Bài 5. Lập trình UART cho STM32

UART là một trong những chuẩn giao tiếp được sử dụng nhiều nhất trong lập trình nhúng. Ưu điểm của nó là sự đơn giản trong cách sử dụng, tuy nhiên chuẩn giao tiếp này có tốc độ khá thấp (115200bit/s) so với các chuẩn truyền khác (SPI, I2C, USB) và khoảng cách cho phép truyền cũng tương đối gần.

Tại sao tốc độ thấp và khoảng cách truyền gần như vậy, UART không bị loại bỏ so với các chuẩn giao tiếp có nhiều ưu điểm hơn ?

1. Lý thuyết

Để hiểu rõ hơn về chuẩn giao tiếp UART, các bạn có thể tham khảo bài viết tại đây (link).

Đối với dòng chip STM32F4 sẽ có một số thanh ghi quan trọng để dễ dàng cho việc Debug chương trình.

Một số thanh ghi quan trọng

USART_CR1

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PEIE: cho phép ngắt PE hay không ngắt.

TXEIE: cho phép ngắt truyền hay không.

RXNEIE: cho phép ngắt nhận hay không.

TE: cho phép truyền hay không.

RE: cho phép nhận hay không.

USART_BRR

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Thanh ghi cấu hình tốc độ baudrate UART.

USART_RDR

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	RDR[8:0]								
							r	r	r	r	r	r	r	r	r

Thanh ghi chứa data nhận được.

USART_TDR

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Thanh ghi chứa giá trị data muốn truyền đi.

USART_ISR

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TCBGT	Res.	Res.	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY
						r			r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

TXE : bit báo có data đã truyền hay không, =0 tức là data rỗng, có thể truyền, =1 data đã được truyền đi.

RXNE: bit báo data đã nhận hay chưa =1: đã nhận, =0 chưa nhận hoặc nhận chưa xong.

TC : cờ báo đã nhận data hoặc data vừa mới truyền xong.

Khi lập trình UART không thấy nhận được dữ liệu data, chúng ta cần debug những thanh ghi nào ?

Các bạn theo dõi ảnh bên dưới để hiểu cách ngắt USART được tạo ra nhé !

Table 86. USART interrupt requests

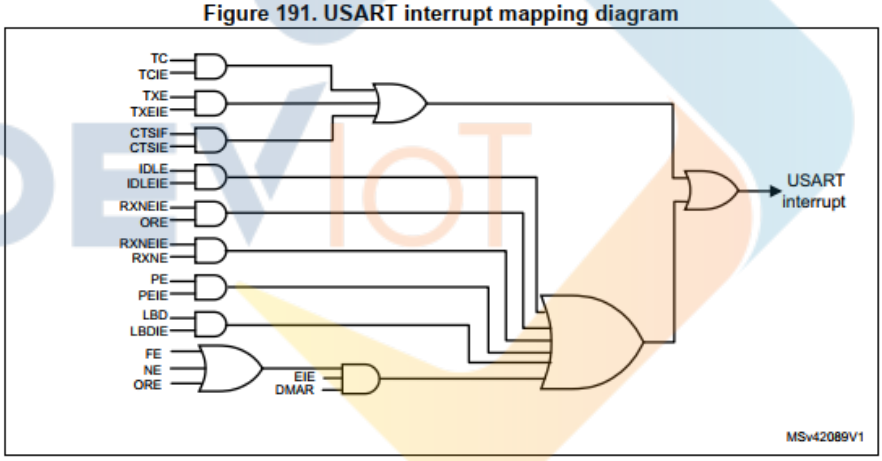
Interrupt event	Event flag	Enable control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NF or ORE or FE	EIE

The USART interrupt events are connected to the same interrupt vector (see *Figure 191*).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 191. USART interrupt mapping diagram

