Universal Asynchronous Serial Communications

USART/UART

محتويات الجلسة:

1- مقدمة
2- مفهوم UART و USART
3- أنماط العمل المختلفة للـ UART في متحكمات STM32
4- دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في نمط الـ Polling
5- التطبيق العملي الأول: مراقبة عمل أي كود (Debugging) من خلال نافذة الاتصال التسلسلي UART وباستخدام نمط الـ Polling
6- إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt
7- تطبيق عملي لاستخدام المنفذ التسلسليUSART من خلال نمط الـ interrupt

الأدوات اللازمة للجلسة:

- لوحة Nucleo-64bit
- كبل Type-A to Mini-B
- ليدات
- مفاتيح لحظية

1- مقدمة :

في هذه الأيام يوجد العديد من بروتوكولات الاتصال التسلسلية Serial communication Protocols ، حيث يركز أغلبها على سرعة نقل البيانات مثل USB 3.0 /USB2.0 وغيرها...، إحدى طرق الاتصال التي تم إنشاؤها قديماً ومازالت تستخدم حتى الان للربط بين المتحكمات المصغرة هي Universal Synchronous/Asynchronous Receiver/Transmitter Interface (USART).

Synchronous: هو الإرسال والاستقبال المتزامن المبني على وجود clock بين المرسل والمستقبل .

Asynchronous: لا يعتمد على clock وإنما يتم الاكتفاء بإرسال البيانات على خط الإرسال ويتم استقبالها على خط الاستقبال.

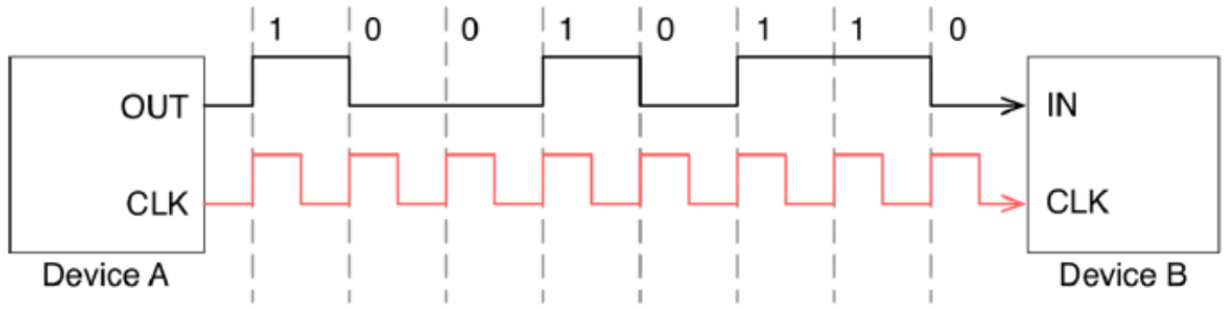
كل متحكم STM32 يحتوي على الأقل على وحدة طرفية UART واحدة، وأغلب متحكمات STM32 توفر على الأقل اثنتين من UART/USART ، وأخرى توفر لحد 8 وحدات طرفية UART/USART، حيث توفر لوحة Nucleo-64 التي ستخدمها في منهاجنا 4 وحدات طرفية UART/USART، وتتصل الوحدة UART2 مع دارة المبرمجة ST-LINK المدمجة على اللوحة.

2- مفهوم UART و USART:

عندما تريد نقل البيانات ما بين جهازين أو أكثر، يوجد طريقتين لإرسال البيانات الأولى وهي:

- نقل البيانات على التوازي Parallel : في هذه الطريقة توجد مجموعة من خطوط البيانات على حسب طول البيانات التي سيتم إرسالها (مثلاً 8 خطوط بيانات في حال إرسال بيانات بطول 8بت).

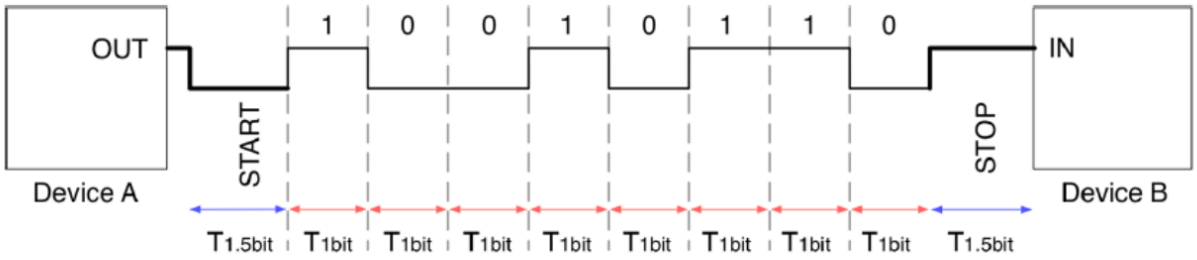
- نقل البيانات على التسلسل Serial : وفي هذه الطريقة يتم إرسال البيانات على التوالي بت تلو الآخر باستخدام خط بيانات واحد حيث يكون أحد الجهازين المتصلين مرسل والآخر مستقبل ففي وضع Synchronous يتم مشاركة clock ما بين المرسل والمستقبل والتي يتم إنشاؤها دائماً باستخدام الجهاز الذي يدير الاتصال.



الشكل(1): الاتصال التسلسلي المتزامن (Synchronous) ما بين جهازين

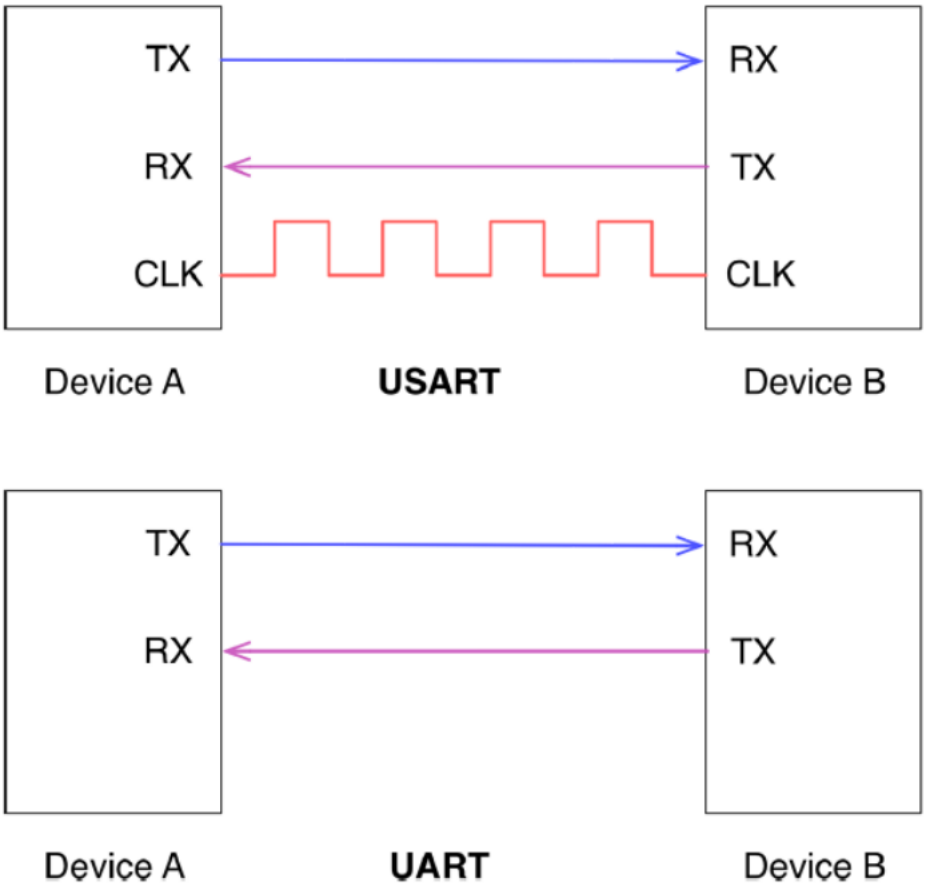
يبين الشكل(1) المخطط الزمني لعملية الإرسال التسلسلي المتزامن (Synchronous) لبايت واحد (0b01101001) من الجهاز Device A إلى الجهاز Device B حيث تم ا ستخدام Clock لـ ضبط توقيت إر سال البيانات، حيث يتم إرسال بت واحد مع كل جبهة صاعدة للـ Clock ، حيث تتعلق سرعة نقل البيانات بتردد الـ Clock، فكلما زاد تردد الـClock كلما زادت سرعة نقل البيانات.

أما في حالة الإرسال الغير متزامن (Asynchronous) يتم الاستغناء عن الـ clock حيث يتم استخدام بت عند بداية الإرسال Start Bit و بت عن انتهاء الإرسال Stop Bit.



