر تعادل زوج یا فرد در دیتا دریافتی با بیت تعادل برابر باشد، دریافت بدون اشکال است اما اگر مثلا بیت تعادل 0 (تعادل زوج) باشد و تعداد 1 های دیتا فرد باشد یا بالعکس، نشان دهنده اشکال در دریافت دیتا می‌باشد.

ارسال از طریق پورت سریال UART/USART در STM32 به روش Polling (سرکشی)

در این مطلب برای ارسال دیتا از توابع HAL استفاده می‌کنیم. در ابتدا یک پروژه جدید ایجاد کرده و میکروکنترلر مورد نظر (در این مطلب STM32F103C8T6) را انتخاب و تنظیمات آن را مطابق مراحل زیر انجام می‌دهیم.

تنظیمات کلاک

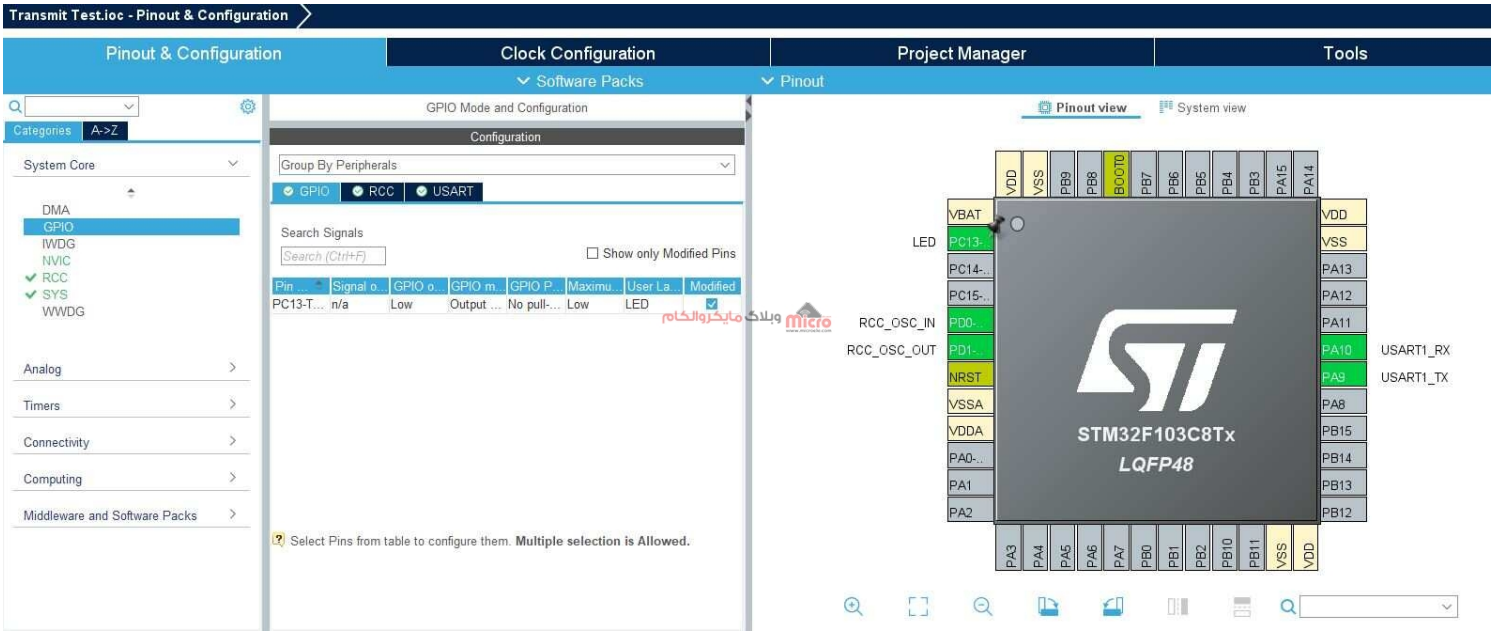
بعد از باز شدن پنجره Mx از بخش System Core وارد RCC شده و منبع کلاک (HSE) را کریستال خارجی انتخاب و نهایتا در بخش Clock Configuration فرکانس کاری میکرو را مطابق با نیاز انتخاب نمایید.