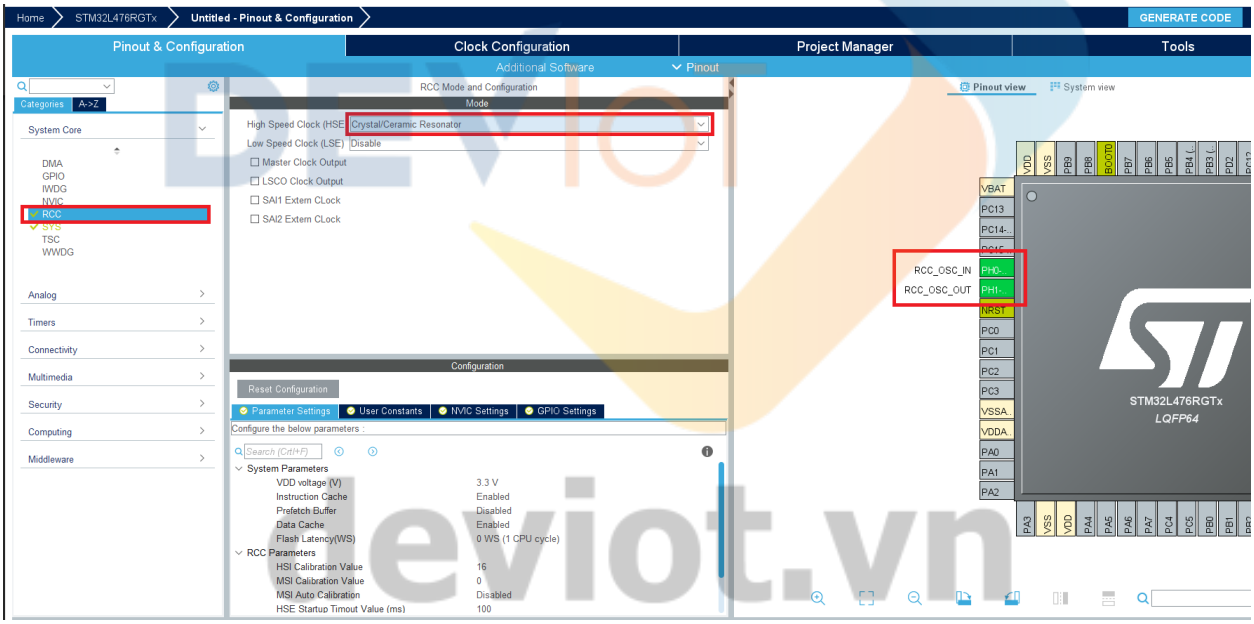


2. Lập trình

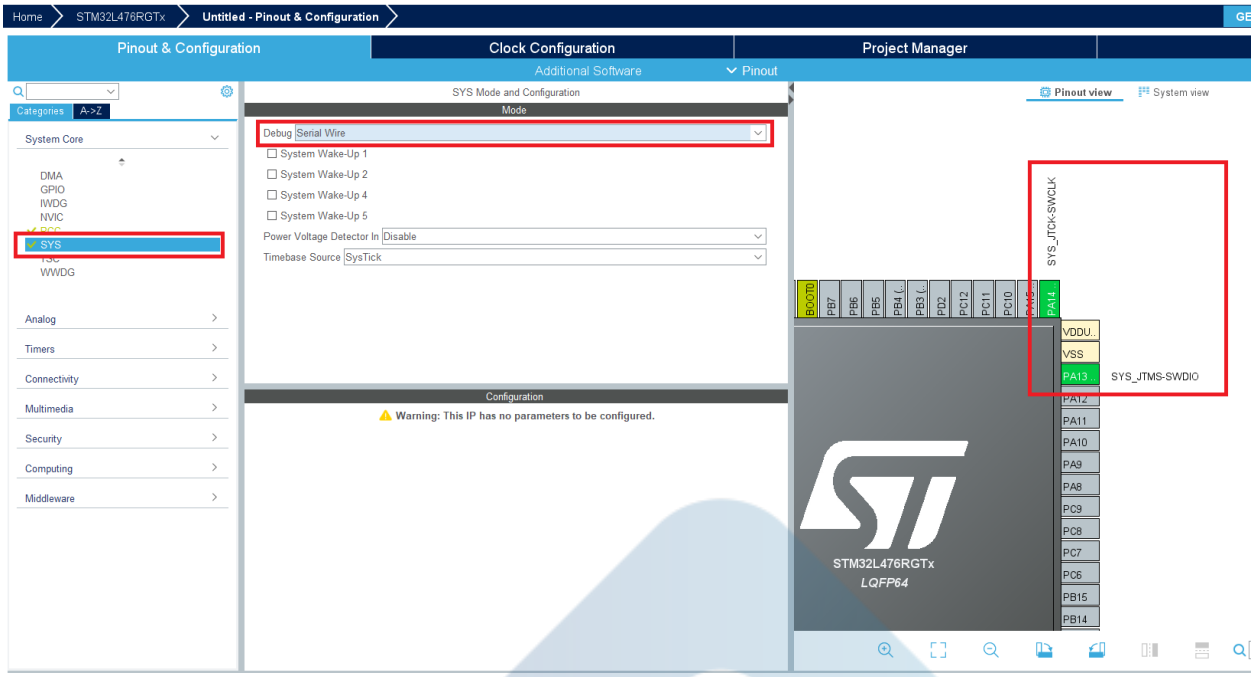
Mở phần mềm STM CubeMX, chọn dòng chip bạn sử dụng. Ở đây mình chọn chip STM32L476RG.

Đối với các dòng chip STM32 đời 4, tất cả mọi câu lệnh khi sử dụng thư viện HAL đều giống nhau. Chỉ khác nhau phần cấu hình Clock phụ thuộc riêng vào mỗi chip hỗ trợ.