الشكل(2): المخطط الزمني للاتصال التسلسلي الغير متزامن (Asynchronous) ما بين جهازين

يو ضح الـ شكل الـ سابق المخطط الزمني للإر سال الغير متزامن Asynchronous، حيث تمثل الحالة الخاملة Idle state بإشارة HIGH وهي حالة عدم الإر سال، بداية الإر سال تتم من خلال Start Bit وتمثل بإشارة LOW، هذه الإ شارة يتم اكتـ شافها من قبل المـ ستقبل وتـ ستغرق وقت 1.5T حيث أن T هو الدور وهو عبارة عن مقلوب التردد أو مايسمى Baud Rate أي معدل نقل البيانات بين المرسل والمستقبل، بعد ذلك يتم إرسال الـ 8bit والتي تمثل البيانات المراد إرسالها حيث يتم إرسال البت الأقل أهمية أولاً LSB ، وأحياناً يتم استخدام Parity Bit للتأكد من خلو البيانات من الأخطاء ويتم إنهاء الإرسال بـ Stop Bit.



الشكل(3): شكل الإشارة في حالة UART و USART

كما ذكرنا سابقاً فإن لوحة Nucleo-64 التي نستخدمها في منهاجنا تحتوي على دارة ST-LINK مدمجة معها وهي توفر إمكانية الاتصال بلوحة Nucleo في وضع الـ Serial - وبالتالي يمكن استخدام الـ ST-LINK - لقراءة خرج الـ UART (في حالتنا تم ا ستخدام المنفذ التـ سلـ سلي UART2)، لكن في حالة ا ستخدام لوحات تطويرية أخرى كلوحة Blue Pill لا تتوفر هذه الميزة لعدم وجود دارة ST-LINK مدمجة على اللوحة وبالتالي نحتاج لاستخدام دارة ST-LINK خارجية وهي لا تدعم خاصية الـ Serial، لذا إذا أردنا الاتصال باللوحة من خلال الـ Serial نحتاج إلى دارة FT232RL.

Pin number								Pin name (function upon reset)	Pin type	I/O structure	Note	Alternate functions	Additional functions
WLCSP25	UFQFPN28 - GP	UFQFPN28 - N	LQFP32 / UFQFPN32 - GP	LQFP32 / UFQFPN32 - N	LQFP48 / UFQFPN48	UFBGA64	LQFP64						
D4	7	7	8	8	12	H3	18	PA1	I/O	FT_a	-	SPI1_SCK/I2S1_CK, USART2_RTS_DE_CK, TIM2_CH2, USART4_RX, TIM15_CH1N, I2C1_SMBA, EVENTOUT	COMP1_INP, ADC_IN1
E4	8	8	9	9	13	G3	19	PA2	I/O	FT_a	-	SPI1_MOSI/I2S1_SD, USART2_TX, TIM2_CH3, UCPD1_FRSTX, TIM15_CH1, LPUART1_TX, COMP2_OUT	COMP2_INM, ADC_IN2, WKUP4,LSCO
C3	9	9	10	10	14	F3	20	PA3	I/O	FT_a	-	SPI2_MISO, USART2_RX, TIM2_CH4, UCPD2_FRSTX, TIM15_CH2, LPUART1_RX, EVENTOUT	COMP2_INP, ADC_IN3
-	-	-	-	-	15	H4	21	PA4	I/O	TT_a	-	SPI1_NSS/I2S1_WS, SPI2_MOSI, TIM14_CH1, LPTIM2_OUT, UCPD2_FRSTX, EVENTOUT	ADC_IN4, DAC_OUT1, RTC_OUT2

الشكل(4):أقطاب المتحكم المتصلة بالمنفذ التسلسلي USART2

3- أنماط العمل المختلفة لـ UART في متحكمات STM32:

توجد 3 أنماط عمل مختلفة لـ UART وهي:

1. نمط Polling Mode:

يسمى أيضاً Blocking Mode ، في هذا النمط يتم فحص عملية إرسال واستقبال البيانات بشكل مستمر ، حيث ينتظر المعالج لحين انتهاء عملية الإرسال مما يؤدي إلى تأخير معالجة باقي التعليمات وتنفيذ المهام ، وهو نمط العمل الأبسط من ناحية الكود ومن ناحية الـ Hardware ويستخدم عندما تكون كمية البيانات المتبادلة ليست كبيرة نسبياً ولا تمثل أهمية عالية من ناحية المعالجة.

2. نمط المقاطعة Interrupt Mode:

ويسمى أيضاً non-Blocking Mode ، في هذا النمط لا يتم الانتظار وتفقد البيانات من حين لآخر للتأكد من عملية الارسال والاستقبال، حيث عند الانتهاء من إرسال البيانات يتم تفعيل مقاطعة تفيد بانتهاء عملية الإرسال ، وهذا النمط من العمل أفضل من ناحية المعالجة ملائم عندما يكون معدل نقل البيانات صغير نسبياً (أقل من 38400Bps).

3. نمط DMA:

وهو النمط الأفضل من ناحية إنتاجية نقل البيانات ومن ناحية سرعة نقل البيانات وعندما نريد تحرير المتحكم من الحمل الإضافي الذي ينتج عن من إحضار البيانات من RAM ومعالجتها، فالـ DMA يقوم بالوصول إلى الذاكرة RAM بدون احتياج أي جهد من المعالج لعمل ذلك، وبدون نمط الـ DMA لا يمكن التعامل مع السرعات العالية في الـ UART.

سنشرح بالتفصيل نمطي الـ polling والـ Interrupt من خلال التطبيق العملي.

4- دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling:

سنستخدم دالتين رئيسيتين للتعامل مع المنفذ التسلسلي إحداهما للإرسال والأخرى للاستقبال:

1. دالة الإرسال:

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

حيث:

huart: وهو مؤشر يشير إلى Struct_UART_HandleTypeDef أي المنفذ التسلسلي المستخدم للاتصال مثلاً قد يكون &huart1 أو &huart2 أو &huart3 أو &huart4 باعتبار لدينا أربع منافذ تسلسلية في لوحة Nucleo.

pData: وهو مؤشر أيضاً يشير إلى البيانات التي سيتم إرسالها عبر UART وكما نرى نوعه uint8_t أي يقبل إرسال بيانات من نوع Unsigned int وبطول 8bit، مثال: قد تكون pData مصفوفة وليكن اسمها Data وتكون معرفة بالشكل التالي:

Uint8_t Data[]= {0,1,2,3,4,5,6,7,8,9};

حيث:

Size: وهو متغير يعبر عن حجم البيانات التي سيتم إرسالها أي pData وهي في المثال السابق 10.

Timeout: أقصى زمن يتم انتظاره بالميللي ثانية حتى يتم اكتمال عملية الإرسال، فإذا تم انتهاء هذا الزمن ولم تتم عملية الإرسال سيتم قطع عملية الإرسال وتقوم الدالة بإرجاع HAL_TIMOUT ماعدا ذلك يتم ارجاع HAL_OK، ويمكن استخدام هذه الدالة مكان Timeout و هي HAL_MAX_DELAY ووظيفتها انتظار أقصى زمن ممكن لعملية الإرسال.

مثال: إذا أردنا إرسال المصفوفة التالية عبر المنفذ التسلسلي الأول UART1:

```
/* USER CODE BEGIN 0 */
uint8_t data[]={0,1,2,3,4,5,6,7,8,9};
/* USER CODE END 0 */
```

نستخدم الدالة التالية:

```
/* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   HAL_UART_Transmit(&huart1,data,10,1000);
 }
 /* USER CODE END 3 */
```

2. دالة الاستقبال:

إذا أردنا استقبال بيانات على UART باستخدام وضع Polling ومكتبات HAL نقوم باستدعاء الدالة التالية:

```
HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t
Timeout);
```

حيث:

Size: وهو متغير يعبر عن حجم البيانات التي سيتم استقبالها أي pData وهي في المثال التالي 10.

Timeout: أقصى زمن يتم انتظاره بالميللي ثانية حتى يتم اكتمال عملية الاستقبال، فإذا تم انتهاء هذا الزمن ولم تتم عملية الاستقبال سيتم قطع عملية الاستقبال وتقوم الدالة بإرجاع HAL_TIMOUT ما عدا ذلك يتم ارجاع HAL_OK، ويمكن استخدام هذا المتحول كما لو كان Timeout وهي HAL_MAX_DELAY ووظيفتها انتظار أقصى زمن ممكن لعملية الاستقبال.

