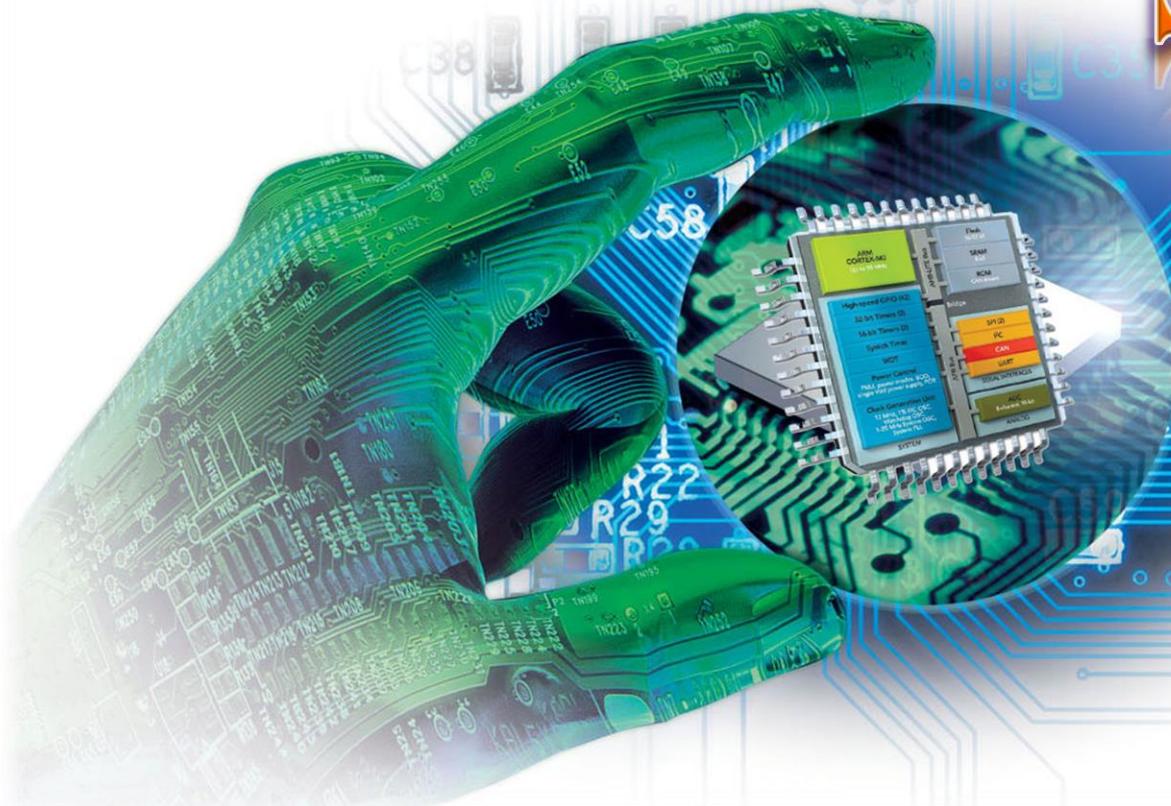


# مُتَحَكِّمات

# STM32

5



# موضع عاًن المعاشرة:

- مقارنة بين بروتوكولات الاتصال التسلسلي الشائعة
- مفهوم USART و UART
- أنماط العمل المختلفة لـ USART في متحكمات STM32
- دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في نمط الـ Polling
- تطبيق عملي لاستخدام المنفذ التسلسلي USART من خلال نمط الـ polling
- إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt
- دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في نمط الـ interrupt
- تطبيق عملي لاستخدام المنفذ التسلسلي USART من خلال نمط الـ interrupt

# مقارنة بين بروتوكولات الاتصال الشائعة:

	<b>RS232</b>	<b>RS485</b>	<b>I2C</b>	<b>SPI</b>	<b>M-wire</b>	<b>1-wire</b>	<b>USB</b>	<b>CAN</b>
<b>Sync/Async</b>	Async	Async.	Sync.	Sync.	Sync.	Async.	Async.	Async.
<b>Type</b>	peer-peer	master/slaves	multi-master	multi-master	master/slaves	master/slaves	host/device	multi-master
<b>Duplex</b>	full	half	half	full	full	half	half	half
<b>Signaling</b>	single-ended	Differential	single-ended	single-ended	single-ended	single-ended	Differential	Differential
<b>Max Devices No.</b>	2	32, 128, 256	40 (cap=400pf)	8 (cap, circuit)	8 (cap, circuit)	20 (cap, power)	127 per controller	2048
<b>Data Rate</b>	Up to 115Kbps	Up to 35Mbps	Std.: 100kbps Fast: 400kbps Hi: 3.4Mbps	Up to 10Mbps	Up to 1Mbps	Std.: 16.3Kbps Overdrive: 142kbps	Low: 1.5Mbps Full: 12Mbps Hi : 480Mbps	Up to 1Mbps
<b>Max. Length</b>	15m	1200m (at 100kbps)	6m	3m	3m	300m	5m	1000m (at 62kbps)
<b>Pin Count</b>	2* (Tx, Rx)	2 (A, B)	2 (SDA, SCL)	3 + SS* (SI, SO, SCK)	3 + SS* (DI, DO, SK)	1 (IO)	2 (A+, A-)	2 (CAN_H, _L)
<b>Interfacing</b>	HW	HW	SW   HW	HW   SW	HW   SW	HW & SW	protocol stack	HW & SW
<b>Flow Control</b>	HW or SW handshake	HW or SW handshake	Acknowledge from slave	None	None	CRC, Pulling	Polling by controller	CSMA / CDAMP