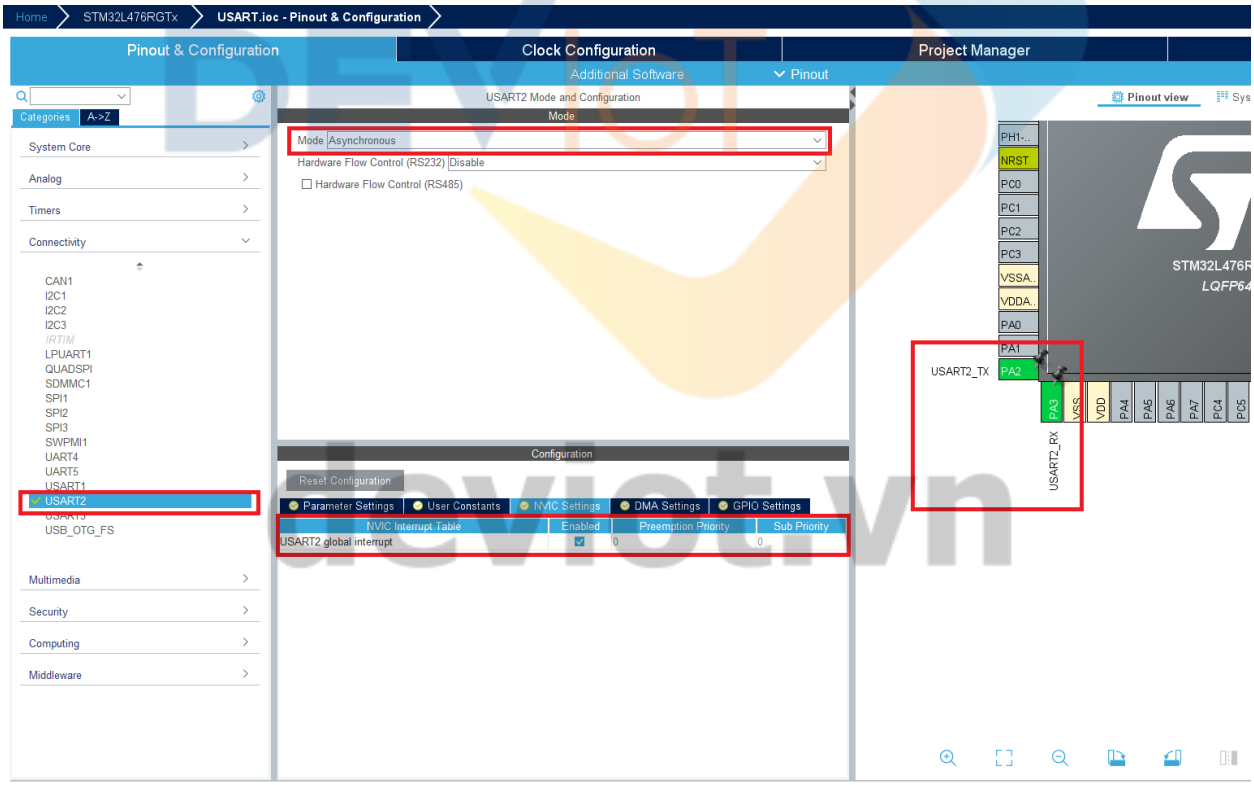
Cấu hình Chip sử dụng thạch anh ngoài.

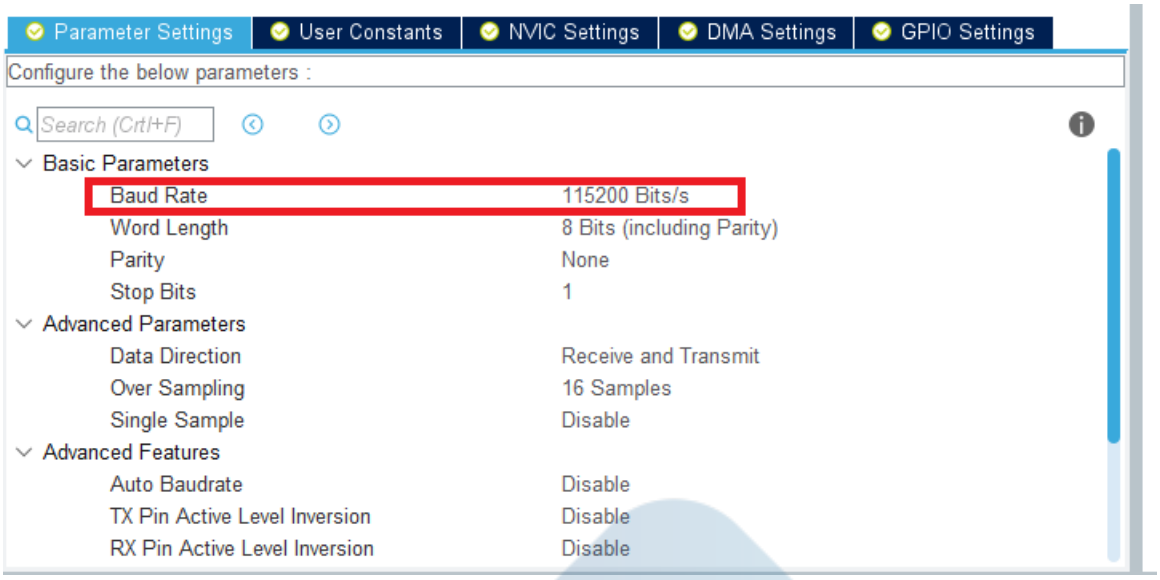


Cấu hình Chip Debug bằng mode SWD.