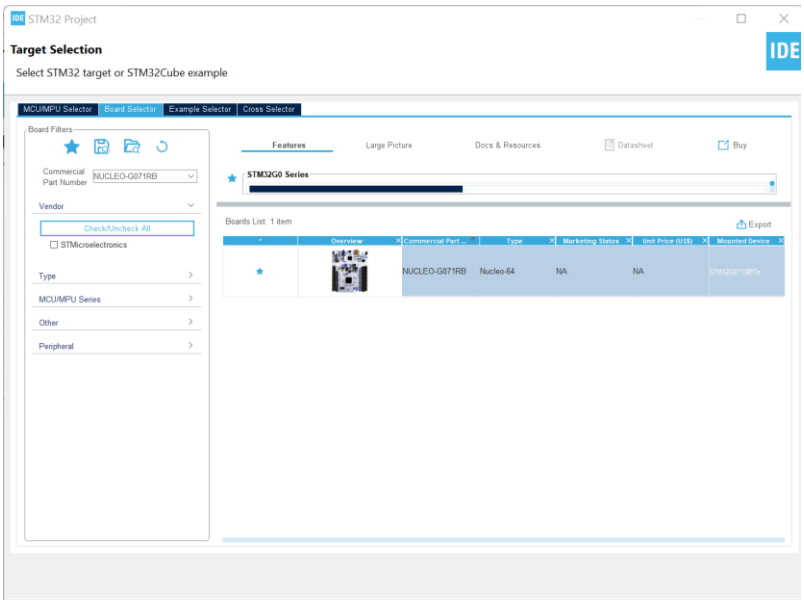
مثال: إذا أردنا استقبال مصفوفة أعداد صحيحة من المنفذ التسلسلي UART1:

```
/* USER CODE BEGIN 0 */
uint8_t data[10];
/* USER CODE END 0 */
```

```
/* USER CODE BEGIN 3 */
 /* Infinite loop */
 while (1)
 {
   HAL_UART_Receive(&huart1,data,10,1000);
 }
 /* USER CODE END 3 */
```

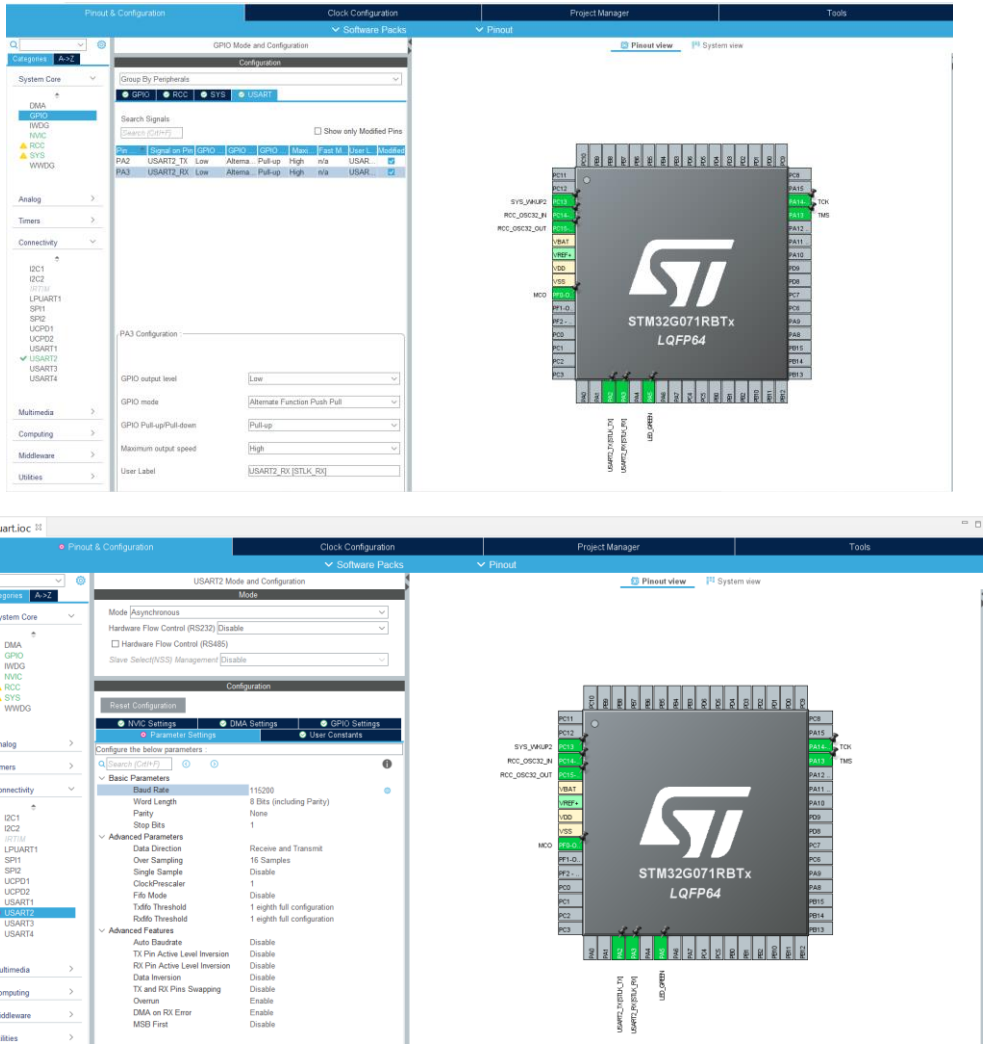
5- التطبيق العملي الأول: مراقبة عمل أي كود (Debugging) من خلال نافذة الاتصال التسلسلي UART وباستخدام نمط الـ Polling:

الخطوة الأولى: قم بفتح برنامج STM32CubeIDE ومن ثم قم بإنشاء مشروع جديد من نافذة File ثم New ثم STM32Project ثم قم باختيار المتحكم المصغر أو من خلال اختيار اسم اللوحة المستخدمة وهي في حالتنا Nucleo-G071RB كما في الشكل التالي:



الشكل(5) :بدء مشروع جديد في بيئة STM32CubeIDE

الخطوة الثانية: قم بضبط إعدادات المنفذ UART2 المتصل داخلياً مع دارة الـ ST-LINK المدمجة مع اللوحة، كما في الشكل التالي:



الشكل(6):ضبط إعدادات المنفذ التسلسلي

من ضمن الإعدادات الموجودة Basic Rate ما يلي:

Baudrate: يمثل معدل نقل البيانات بواحدة Bit/Sec بين المرسل والمستقبل، ولها قيم قياسية يتم الاختيار منها، حيث تعتمد هذه القيم على وحدة الـ Clock Peripheral الخاصة بالـ USART وهي عبارة عن ساعة المتحكم المصغرة على رقم ثابت، لكن ليس كل الـ BaudRates المتاحة يمكن استخدامها فقد ينتج عن المعدلات العالية أخطاء، حيث يوضح الشكل التالي الـ BaudRates القياسية والأخطاء التي قد تنجم عن كل BaudRate، حيث تختلف قيم الـ BaudRates المتاحة من متحكم لآخر بناءً على Clock Peripheral التي يوفرها للـ USART فقد يدعم متحكم BaudRates أكثر أو أقل.

	Baud rate	Oversampling by 16		Oversampling by 8	
S.No	Desired (Bps)	Actual	%Error	Actual	%Error
2	2400	2400	0	2400	0
3	9600	9600	0	9600	0
4	19200	19200	0	19200	0
5	38400	38400	0	38400	0
6	57600	57620	0.03	57590	0.02
7	115200	115110	0.08	115250	0.04
8	230400	230760	0.16	230210	0.8
9	460800	461540	0.16	461540	0.16
10	921600	923070	0.16	923070	0.16
11	2000000	2000000	0	2000000	0
12	3000000	3000000	0	3000000	0
13	4000000	N.A.	N.A.	4000000	0
14	5000000	N.A.	N.A.	5052630	1.05
15	6000000	N.A.	N.A.	6000000	0

الشكل(7):حسابات الخطأ Error Calculation للـBaudRates

WordLength: وتعني عدد البتات التي يتم إر سالها أو ا ستقبالها في Frame (في المرة الواحدة)، وتوفر 3 قيم يمكن الاختيار بينها 7bit,8bit,9bit حيث لا يتضمن هذا الرقم البتات الخاصة بـ Start و الـ Stop وغيرها.

StopBits: يحدد عدد البتات الخاصة بالـ Stop - التي سيتم إرسالها، ويمكن الاختيار بين 1 و 2 أي بت واحد أو 2bit في نهاية الإشارة.