# مقارنة بين بروتوكول الاتصال التسلسلي والنفري:

المُرسِل

المُستَقِبِل



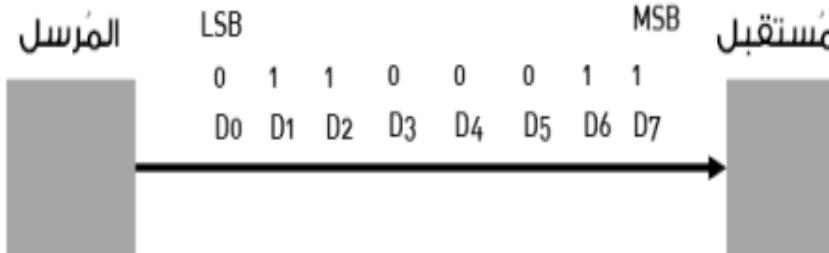
في الاتصال التفرعي يمكن إرسال عدة برات رقمية بنفس اللحظة الزمنية، ما يمنح سرعة كبيرة في نقل البيانات. بنفس الوقت، تتطلب عملية الاتصال التفرعي ضمان توافق ساعة المُرسِل والمُستَقِبِل والتأكد من عدم تشويش نواقل الاتصال على بعضها البعض.

المُرسِل

البت الأقل أهمية

MSB

المُستَقِبِل



في الاتصال التسلسلي يتم إرسال البارات الرقمية الواحد تلو الآخر مع كل نبضة ساعة، ما يعني معدل أبطأ لنقل البيانات، ولكنها أقل تطلبًا فيما يتعلق بمتطلبات توافق الساعة وتشويش نواقل الاتصال.

# النافذة التسلسليّة USART/UART

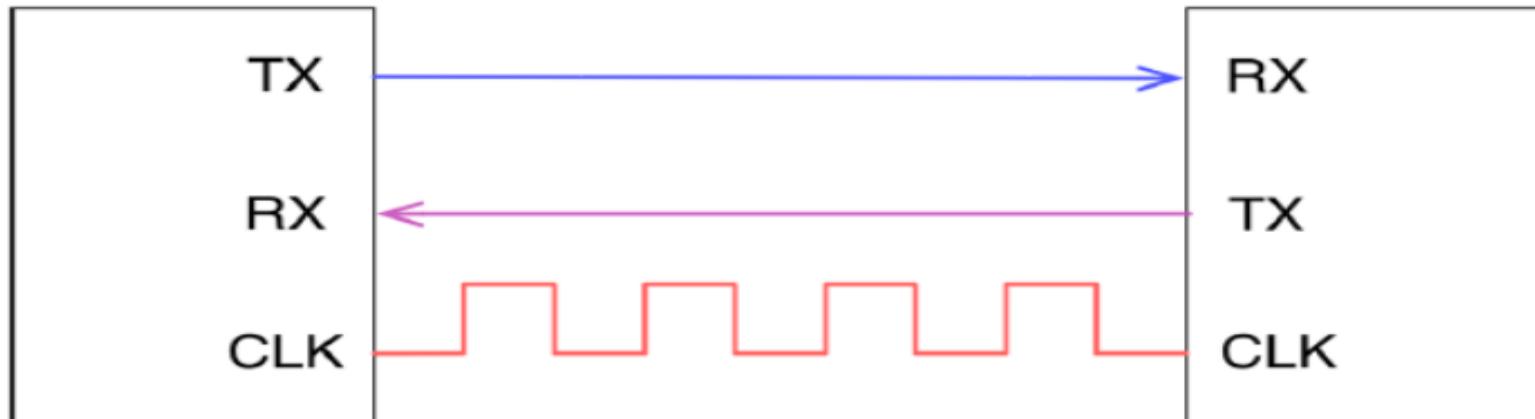
Unleash your Creativity!

كل متحكم STM32 يحتوي على الأقل على وحدة طرفية UART واحدة، وأغلب متحكمات STM32 توفر على الأقل اثنتين من آخرى توفر لحد 8 وحدات طرفية USART/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.