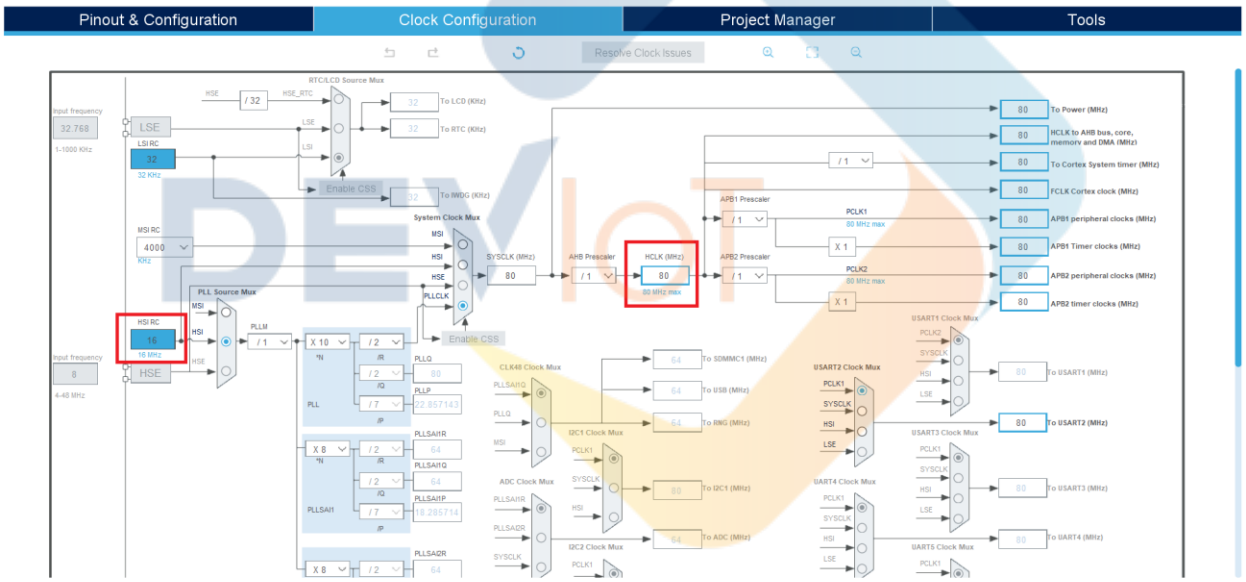


Ở đây mình cấu hình module UART2 để sử dụng. Cấu hình chế độ bất đồng bộ UART (khác với chế độ đồng bộ USART). Enable ngắt UART và cấu hình tốc độ Baudrate là 115200b/s.





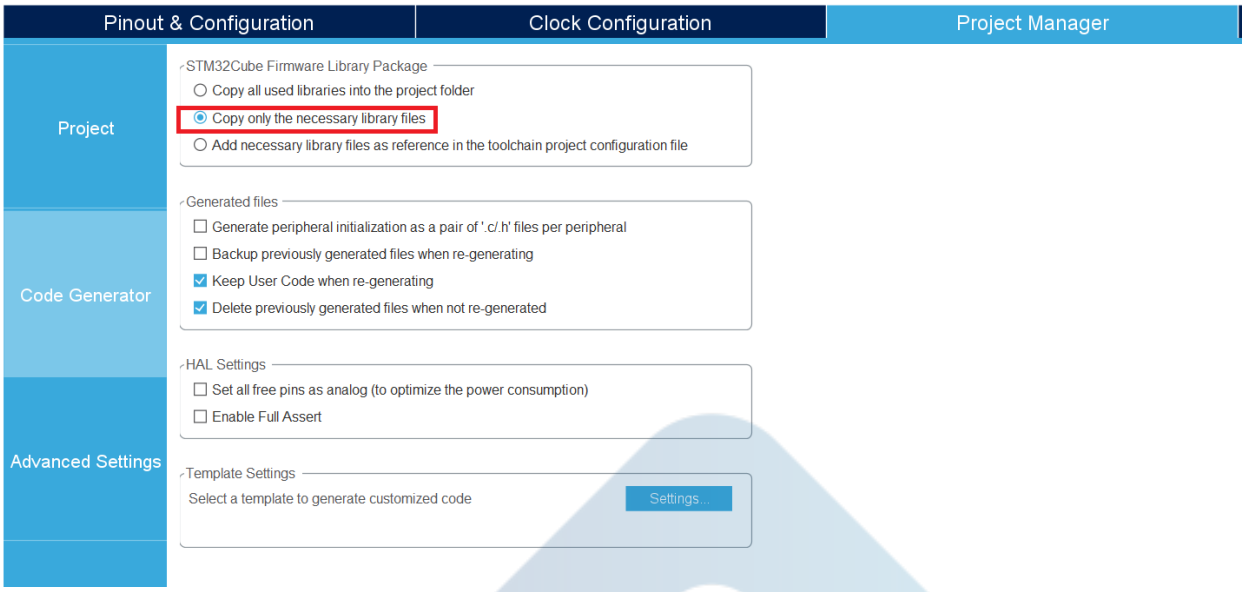
Tiếp theo, chuyển sang cấu hình tần số Clock cho Chip.