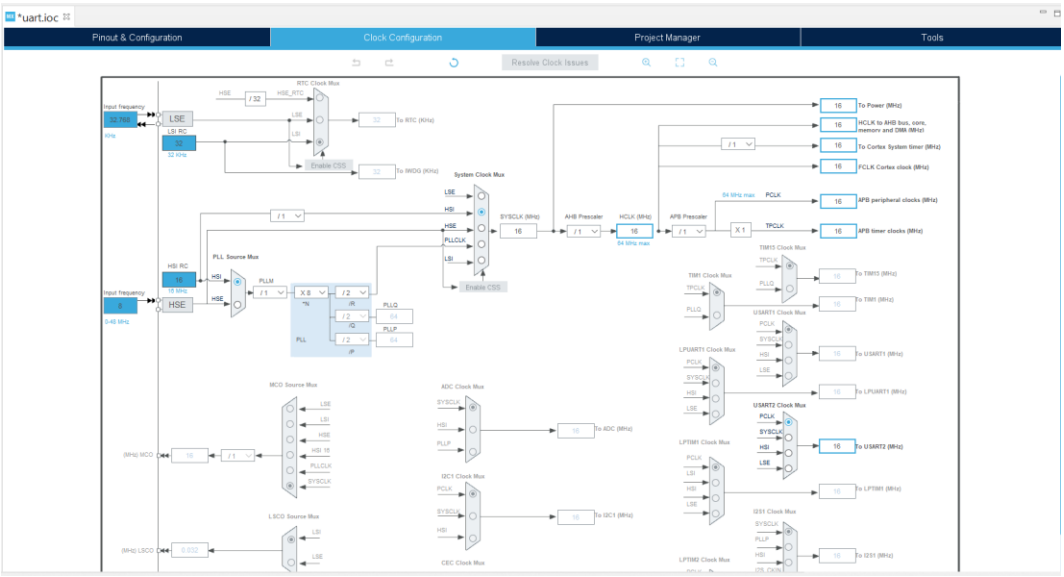
Parity: هو عبارة عن اختبار يـ ستخدم لاكتـ شاف الأخطاء أثناء عملية الإر سال والا ستقبال للبيانات من خلال الـ USART ، وهو عبارة عن بت يكون مكانه عند البت الأكثر أهمية MSB بحيث لو تم ا ستخدام Word Length بـ 8-bit يكون مكانه في البت الثامن، أما لو تم استخدام 9-bit يكون مكانه هو في البت التاسع، ولها نمطين:

- فردي Odd: وتكون قيمة بت الـ Parity - مـ ساوٍ للواحد المنطقي عندما يكون عدد الواحدات الموجودة في الكلمة المراد إر سالها زوجي، و صفر منطقي في حال كان عدد الواحدات الموجودة في الكلمة المراد إر سالها فردي.
- زوجي Even: وتكون قيمة بت الـ Parity - مـساوٍ للواحد المنطقي عندما يكون عدد الواحدات الموجودة في الكلمة المراد إر سالها فردي، و صفر منطقي في حال كان عدد الواحدات الموجودة في الكلمة المراد إر سالها زوجي.

على سبيل المثال: عندما تريد إر سال أي بيانات يتم تحويلها للـ Binary - فمثلاً إذا كنا نريد إر سال الكلمة التالية 0b01101110 فمن خلال الـ Parity - يتم حساب عدد الواحدات الموجودة ضمن هذه الكلمة المراد إرسالها وهي في هذه الحالة 5، ففي حال كنت تـ ستخدم نمط الفردي ستكون قيمة الـ Parity - صفر، أما في حال كنت تـ ستخدم نمط الزوجي ستكون قيمة الـ Parity واحد.

ويتم إرسال قيمة البت الخاص بالـ Parity من المرسل إلى المستقبل، فإن لم يحصل تطابق بين قيمته عند المرسل مع قيمته عند المستقبل فهذا يعني وجود خطأ ما في الإرسال حيث يتم طلب إعادة الإرسال.

الخطوة الثالثة: ضبط تردد الساعة للمتحكم (تذكر أن لوحة Nucleo التي نستخدمها لا تحتوي على كريستالة خارجية ولكن بإمكانك إضافة واحدة) سنختار مصدر الساعة الداخلي HSI:



الشكل(8): ضبط تردد الساعة للمتحكم

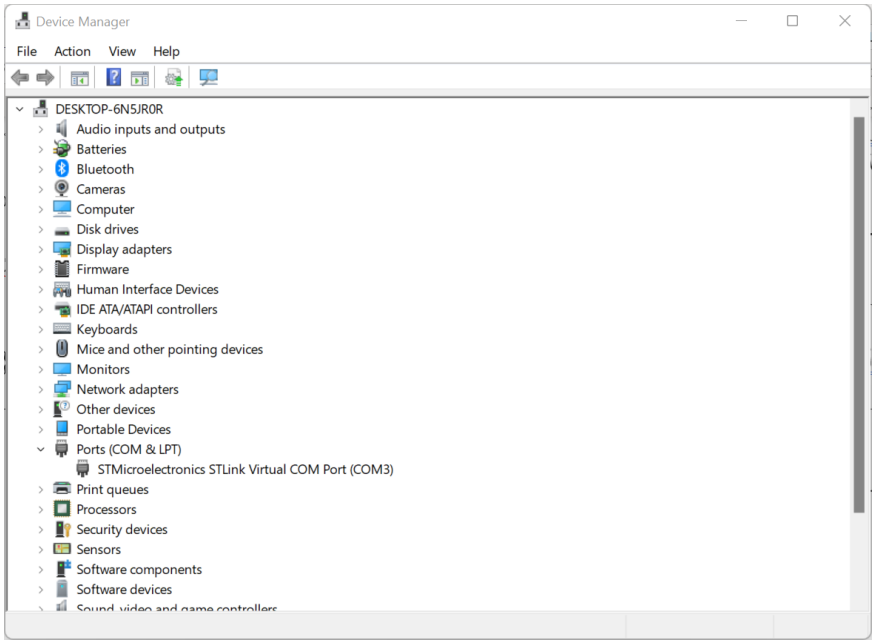
الخطوة الرابعة: توليد الكود اعتماداً على الإعدادات التي تم اختيارها من Project...Generate code

الخطوة الخامسة: كتابة الكود المناسب، حيث سنقوم بطباعة عبارة ("Hello Dudes! Tracing X =") و سنقوم بتعريف عداد X تزداد قيمته في كل مرة يعيد فيها المعالج تنفيذ الحلقة اللانهائية While(1)، ويكون الكود بالـ شكل التالي:

#include "main.h"	تضمين المكتبة الرئيسية main.h
UART_HandleTypeDef huart2;	تعريف منفذ الاتصال المستخدم وهو في حالتنا UART2
void SystemClock_Config(void);	تعريف الدالة المستخدمة لضبط إعدادات ساعة المتحكم
static void MX_GPIO_Init(void);	تعريف الدالة المستخدمة لضبط إعدادات أقطاب الدخل والخرج للمتحكم
static void MX_USART2_UART_Init(void);	تعريف الدالة المستخدمة لضبط إعدادات المنفذ التسلسلي للمتحكم
int main(void) {	البرنامج الرئيسي
uint8_t MSG[35] = {'\0'};	تصريح عن مصفوفة محارف بعدد 35 محرف
uint8_t X = 0;	تصريح عن متحول من نوع uint_8t وإسناد قيمة صفرية له كقيمة ابتدائية
HAL_Init();	استدعاء الدالة المسؤولة عن تهيئة مكتبة HAL
SystemClock_Config();	استدعاء الدالة المسؤولة عن ضبط إعدادات ساعة المتحكم

MX_GPIO_Init();
 	
MX_USART2_UART_Init();
 	
while (1)
{
sprintf(MSG, "Hello Dudes! Tracing X = %d\r\n", X);
HAL_UART_Transmit(&huart2, MSG, sizeof(MSG), 100);
HAL_Delay(500);
X++;
}}
	استدعاء الدالة المسؤولة عن ضبط أقطاب الدخل والخرج للمتحكم
	استدعاء الدالة المستخدمة لضبط إعدادات المنفذ التسلسلي للمتحكم
	
	حلقة While(1) اللانهائية
	تشكيل الجملة المراد إرسالها على المنفذ التسلسلي
	
	إرسال السلسلة MSG إلى المنفذ التسلسلي UART2
	إضافة تأخير زمني بين عمليات الإرسال المتكررة بمقدار 500 ميللي ثانية
	زيادة المتحول X في كل مرة

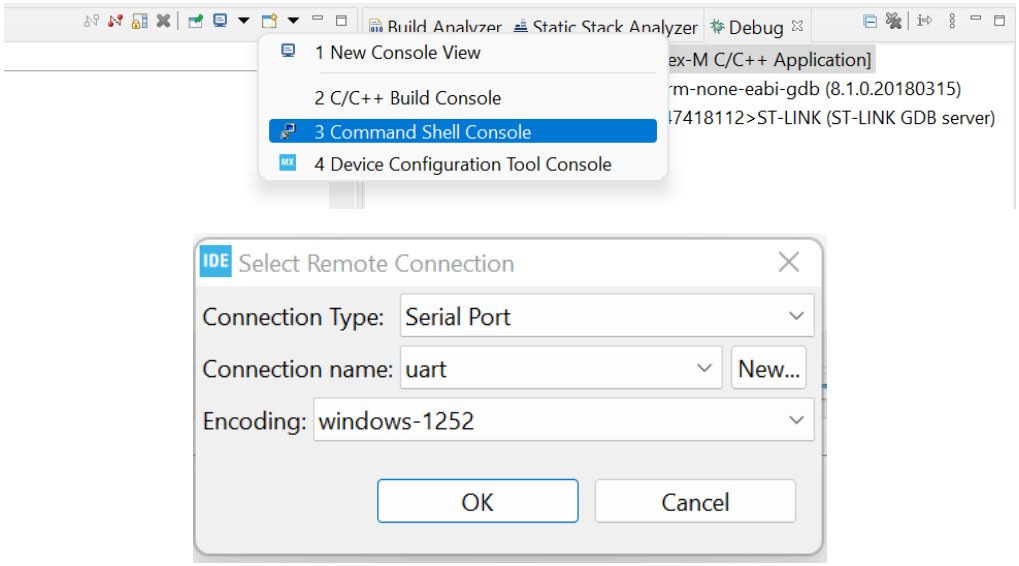
الخطوة الـ ساد سة: ترجمة الكود وإر ساله إلى لوحة Nucleo ، ثم قم بفتح Device Manager لتعرف رقم المنفذ التسلسلي المستخدم من الحاسب للاتصال مع اللوحة:



الشكل(9):معرفة رقم المنفذ التسلسلي من الحاسب المتصل بـ ST-LINK

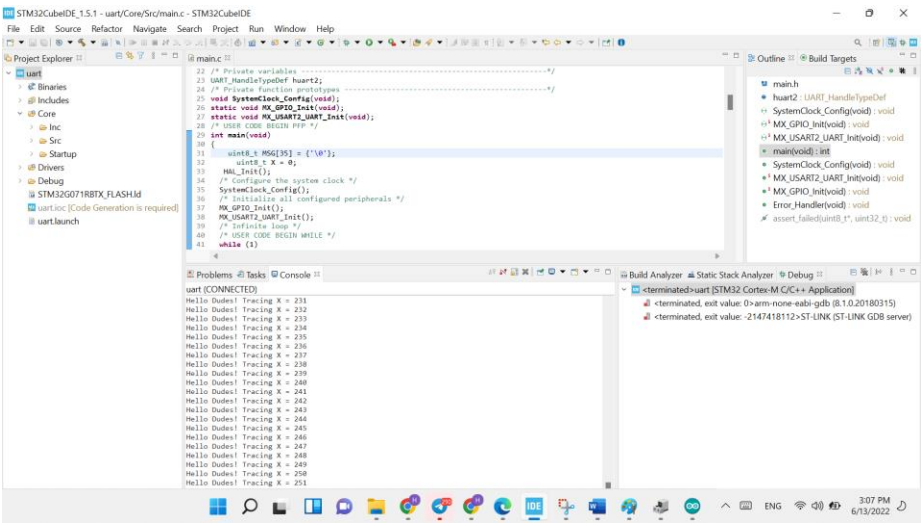
الخطوة السابعة: من القائمة Windows اختر show view ثم console فتجد أن الـ console ظهرت في أسفل الشاشة ، ثم من console نختار command shell console ثم نختر نوع منفذ الاتصال وهو Serial port

ونعطيه اسم مثلاً uart كما في الشكلين التاليين:



الشكل(11):إعداد بيئة STM32CubeIDE للاتصال التسلسلي مع اللوحة

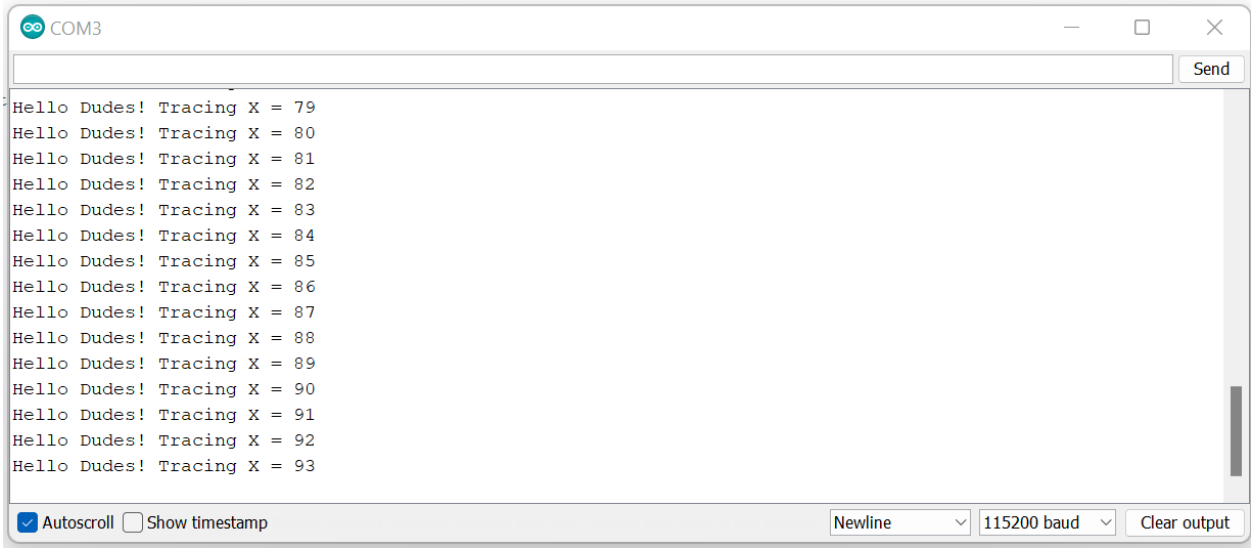
فنبدأ عملية طباعة الجملة المرادة كما في الشكل التالي:



الشكل(12):الطباعة على النافذة التسلسلية ضمن بيئة STM32CubeIDE

ملاحظة:

بإمكانك ا ستخدام أي أداة للتخاطب مع المنفذ التـ سلـ سلي على سبيل المثال Mobaxterm أو حتى من خلال بيئة Arduino بعد ضبط الإعدادات الأ سا سية مثل رقم المنفذ التـ سلـ سلي للحا سب المتـ صل باللوحة وأيـ ضاً معدل نقل البيانات BaudRate كما في الشكل التالي:



الشكل(13):الطباعة على النافذة التسلسلية ضمن بيئة Arduino

6- إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt:

توفر جميع متحكمات STM32 مقاطعات لـ UART كما في الجدول التالي:

Interrupt Event	Event Flag	Enable Control Bit
Transmit Data Register Empty	TXE	TXEIE
Clear To Send (CTS) flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	RXNEIE
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multi buffer communication	NF or ORE or	FE EIE

الشكل(14): المقاطعات ذات الصلة بمنفذ الاتصال التسلسليUART

تتضمن هذه المقاطعات IRQs الخاصة بإرسال البيانات وأخطاء الاتصال، ويمكن تقسيمهم لمجموعتين:

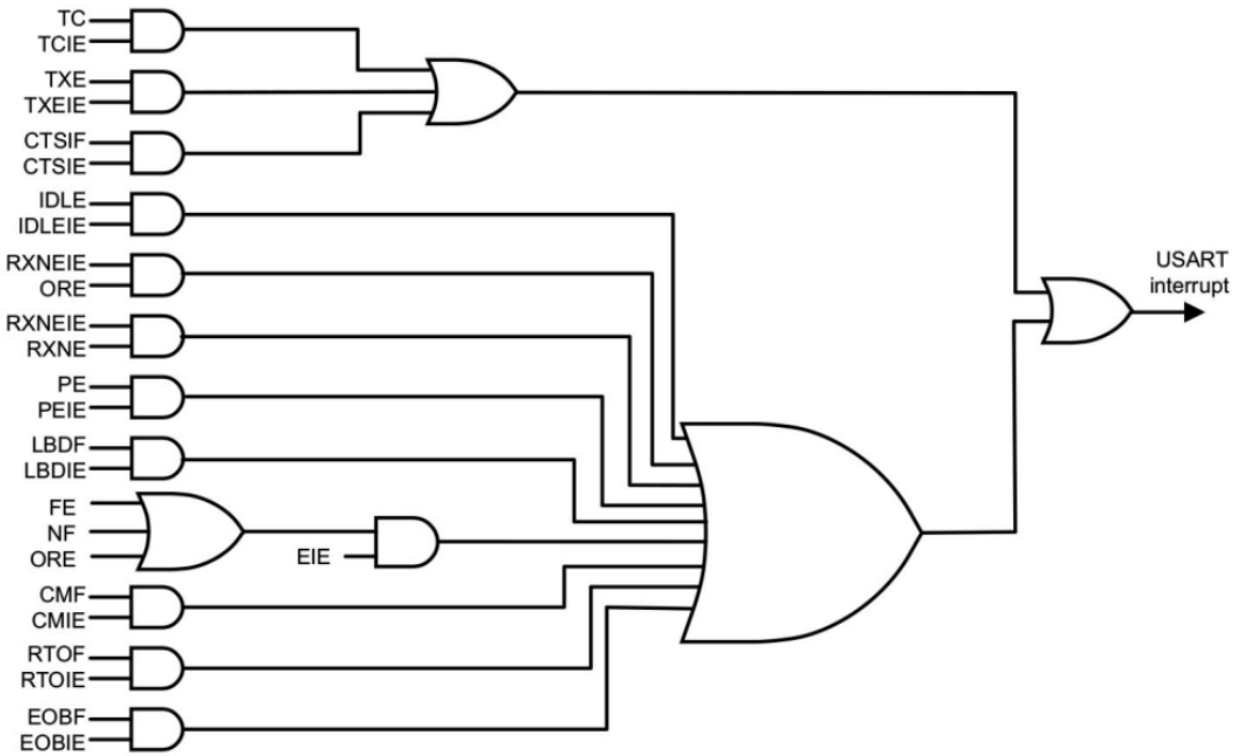
IRQs: التي يتم استدعائها أثناء الإرسال:

- اكتمال الارسال Transmission complete
- Clear to send(CTS)
- مسجل البيانات فارغ transmission Data Register Empty

IRQs: التي يتم استدعائها أثناء الاستقبال وهي:

- Idle line detection
- Overrun error
- مسجل الاستقبال غير فارغ Receive data register not empty
- Parity error
- Lin break detection
- Noise Flag
- Framing error

يتم تفعيل حدث المقاطعة Interrupt event لكل نوع من خلال Enable Control Bit الخاص به كما في الجدول السابق، حيث كل هذه الـ IRQs لها فقط خط مقاطعة وحيد لكل USART Peripheral كما هو موضح بالشكل التالي:

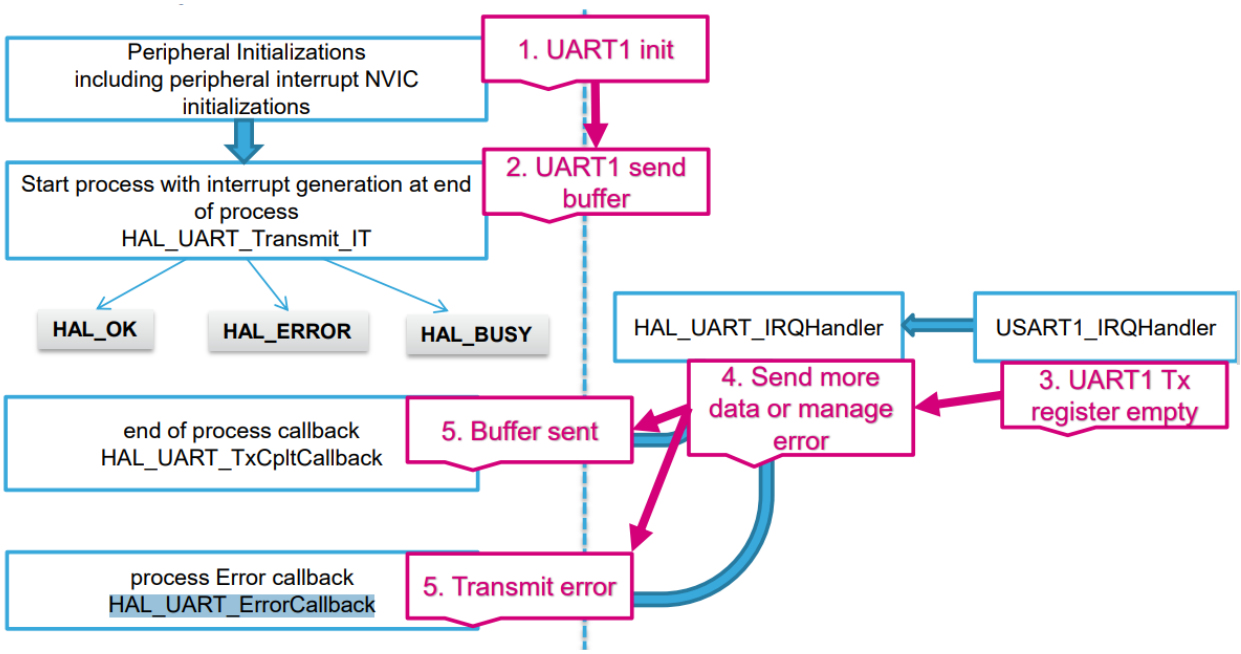


الشكل(15): أحداث المقاطعة Interrupt Events المتصلة بنفس Interrupt Vector

نلاحظ من الشكل السابق أن لكل وحدة USART في متحكمات STM32 خط مقاطعة وحيد وعلى المستخدم تحليل علم المقاطعة Flag Event الذي تم رفعه لمعرفة المقاطعة التي حدثت، حيث يتم تفعيل حدث المقاطعة من خلال تفعيل بت التحكم Enable Control Bit.

خطوات تفعيل واستخدام المقاطعة مع المنفذ التسلسلي:

يتم تهيئة و استخدام المقاطعة مع المنفذ التسلسلي وفق المخطط التالي:



الشكل(16): المخطط التدفقي لعملية إرسال/ استقبال بيانات HAL Library Receive Flow

الخطوة الأولى:

تهيئة المنفذ التسلسلي من خلال اختيار رقم المنفذ التسلسلي و ضبط معدل نقل البيانات BaudRate وتفعيل المقاطعة على المنفذ وغيرها من الإعدادات ويتم ذلك با ستخدام أداة الـ CubeMx - (التي أ صبحت مدمجة داخل بيئة STM32CubeIDE) ، وتلقائياً يتم إ ضافة سطور التهيئة عند توليد الكود ضمن ملف الـ main.c - والملف stm32G0xx_hal_msp.c.

الخطوة الثانية:

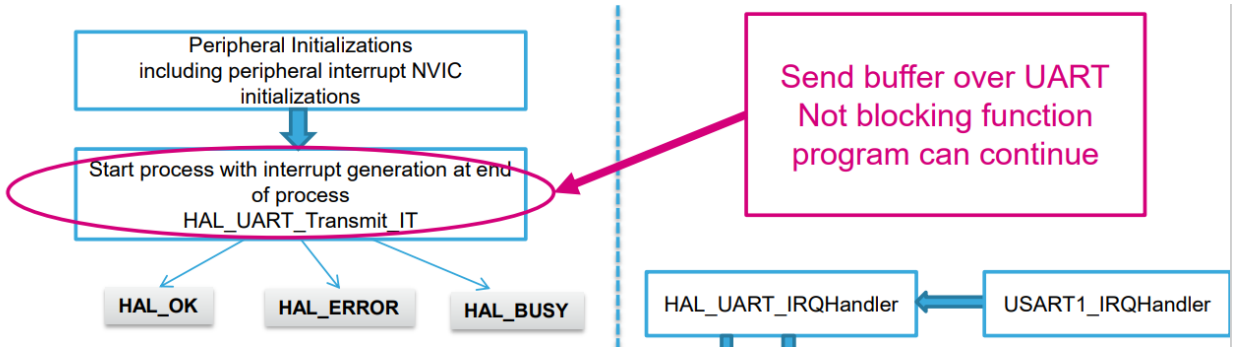
إرسال البيانات عبر المنفذ التسلسلي من خلال استخدام الدالة التالية:

```
HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
```

حيث يتم إدخال بارامترات هذه الدالة كما قمنا بالشرح سابقاً ، وكما تلاحظ فقد تم إ ضافة IT في ا سم الدالة وأيضاً تم إزالة Timeout من بارامترات هذه الدالة مقارنةً بالدالة الم ستخدمة في نمط الـ Polling - ، لأنه لم يعد هناك زمن انتظار في نمط المقاطعة.

لاستقبال مجموعة من البايتات عبر المنفذ التسلسلي نستخدم الدالة التالية:

```
HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
```



الشكل(17):مهمة دالة الإرسال

تعيد دالة الإرسال أو الاستقبال إما HAL_OK في حال تمت عملية الإرسال/الاستقبال بنجاح، أو HAL_error في حال حدوث خطأ أثناء عملية الإرسال/ الاستقبال أو HAL_Busy .

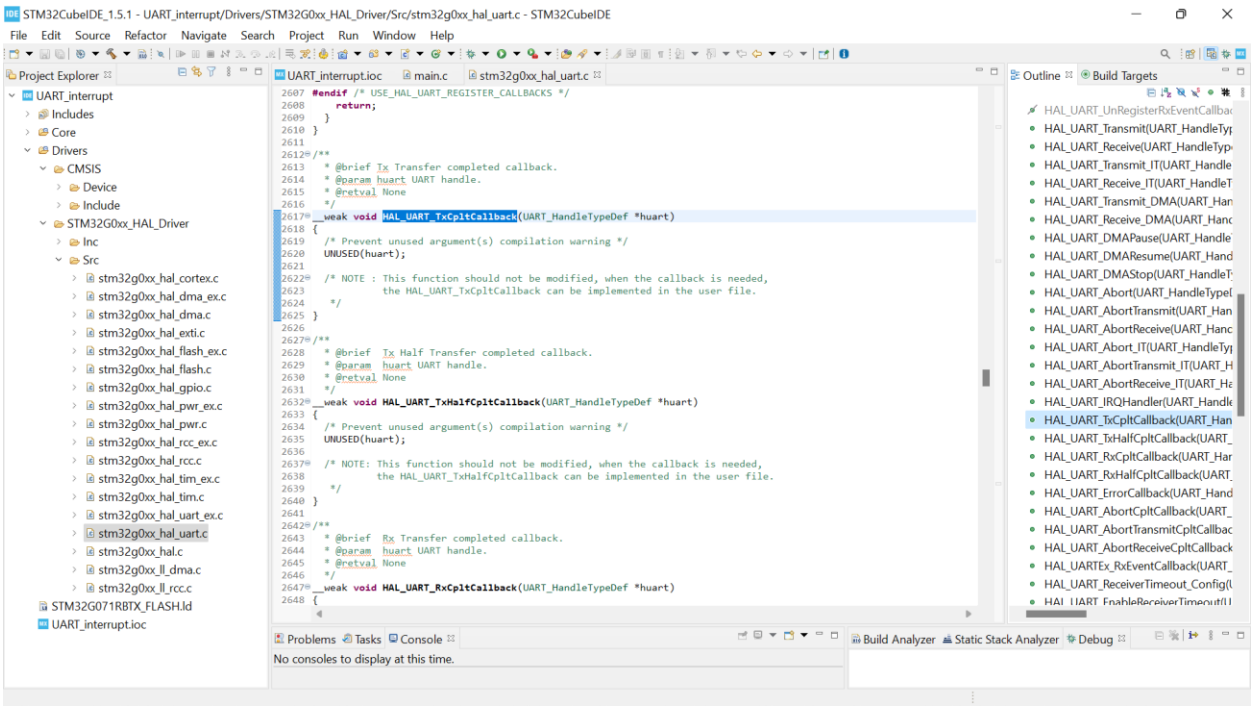
الخطوة الثالثة: تفحص فيما إذا أصبح مسجل البيانات UART1_TX فارغاً