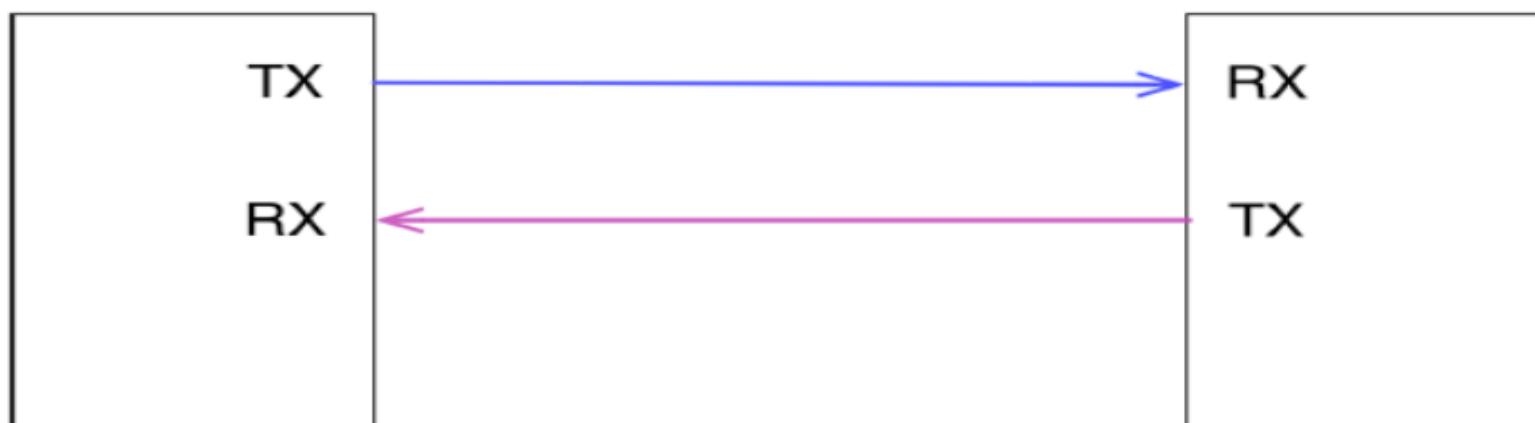
# النافذة التسلسليّة USART/UART



Device A

**USART**

Device B



Device A

**UART**

Device B

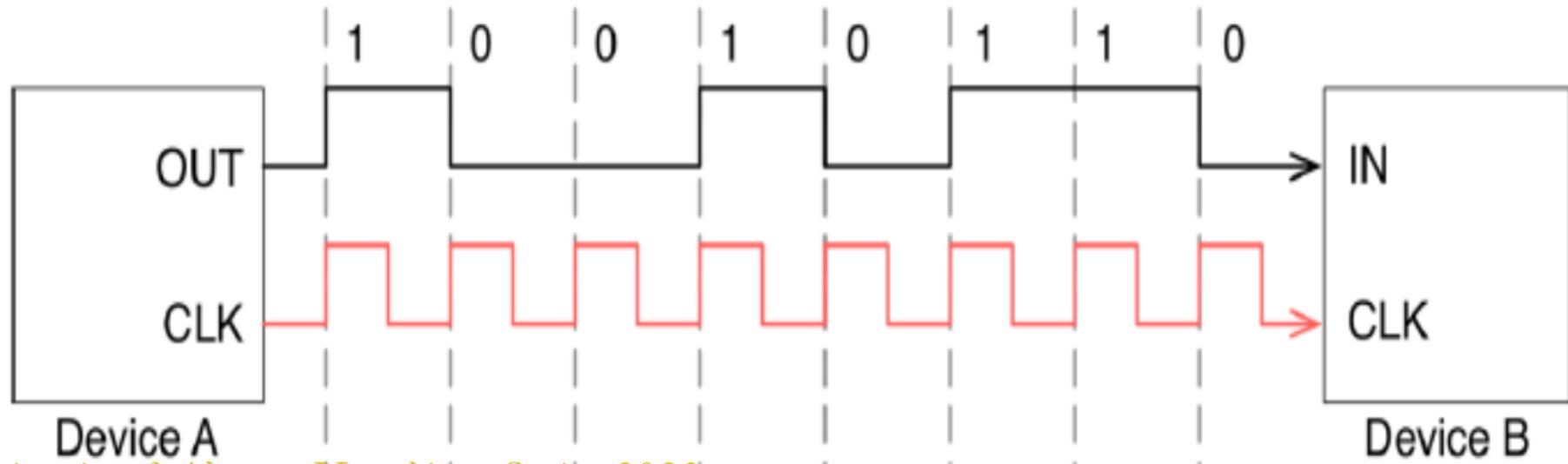
# USART

**Universal Synchronous and Asynchronous  
serial Receiver and Transmitter**



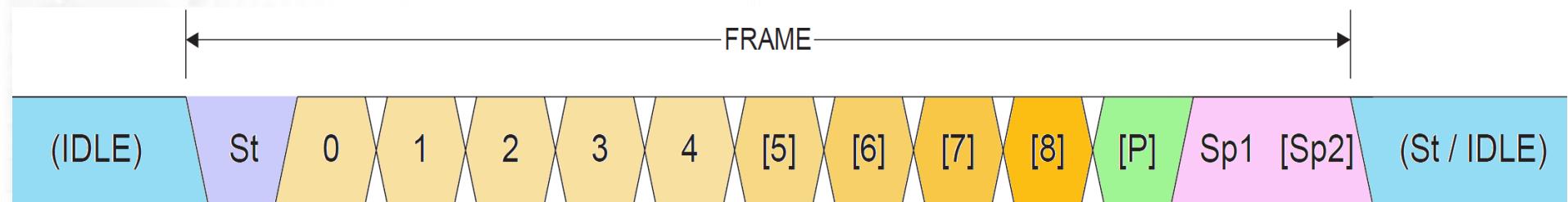
# النافذة التسلسليّة USART

المخطط الزمني لعملية الإرسال التسلسلي المتزامن  Synchronous لبيان واحد بقيمة  $0b01101001$  من الجهاز إلى الجهاز Device B حيث تم استخدام Clock Device A لضبط توقيت إرسال البيانات، حيث يتم إرسال بت واحد مع كل جبهة صاعدة لـ Clock، حيث تتعلق سرعة نقل البيانات بتردد الـ Clock، فكلما زاد تردد الـ Clock كلما زادت سرعة نقل البيانات.