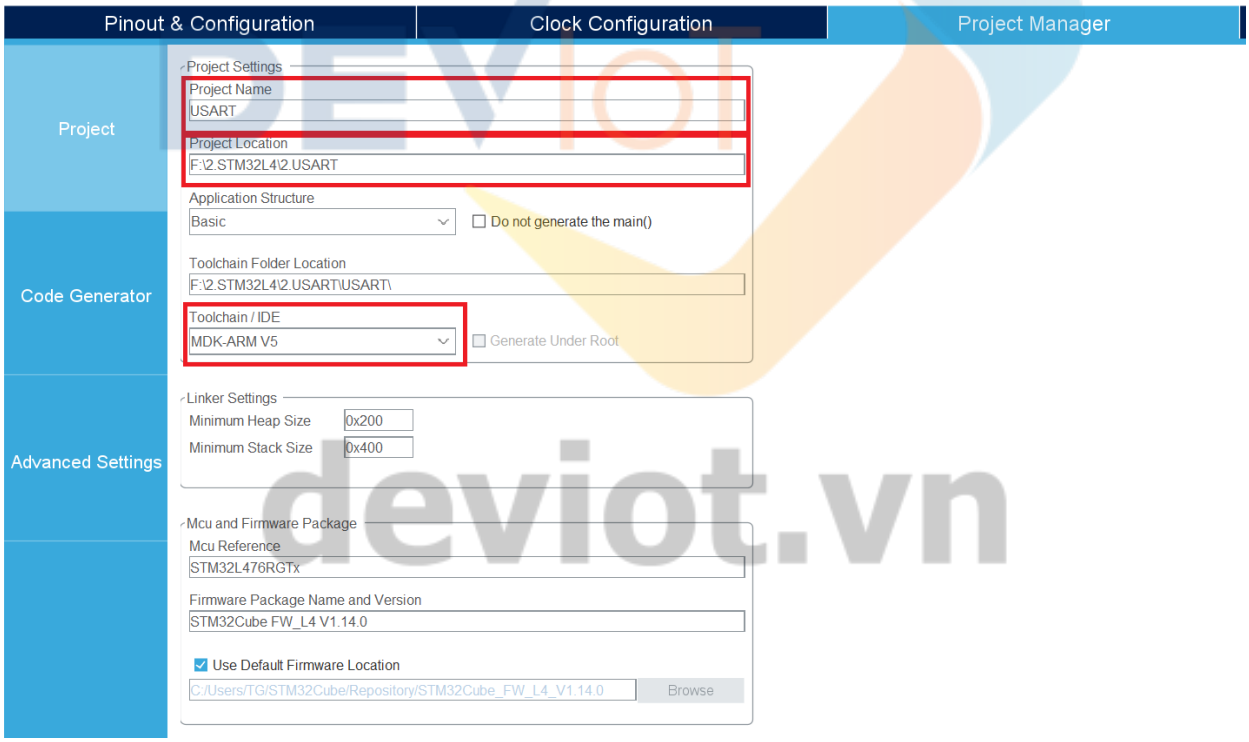
Ở đây mình chọn sử dụng nguồn Clock nội (HSI) được chip hỗ trợ sẵn, như vậy mình sẽ không cần hàn thêm thạch anh ngoại mà chương trình vẫn chạy được. Clock nội đi qua bộ nhân tần PLLCLK để đạt được tần số hoạt động tối đa mà chip hỗ trợ HCLK = 80MHz. Việc còn lại Cube MX sẽ tự cấu hình cho các bạn.

Tại sao bản thân chip đã có Clock nội nhưng chúng ta vẫn sử dụng thêm thạch anh ngoài ?

Cuối cùng chúng ta cấu hình vị trí lưu Code và sinh source Code.