الخطوة الرابعة: في حال كان م سجل البيانات UART1_TX فارغاً (يتم اكت شافه من خلال حدث المقاطعة وبالتالي ا ستدعاء HAL_UART_IRQHandler والمعرف ضمن ملف stm32G0xx_hal_uart.c) فلدينا خيارين إما إر سال البيانات المتبقية في حال كان الـ Buffer - غير فارغ أو الإعلان عن انتهاء عملية الإر سال ، أو الكشف عن وجود خطأ في عملية الإرسال.

الشكل(18):فحص مسجل بيانات UART1_TX

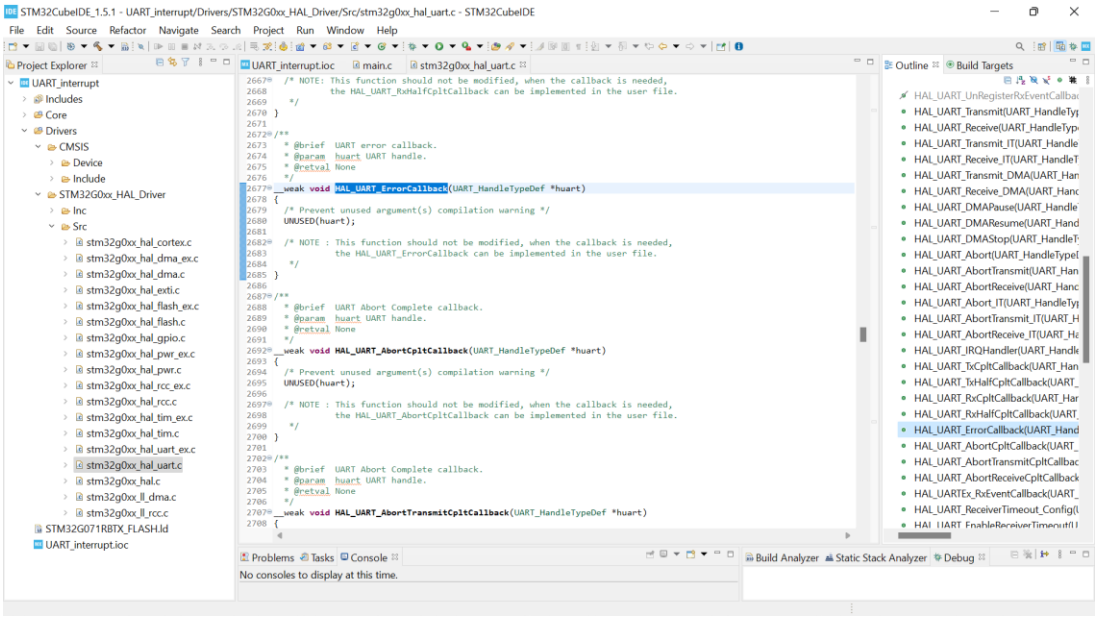
الخطوة الخامسة: لدينا خيارين:

في حال تم إر سال البيانات بنجاح عندها يتم ا ستدعاء الدالة HAL_UART_TxCpltCallback والتي تكون معرفة ك - _weak (أي يتم ا ستدعائها في حال لم يتم إعادة تعريفها ضمن البرنامج الرئي سي) ضمن ملف stm32G0xx_hal_uart.c ضمن مجلد stm32G0xx_HAL_Driver ثم المجلد Src ، نقوم بنـ سخ الدالة HAL_UART_TxCpltCallback ووضعها ضمن ملف البرنامج الرئيسي main.c لاستخدامها لاحقاً.



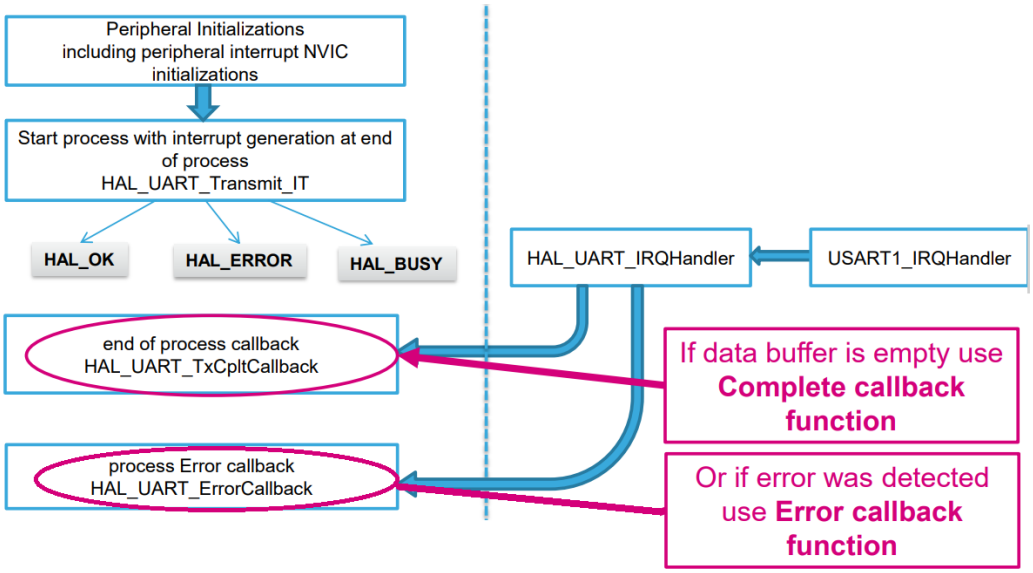
الشكل(19):نسخ دالة HAL_UART_TxCpltCallback من ملف stm32g0xx_hal_uart.c

في حال تم اكتشاف خطأ أثناء إرسال البيانات يتم استدعاء الدالة HAL_UART_ErrorCallback والتي تكون معرفة كـ weak_ (أي يتم استدعائها في حال لم يتم إعادة تعريفها ضمن البرنامج الرئيسي) ضمن ملف stm32G0xx_hal_uart.c ضمن مجلد stm32G0xx_HAL_Driver ثم المجلد Src كما في الشكل التالي:



الشكل(20): نسخ دالة HAL_UART_ErrorCallback من ملف stm32g0xx_hal_uart.c

نقوم بنسخ الدالة HAL_UART_ErrorCallback ووضعها ضمن ملف البرنامج الرئيسي main.c لاستخدامها لاحقاً.

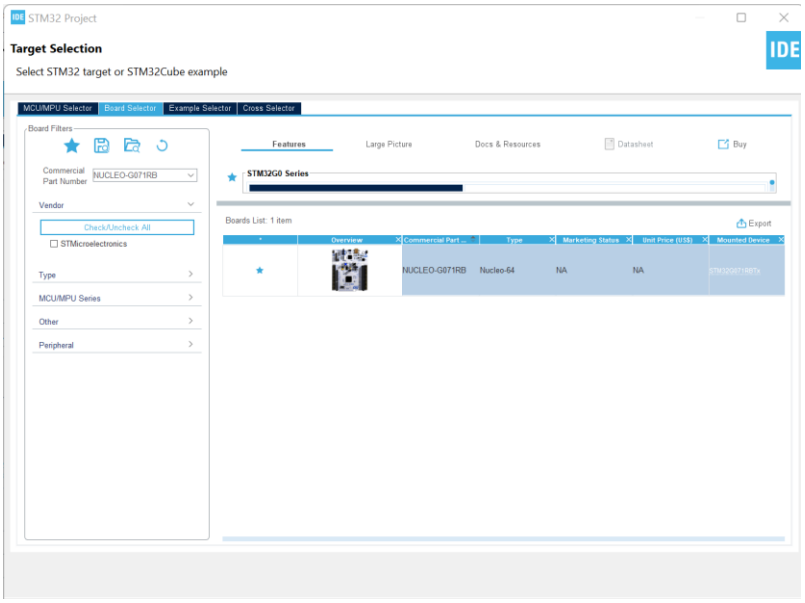


الشكل(21): استدعاء الدالة المناسبة

من أجل فهم أكبر لا ستخدام المقاطعة مع منفذ الاتـ صال التـ سلـ سلي UART سنقوم بتو ضيح ذلك من خلال تطبيق عملي.