## (Universal Asynchronous Receiver and Transmitter Interface)

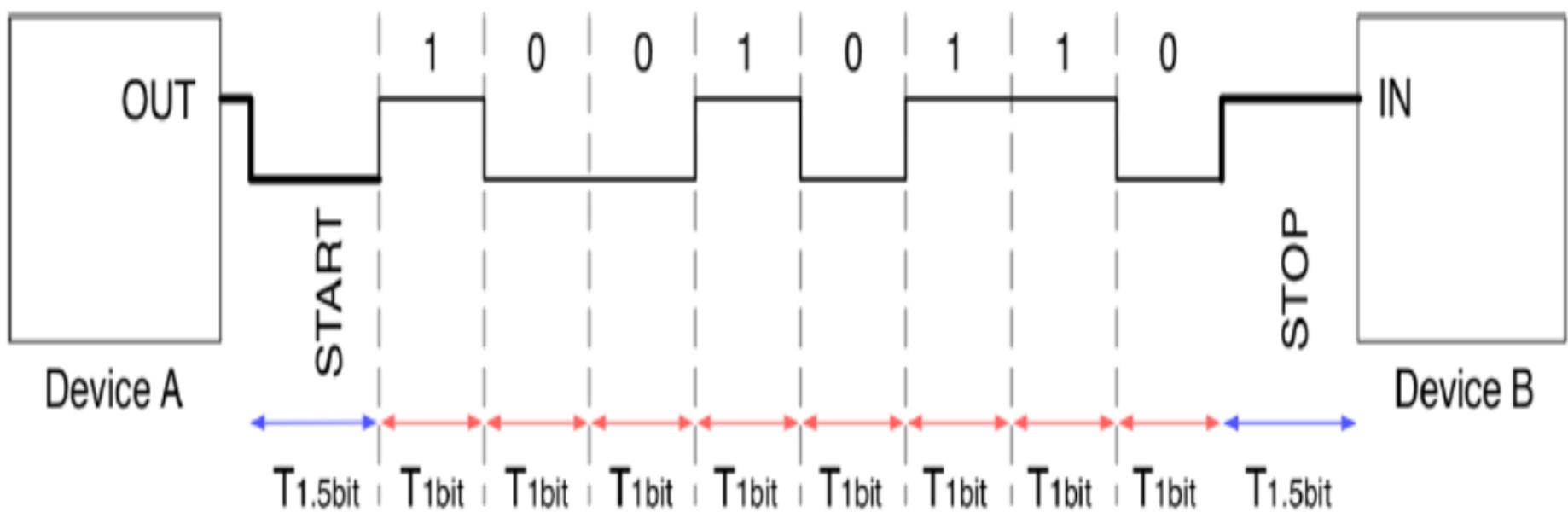
تعتبر هذه النافذة من أكثر نوافذ الاتصال التسلسلي استخداماً في الأنظمة الرقمية والمتحكمات المصغرة والطيفيات التي يمكن أن تربط معها...



بنية إطار البيانات (UART Frame Format)

# النافذة التسلسليّة UART

□ 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**



# النافذة التسلسليّة UART

- تمثل الحالة الخامّلة **Idle state** بإشارة **HIGH** وهي حالة عدم الإرسال
- بداية الإرسال تتم من خلال **Start Bit** وتمثل بإشارة **LOW**
- هذه الإشارة يتم اكتشافها من قبل المستقبل وتستغرق وقت  $T = 1.5$   $T$  حيث أن  $T$  هو الدور وهو عبارة عن مقلوب التردد أو ما يسمى **Baud Rate** أي معدل نقل البيانات بين المرسل والمستقبل
- بعد ذلك يتم إرسال الـ **8 bit** 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 إنتهاء الإرسال بـ **Stop Bit**

# أنماط العمل المختلفة لـ **STM32** في متحكمات **UART**

نقطة  **Blocking Mode**: يسمى أيضاً **Polling Mode**، في هذا النمط يتم تفحص 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

نقطة  **non-Interrupt Mode**: ويسمى أيضاً **Blocking Mode**، في هذا النمط لا يتم الانتظار وتفقد البيانات من حين لآخر للتأكد من عملية الإرسال 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 38400 Bps)

# أنماط العمل المختلفة لـ **STM32** في متحكمات **UART**

□ **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** يمكن التعامل مع السرعات العالية في الـ **UART**

هناك بارامترات يجب تحديدها بين المرسل والمستقبل قبل إرسال البيانات في الاتصالات غير المتوافقة وهي:

- ✓ معدل سرعة الإرسال :**(Baud Rate)** ...1200, 2400, 9600
- ✓ خانة فحص الإيجابية :**(Parity Bit)** .Even, Odd, or None
- ✓ عدد البتات :**(Data Bits)** .6, 7, 8, or 9-bit
- ✓ عدد برات التوقف :**(Stop Bit)** .1 or 2

هناك **نماطين** للبيانات في الاتصالات التسلسليّة وهما:

(1) **نمط الأسكى** (Ascii Mode)

123 > 3-byte

(2) **النمط الثنائي** (BIN Mode)

123 > 1-Byte

# مفاهيم أساسية في الاتصالات التسلسليّة:

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

## مفاهيم أساسية في الاتصالات التسلسليّة:

: وتعني عدد البتات التي يتم إرسالها أو WordLength  استقبالها في Frame في المرة الواحدة، وتتوفر 3 قيم يمكن الاختيار بينها 7bit,8bit,9bit حيث لا يتضمن هذا الرقم البتات الخاصة بـ Start وـ Stop وغيرها

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

## مفاهيم أساسية في الاتصالات التسلسليّة:

□ **Parity**: هو عبارة عن اختبار يستخدم لاكتشاف الأخطاء أثناء عملية الإرسال والاستقبال للبيانات من خلال الـ USART، وهو عبارة عن بت يكون مكانه عند البت الأكثر أهمية MSB حيث لو تم استخدام 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**: تكون قيمة بت الـ Parity مساوٍ للواحد المنطقي عندما يكون عدد الواحات الموجودة في الكلمة المراد إرسالها زوجي، وصفر منطقي في حال كان عدد الواحات الموجودة في الكلمة المراد إرسالها فردي.