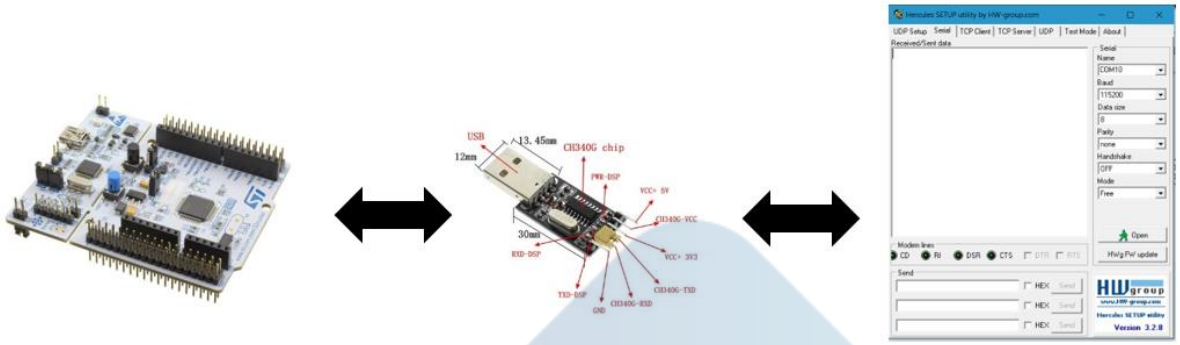


Bạn nhớ tick vào đây để CubeMX chỉ copy những file thư viện cần thiết cho Project đã cấu hình. Như vậy sẽ giảm thiểu được đáng kể dung lượng Project đấy.

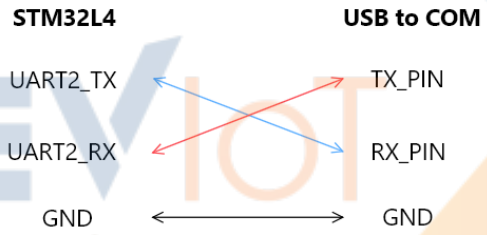


Bây giờ chúng ta cùng nhau sửa code nhé.

Ý tưởng của mình sẽ là khởi tạo module UART2, kết nối KIT STM32L4 với phần mềm Hercules trên mấy tính thông qua USBtoCOM. Sau đó sẽ gửi dữ liệu từ Hercules xuống KIT STM32L4, rồi từ KIT STM32L4 sẽ gửi ngược lại dữ liệu nhận được lên máy tính, hiển thị trên phần mềm Hercules.



Sơ đồ nối dây như sau



Hàm khởi tạo UART2 được sinh ra bởi Hercules

```
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}
```

Trong file main.c chúng ta init các Module cần thiết

```
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  HAL_UART_Transmit(&huart2,(uint8_t *)"HELLO WORLD\r\n",13,1000);
  HAL_UART_Receive_IT(&huart2,&c,1);
```

Khai báo hàm Callback xử lý dữ liệu nhận được trong file main.c

```
uint8_t ch;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if(huart->Instance == huart2.Instance)
  {
    HAL_UART_Receive_IT(&huart2,&ch,1);
    HAL_UART_Transmit(&huart2,&ch,1,1000);
  }
}
```

Ở đây chúng ta có 2 hàm khá đặc biệt. Mình sẽ giải thích kỹ một chút.

HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)

// *pData là con trỏ trỏ tới địa chỉ đầu tiên của mảng data cần truyền đi

// Size là số lượng byte cần truyền đi

// Timeout là khoảng thời gian tối đa chấp nhận cho quá trình truyền. Nếu quá trình vượt quá khoảng thời gian này thì hàm sẽ trả về mã lỗi.

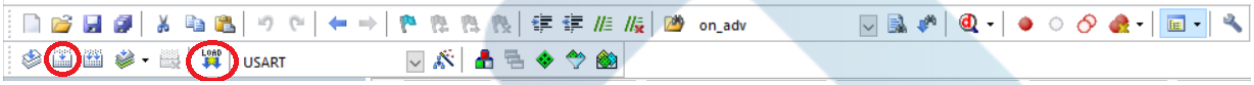
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)

// *pData là con trỏ trỏ tới mảng data nhận được

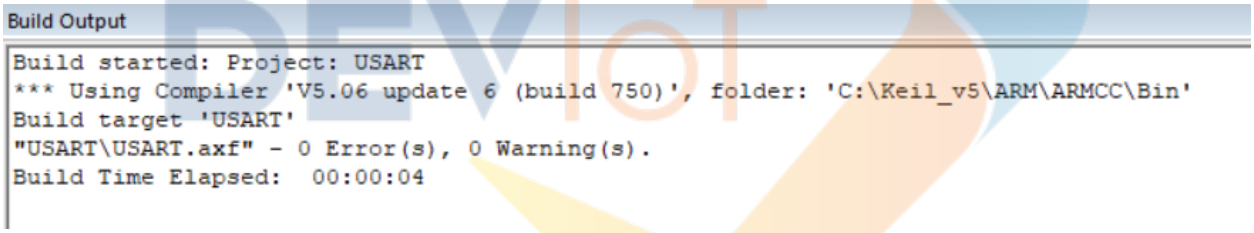
// Size là số lượng data nhận được

// Hàm này còn enable 2 bit PEIE và RXNEIE trong thanh ghi USART_CR1 để có thể tiếp tục nhận dữ liệu trong lần tiếp theo