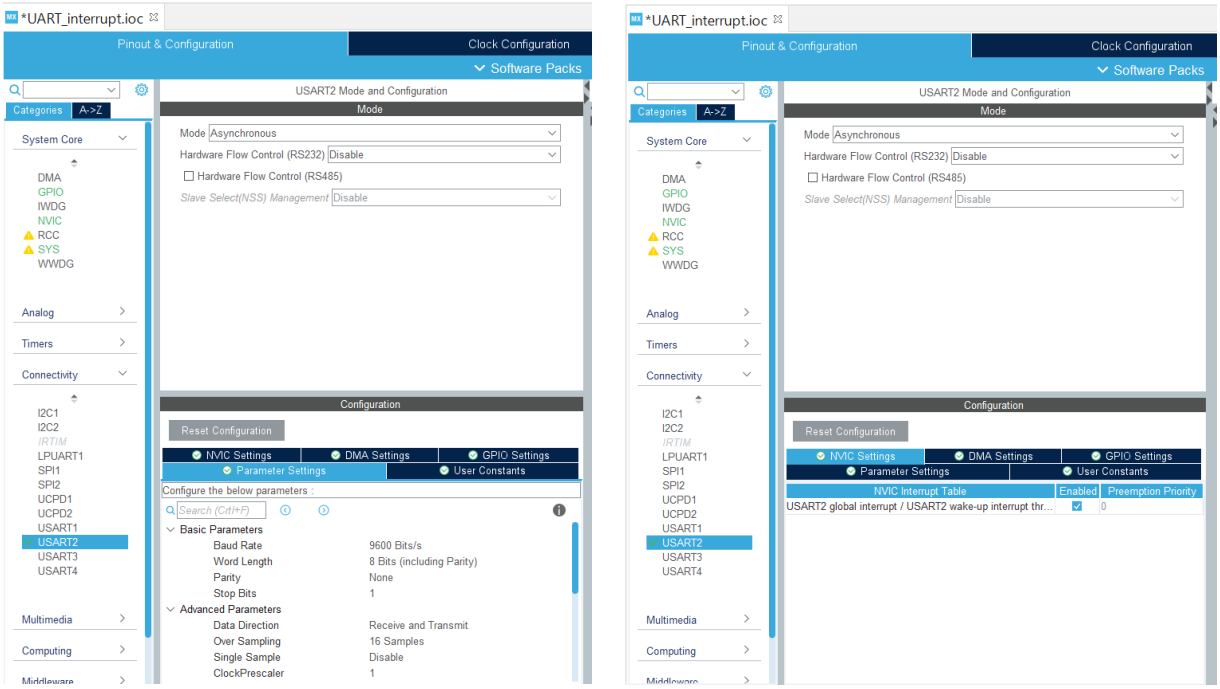
7- تطبيق عملي لاستخدام المنفذ التسلسليUSART من خلال نمط الـ interrupt:

الخطوة الأولى: قم بفتح برنامج STM32CubeIDE ومن ثم قم بإ نشاء م شروع جديد من نافذة File ثم New ثم STM32Project ثم قم باختيار المتحكم المصغر أو من خلال اختيار ا سم اللوحة المستخدمة وهي في حالتنا Nucleo-G071RB كما في الشكل التالي:



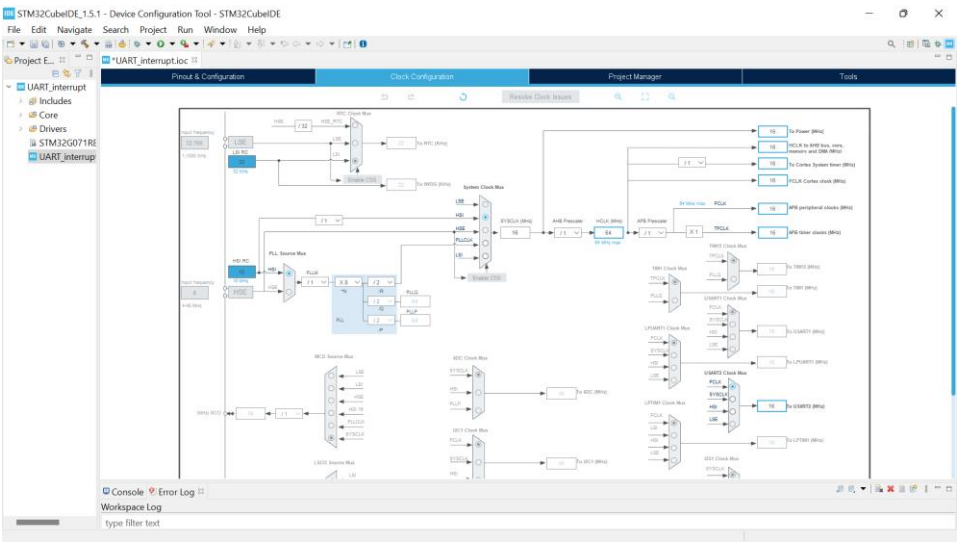
الشكل(22): إنشاء مشروع جديد

الخطوة الثانية: قم بضبط إعدادات المنفذ UART2 المتصل داخلياً مع دارة الـ ST-LINK المدمجة مع اللوحة، اختر معدل نقل البيانات 9600 (عند استخدام مصدر الساعة الداخلية للمتحكم فلا يمكن للمتحكم نقل البيانات عبر منفذ الـ UART بسرعة عالية) وقم بتفعيل المقاطعة للمنفذ التسلسلي من خلال شريط الـ NVIC ، كما في الشكل التالي:



الشكل(23):ضبط إعدادات المنفذ التسلسلي

الخطوة الثالثة: نقوم بضبط إعدادات ساعة النظام كما هو موضح بالشكل التالي:



الشكل(24):ضبط إعدادات ساعة النظام

ثم نقوم بتوليد الكود من خلال الضغط على ctrl+s فيتم بتوليد الكود آلياً.

الخطوة الرابعة: قم بكتابة الكود بالشكل التالي(طبعاً هناك قسم كبير من الكود تم توليده من خلال MX وقمنا نحن فقط بإكمال الكود بما يناسب التطبيق)

`#include "main.h"`	
`UART_HandleTypeDef huart2;`	المنفذ التسلسلي الذي سيتم استخدامه هو UART2 وتم تسميته بـhuart2
`uint8_t rx_buffer[4];`	تعريف مصفوف لتخزين البيانات التي يتم استقبالها
`uint8_t tx[]="hello";`	تعريف مصفوفة محارف ليتم إرسالها إلى المنفذ التسلسلي
`void SystemClock_Config(void);`	تعريف الدالة الخاصة بتهيئة ساعة النظام
`static void MX_GPIO_Init(void);`	تعريف الدالة الخاصة بتهيئة أقطاب الدخل والخرج
`static void MX_USART2_UART_Init(void);`	تعريف الدالة الخاصة بتهيئة المنفذ التسلسلي

`int main(void)`
`{`
`  HAL_Init();`
`  SystemClock_Config();`
`  MX_GPIO_Init();`
`  MX_USART2_UART_Init();`
	بداية البرنامج الرئيسي
	تهيئة مكتبة HAL

`  HAL_UART_Transmit_IT(&huart2, tx, 5);`	إرسال المصفوفة tx إلى المنفذ التسلسلي
`  HAL_UART_Receive_IT(&huart2, rx_buffer, 4);`	استقبال بيانات من المنفذ التسلسلي وتخزينها ضمن المصفوفة rx_buffer

`  while (1)`
`  {`
 	
`  }`
`  }`	

`void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)`
`{`
`HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5)  ;`
`HAL_UART_Receive_IT(&huart2, rx_buffer, 4);`
	الدالة التي يتم استدعائها عند إتمام عملية استقبال البيانات(قمنا بنسخ اسمها من (stm32g0xx_hal_uart.c) حيث تكون معرفة كـ weak)
	لاحظ أننا قمنا بتغيير الحالة المنطقية لليد وإرسال البيانات التي تم استقبالها كي نتمكن من معرفة فيما

إذا تمت عملية الاستقبال بشكل صحيح أم لا، ثم قمنا بإعادة طلب مقاطعة الاستقبال مرة أخرى
 	
الدالة التي يذهب إليها المعالج في حال لم تتم عملية الاستقبال بشكل صحيح لأي سبب من الأسباب
 	
وضعنا هذه التعليمة فقط من أجل الـ debug.
	HAL_UART_Transmit_IT(&huart2, rx_buffer, 4);
	}
	
	void
	HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
	{
	__NOP();
	}