□ **زوجي Even**: تكون قيمة بت الـ Parity مساوٍ للواحد المنطقي عندما يكون عدد الواحات الموجودة في الكلمة المراد إرسالها فردي، وصفر منطقي في حال كان عدد الواحات الموجودة في الكلمة المراد إرسالها زوجي.

## على سبيل المثال:

- عندما تريدين إرسال أي بيانات يتم تحويلها إلى **Binary** فمثلاً إذا كنا نريد إرسال الكلمة التالية **0b01101110** فمن خلال الـ **Parity** يتم حساب عدد الوحدات الموجودة ضمن هذه الكلمة المراد إرسالها وهي في هذه الحالة 5، ففي حال كنت تستخدم نمط الفردي ستكون قيمة الـ **Parity** صفر، أما في حال كنت تستخدم نمط الزوجي ستكون قيمة الـ **Parity** واحد.
- يتم 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.

# دوال مكتبة HAL المستخدمة للتتعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

سنستخدم دالتين رئيسيتين للتتعامل مع المنفذ التسلسلي إحداهما للارسال والأخرى للاستقبال:  
دالة الإرسال:

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

حيث:

هو مؤشر يشير إلى : **huart**  أي المنفذ التسلسلي المستخدم للاتصال مثلاً قد يكون **huart1 & huart2** أو **huart3 & huart4**

# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

سنستخدم دالتين رئيسيتين للتعامل مع المنفذ التسلسلي إحداهما ل لإرسال والأخرى ل الاستقبال:

□ دالة الإرسال:

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

حيث:

□ Data : وهو مؤشر أيضاً يشير إلى البيانات التي سيتم إرسالها عبر UART وكما نرى نوعه uint8\_t أي قبل إرسال بيانات من نوع و بطول bit8، مثال: قد تكون pData مصفوفة ول يكن اسمها Data وتكون معرفة بالشكل التالي:

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

# دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

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

□ دالة الإرسال:

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

حيث:

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

# دوال مكتبة HAL المستخدمة للتتعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

سنستخدم دالتين رئيسيتين للتتعامل مع المنفذ التسلسلي إحداهما للارسال والأخرى للاستقبال:

□ دالة الإرسال:

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

حيث:

□ **Timeout**: أقصى زمن يتم انتظاره بالميلي ثانية حتى يتم اكتمال عملية الإرسال، فإذا تم انتهاء هذا الزمن ولم تتم عملية الإرسال سيتم قطع عملية الإرسال وتقوم الدالة برجوع **HAL\_TIMOUT** ماعدا ذلك يتم ارجاع **HAL\_OK**، ويمكن استخدام الدالة **HAL\_MAX\_DELAY** وهي وظيفتها انتظار أقصى زمن ممكن لعملية الإرسال

# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

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 */
```

# دوال مكتبة HAL المستخدمة للتتعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

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:

هو مؤشر يشير إلى **huart**: أي المنفذ التسلسلي Struct\_UART\_HandleTypeDef المستخدم للاتصال مثلاً قد يكون **huart1** & **huart2** أو **huart3** & **huart4**.

# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

دالة الاستقبال:

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

حيث:

: وهو مؤشر أيضاً يشير إلى البيانات التي سيتم استقبالها عبر Data    
وكما نرى نوعه uint8\_t أي يقبل استقبال بيانات من نوع UART   
وبطول bit8، مثال: قد تكون pData مصفوفة ولتكن Unsigned int   
اسمها Data و تكون معرفة بالشكل التالي:

```
Uint8_t Data[10];
```

# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

دالة الاستقبال:

```
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  
وهي في المثال التالي 10. ☐

Timeout: أقصى زمن يتم انتظاره بالميلي ثانية حتى يتم اكتمال  
عملية الاستقبال، فإذا تم انتهاء هذا الزمن ولم تتم عملية الاستقبال سيتم  
قطع عملية الاستقبال وتقوم الدالة بإرجاع HAL\_TIMOUT ماعدا  
ذلك يتم إرجاع HAL\_OK، ويمكن استخدام الدالة  
ووظيفتها انتظار أقصى زمن ممكن لعملية  
استقبال.

# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Polling

دالة الاستقبال:

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

مثال:

```
/* 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 */
```

# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

توفر جميع متحكمات UART مقاطعات ل STM32 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	

# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ **interrupt**

يمكن تقسيم المقااطعات **IRQs** الخاصة بالمنفذ التسلسلي **UART** لمجموعتين:

: **IRQs**  التي يتم استدعائهما أثناء الإرسال:

- **Transmission complete**
- **Clear to send(CTS)**
- **transmission Data Register** مسجل البيانات فارغ
- **Empty**
- **Noise Flag**
- **Framing error**

# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

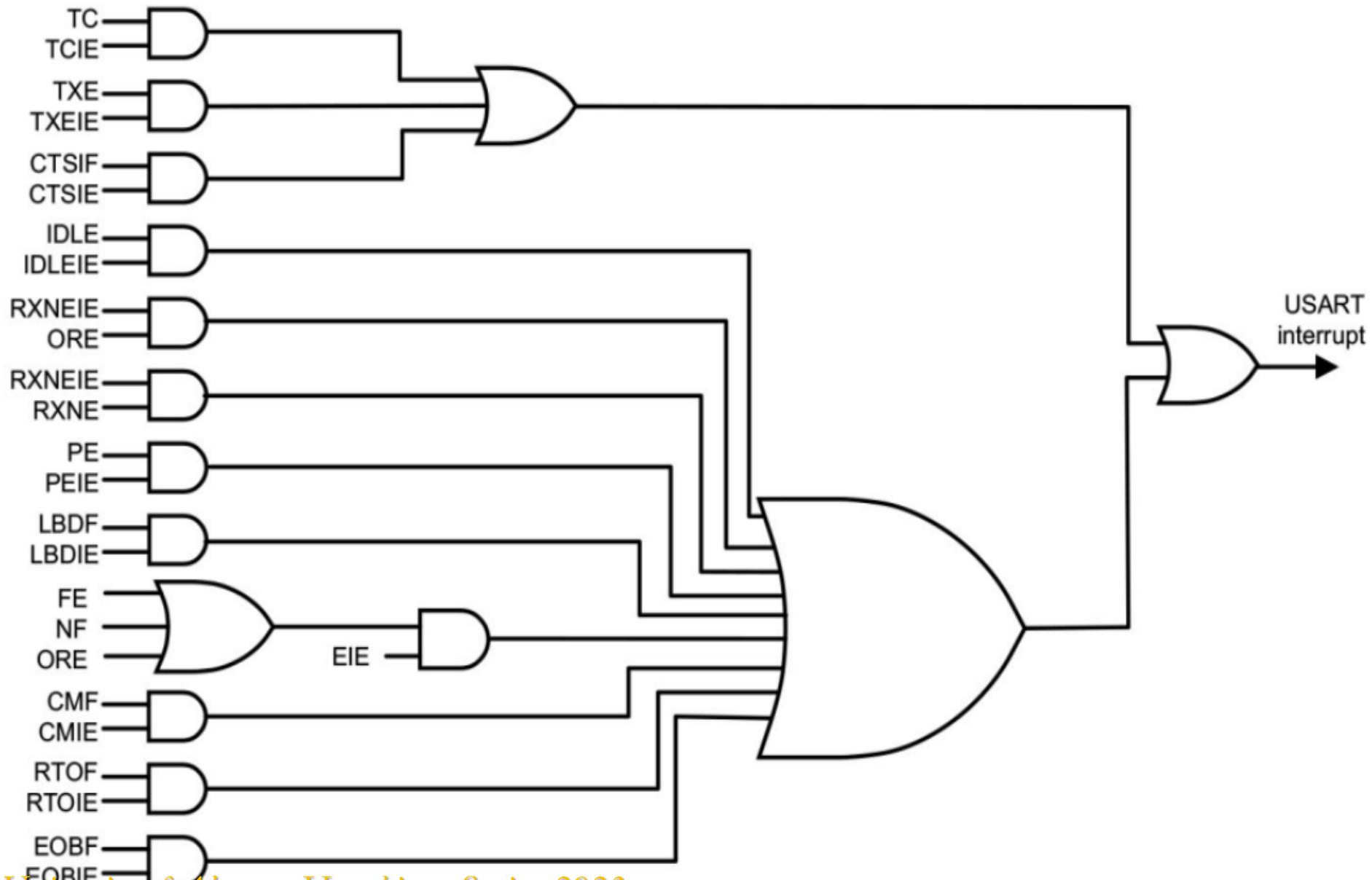
: التي يتم استدعائها أثناء الاستقبال: IRQs

- Idle line detection
- Overrun error
- مسجل الاستقبال غير فارغ empty
- Parity error
- Lin break detection
- Noise Flag
- Framing error

## إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ **interrupt**

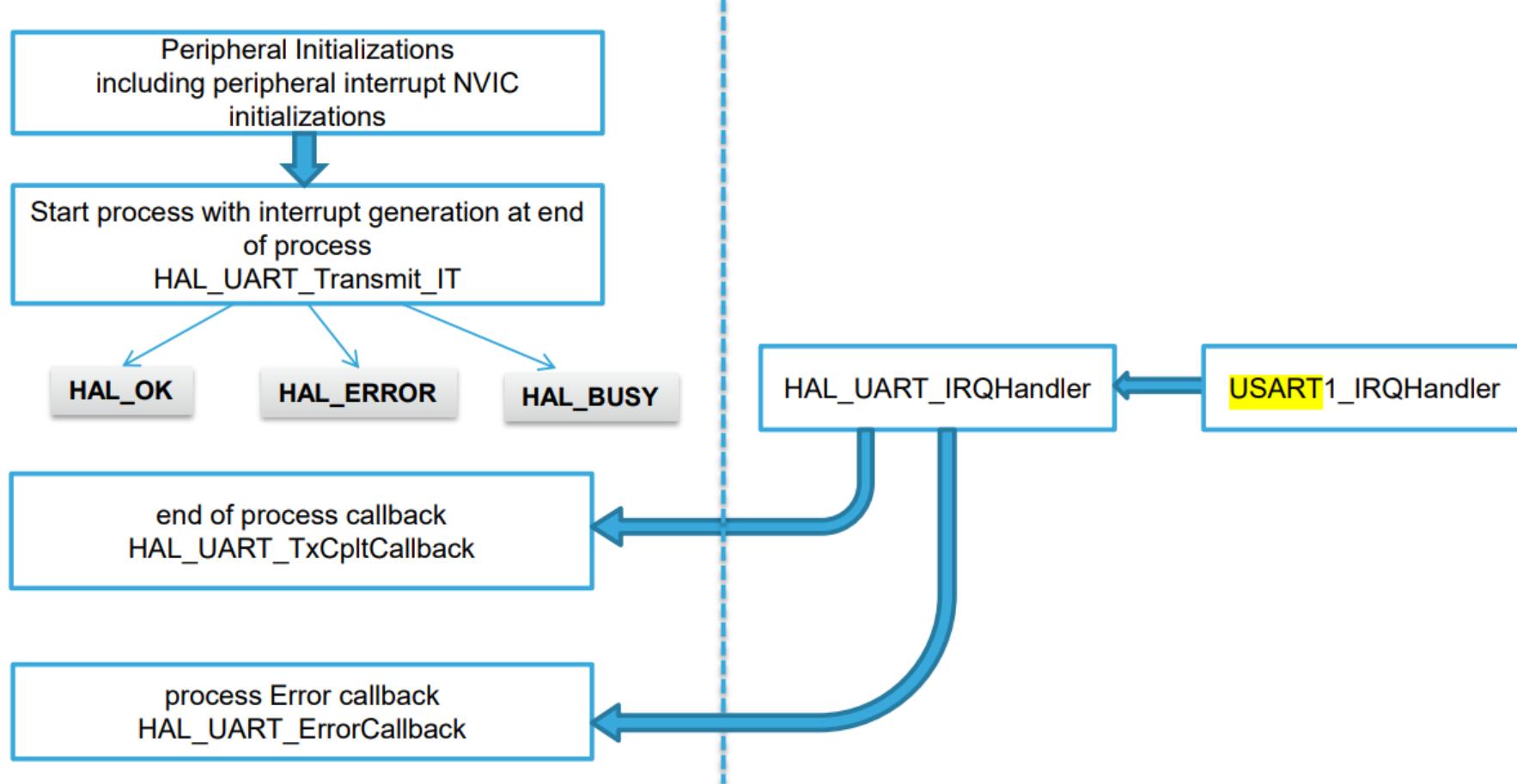
- يتم تفعيل حدث المقاطةة Interrupt event لكل نوع من خلال Enable Control Bit الخاص به كما في الجدول السابق، حيث كل هذه الـ USART Peripheral لها فقط خط مقاطعة وحيد لكل IRQs
- باعتبار أن لكل وحدة USART في متحكمات STM32 خط مقاطعة وحيد لذا يتوجب على المستخدم تحليل علم المقاطةة Flag Event الذي تم رفعه لمعرفة المقاطةة التي حدثت

# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt



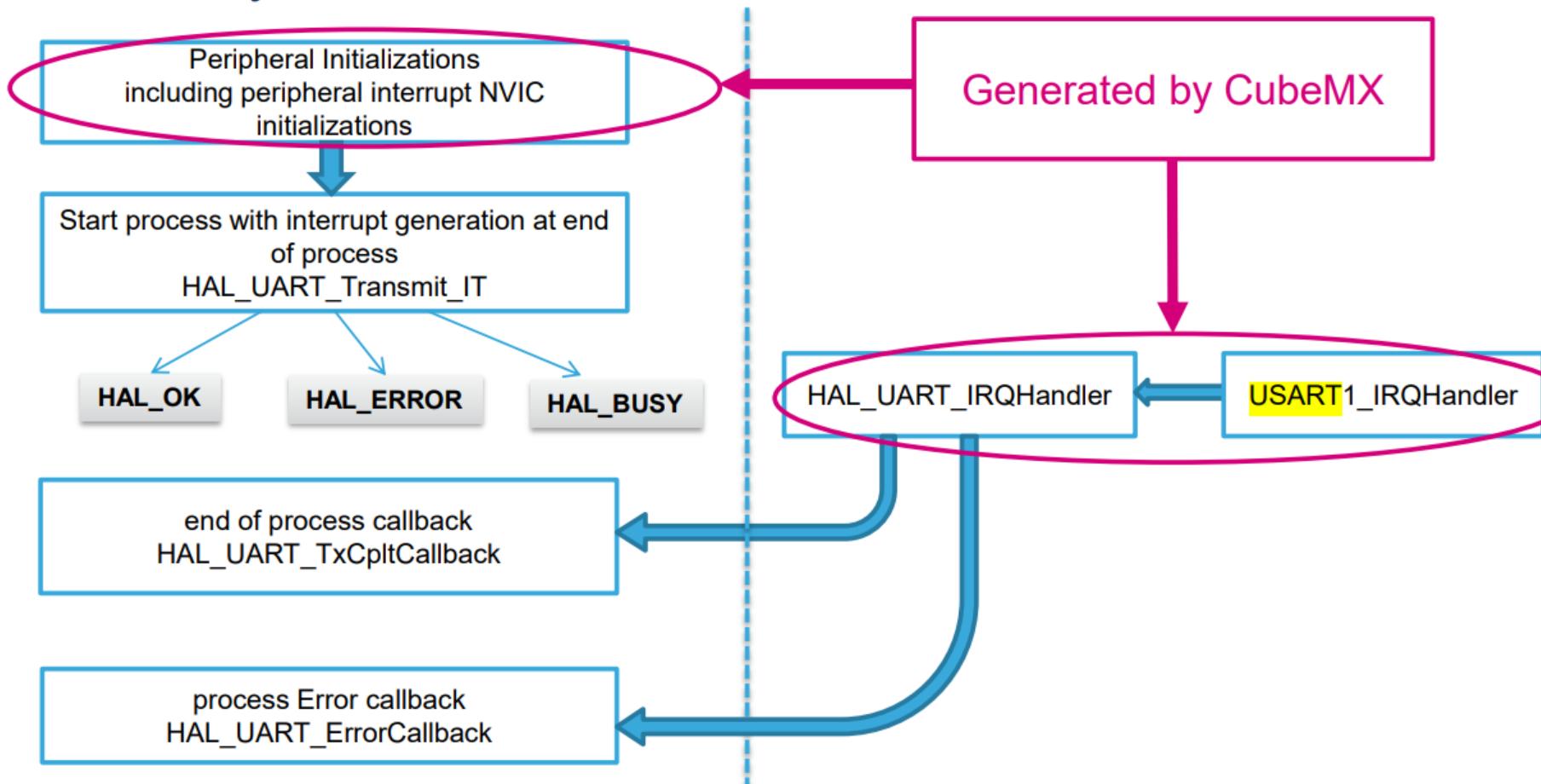
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT transmit flow