تنظیم کلاک در *STM32* برای ارتباط سریال

تنظیمات GPIO

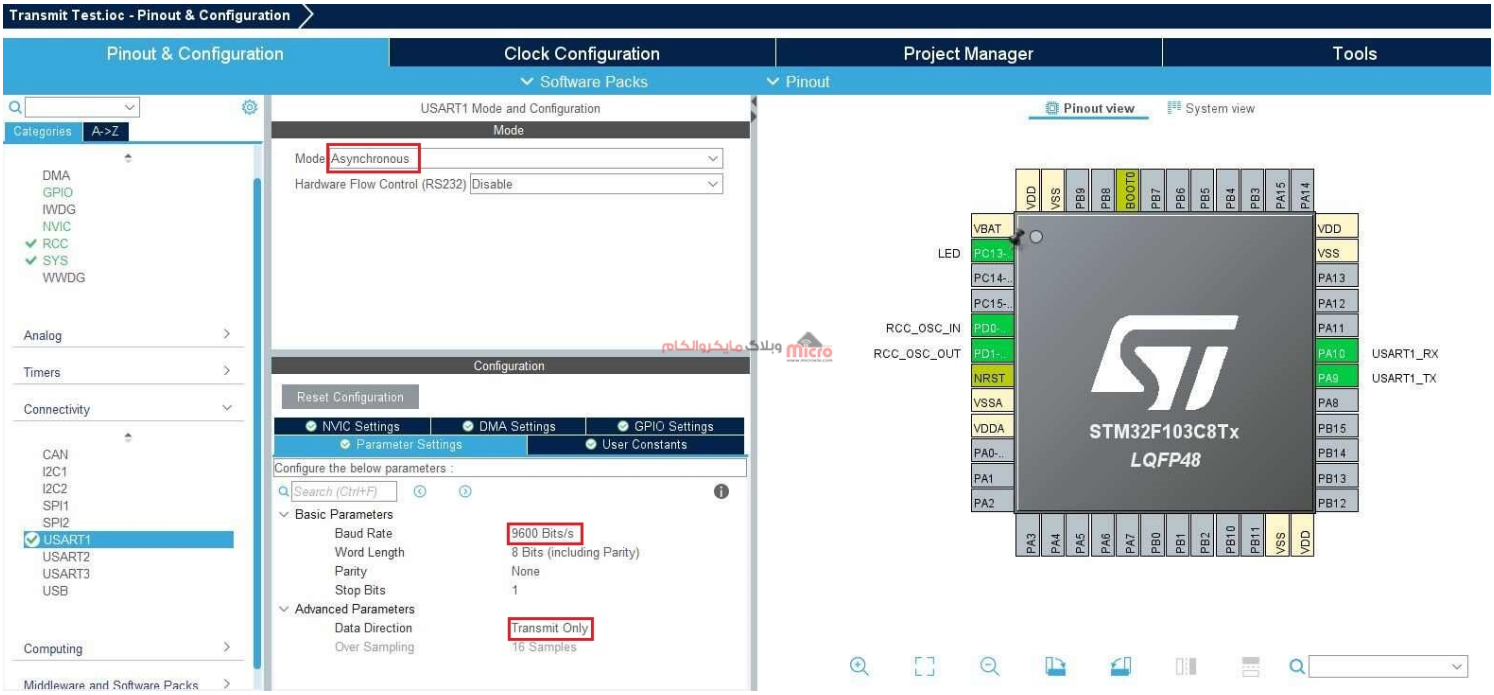
در Pinout & Configuration وارد System Core و GPIO شده و یک پایه را به دلخواه برای اتصال یک LED و چشمک زدن آن انتخاب می‌کنیم. در تصویر زیر PC13 انتخاب شده و یک label با نام LED به آن اختصاص داده شده است.



تنظیمات *GPIO* در *STM32*

تنظیمات اولیه پورت سریال

سپس به بخش Pinout & Configuration بازگشته و از بخش Connectivity گزینه UART1 را انتخاب نمایید. در ابتدا Mode را به حالت Asynchronous تغییر داده و در بخش Parameters Settings باودریت مورد نظر و تنظیمات دلخواه (مطابق با نیاز پروژه) را انجام می‌دهیم.



تنظیمات اولیه بخش سریال *UART* در *STM32*

ایجاد کد و ساخت فایل main.c

نهایتا پس از انجام مراحل فوق از نوار ابزار بالا صفحه، گزینه Project را انتخاب و با زدن گزینه Generate Code پروژه را ذخیره و نسبت به ساخت فایل های مورد نیاز و باز شدن main.c اقدام می‌کنیم. حال در فایل main.c در حلقه while(1) علاوه بر چشمک زن کردن پایه PC13، با استفاده از دستور زیر دیتا را از طریق پورت سریال ارسال خواهیم کرد.

```
HAL_UART_Transmit(huart, pData, Size, Timeout);
```

تابع بالا در روش سرکشی (polling) استفاده شده و به روش blocking نیز معروف است. ورودی تابع بالا تشکیل شده از چند بخش زیر است.

- **ورودی اول:** اشاره گر (pointer) به ساختار شامل تنظیمات UART میکروکنترلر مثلا huart1 که در قسمت /* ---- Private variables*/ در برنامه اصلی در main.c مشخص شده است.
- **ورودی دوم:** اشاره گر به آرایه uint8_t بوده که شامل دیتا ارسالی است.

- **ورودی سوم:** یک عدد uint16_t بوده که مشخص کننده تعداد بایت های ارسالی آرایه دیتا ارسالی است. می‌توان از توابع sizeof یا strlen به این منظور استفاده کرد.
- **ورودی چهارم:** بازه زمانی برحسب میلی ثانیه و بیانگر مدت زمان ارسال دیتا است. دقت شود این زمان را خیلی کم در نظر نگرفته (باعث عدم ارسال کامل دیتا) و معمولا 100ms مناسب می‌باشد.