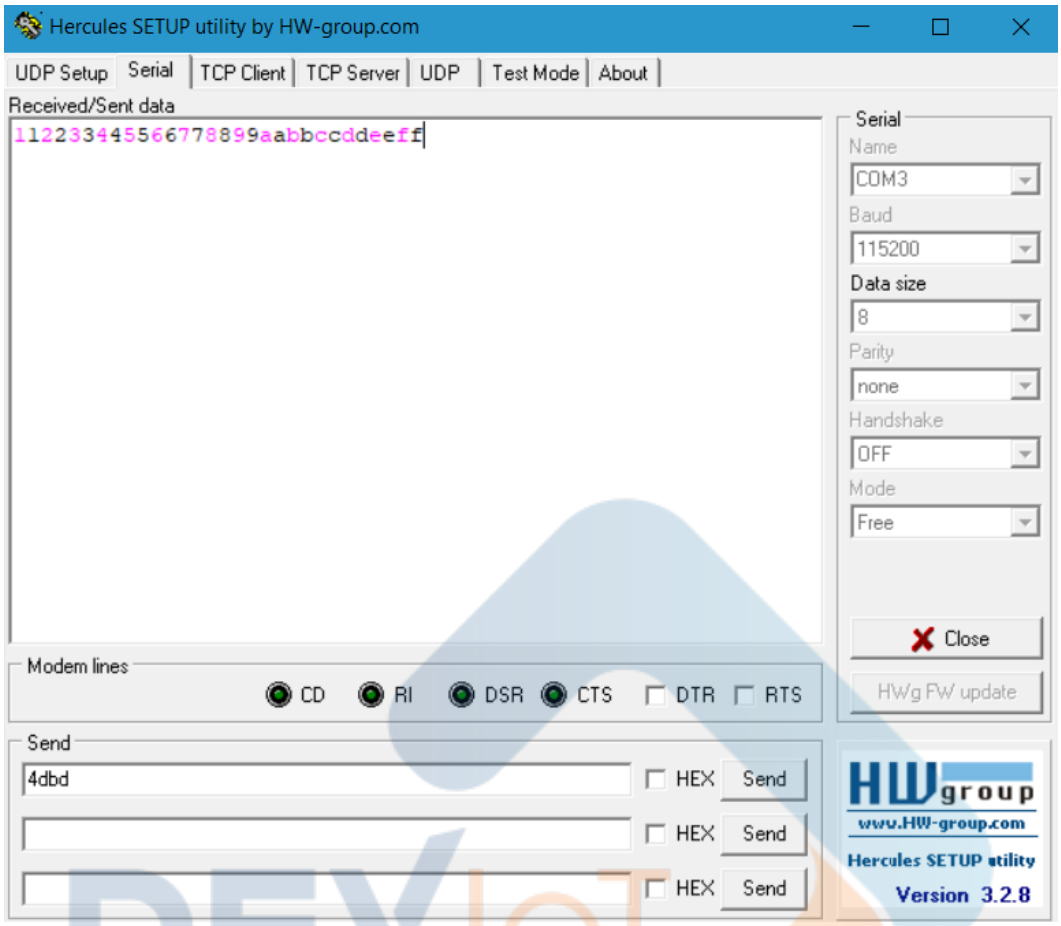
Tiến hành Build Code và Nạp chương trình xuống KIT