نکته: در صورت استفاده از تابع strlen() در ابتدای برنامه باید کتابخانه string.h را معرفی کرد.

ارسال دیتا

برای ارسال یک پیام مثلا "STM32 UART Transmitting Test" ابتدا یک بافر جهت ذخیره سازی این رشته و نهایتا از تابع بالا جهت ارسال آن استفاده می‌کنیم.

```
uint8_t data[] = "STM32 UART Transmitting Test\r\n";
HAL_UART_Transmit(&huart1, data, strlen((char *)data), 10);
```

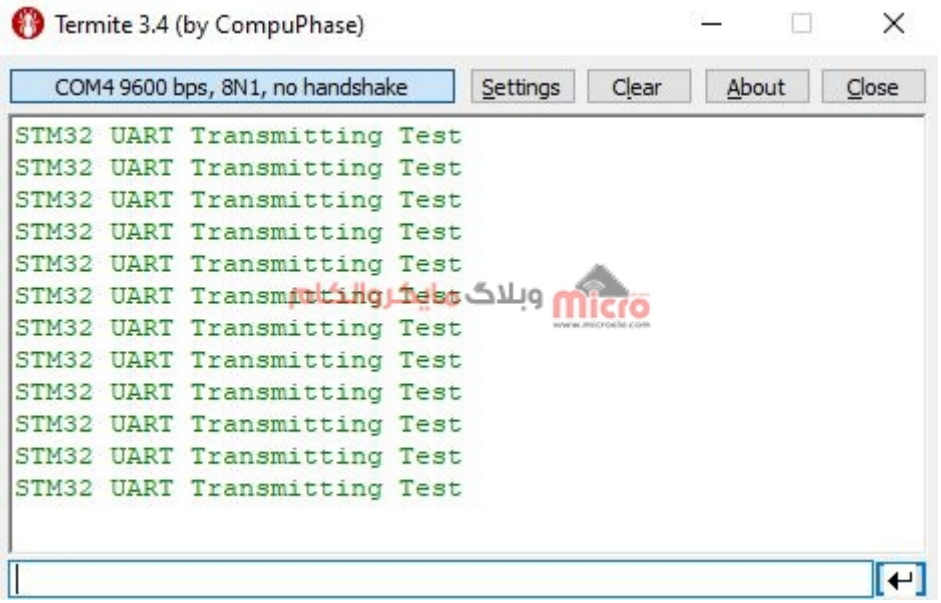
برنامه ارسال دیتا در UART به روش polling در STM32

```
#include "main.h"
#include <string.h>

UART_HandleTypeDef huart1;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);

uint8_t data[] = "STM32 UART Transmitting Test\r\n";

int main(void)
{
```

```
  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();
  MX_USART1_UART_Init();

  while (1)
  {

      HAL_UART_Transmit(&huart1, data, strlen((char *)data), 100);
      HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
      HAL_Delay(1000);
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();

  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);
}

void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {
  }
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```



ارسال دیتا در پورت سریال بااستفاده از *STM32* و توابع *HAL*

اشکال روش polling در ارسال دیتا در سریال UART

همانطور که پیش تر ذکر شد به این روش blocking نیز گویند و باعث درگیر کردن CPU می‌شود. در این حالت در صورتیکه بخواهیم یک دیتا با تعداد بایت زیاد را ارسال کنیم، از آنجا که بایستی timeout را مشخص کنیم، باعث درگیری CPU شده و دستورات بعد این تابع ارسال با تاخیر به اندازه مقدار timeout اجرا خواهد شد. لذا برای حل این مشکل از روش Interrupt (وقفه) یا DMA استفاده خواهد شد.

نتیجه گیری

برای راه اندازی ارتباط سریال (UART/USART) در STM32 دو روش کلی blocking mode (شامل polling) و non-blocking mode (شامل Interrupt و DMA) وجود دارد. در صورت استفاده از روش polling در صورتی که timeout را زیاد کنیم یا بخواهیم دیتای طولانی تری ارسال کنیم، شاهد درگیری CPU در این قسمت از کد شده و از انجام کد های بعدی تا تمام شدن این زمان به نوعی جلوگیری می‌شود. لذا درگیر شدن CPU در این روش را خواهیم داشت و برای برخی از نیاز ها

توصیه نمی‌شود. در این صورت از روش های وقفه یا DMA می‌توان استفاده کرد.

امیدوارم از این مطلب کمال بهره را برده باشید. در صورت داشتن هرگونه نظر یا سوال درباره این مطلب یا تجربه مشابه اون رو در انتهای همین صفحه در قسمت دیدگاه ها قرار بدید. در کوتاه ترین زمان ممکن به اون ها پاسخ خواهم داد. اگر این مطلب براتون فید بود، اون رو به اشتراک بگذارید تا سایر دوستان هم بتوانند استفاده کنند. همینطور میتونید این مطلب را توی اینستاگرام با هشتگ #microelecom به اشتراک بگذارید و پیج مایکروالکام (@microelecom) رو هم منشن کنید.