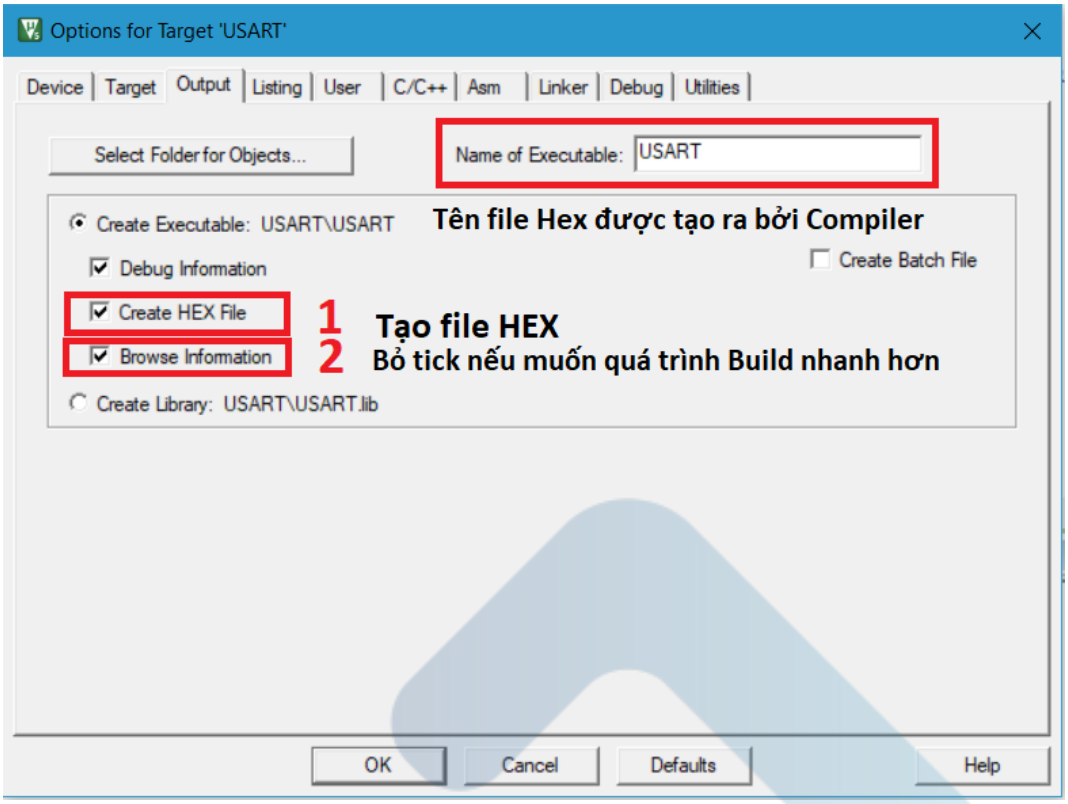
Kết quả build không có lỗi

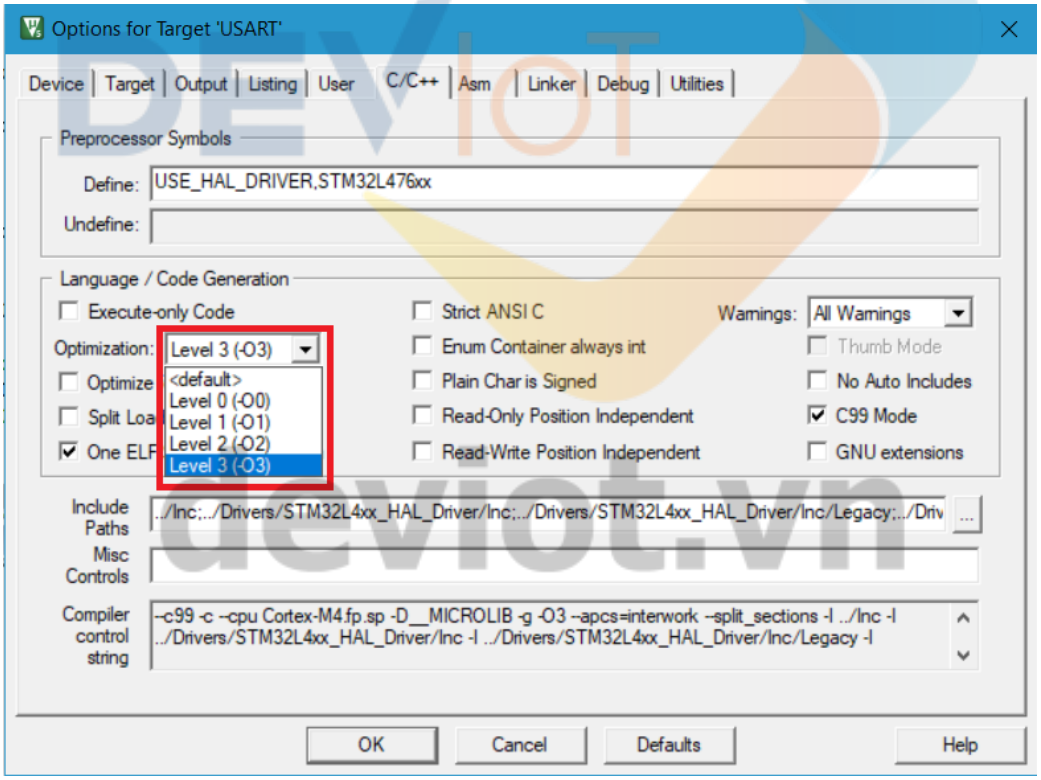


Kết quả khi chạy thực tế.



Một số cấu hình trong KeilC có thể có ích cho bạn





Optimization: Mức độ tối ưu code. Để tiện trong quá trình Debug, chọn Level 0 (mức tối ưu code thấp nhất). Khi xuất Firmware cho sản phẩm thực tế, nên chọn Level 3 (mức tối ưu code cao nhất) sẽ cho ra file Firmware có dung lượng thấp nhất.

DEVIOT - CÙNG NHAU HỌC LẬP TRÌNH IOT

📌 Website: deviot.vn

📌 FanPage: Deviot - Thời sự kỹ thuật & IoT

📌 Group: Deviot - Cùng nhau học lập trình IOT

📌 Hotline: 0969.666.522

📌 Address: Số 101C, Xã Đàn 2

📌 Đào tạo thật, học thật, làm thật