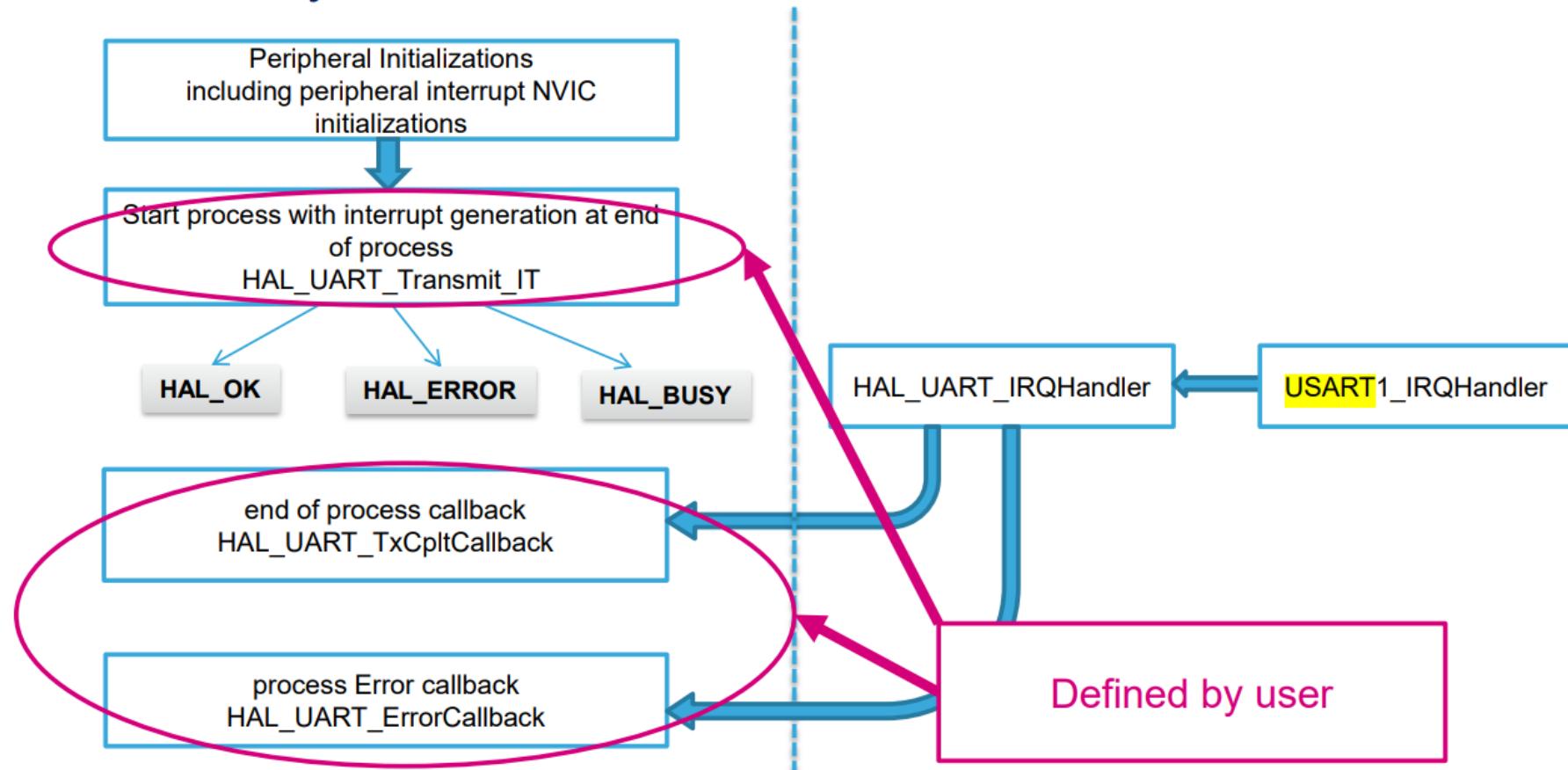
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT transmit flow



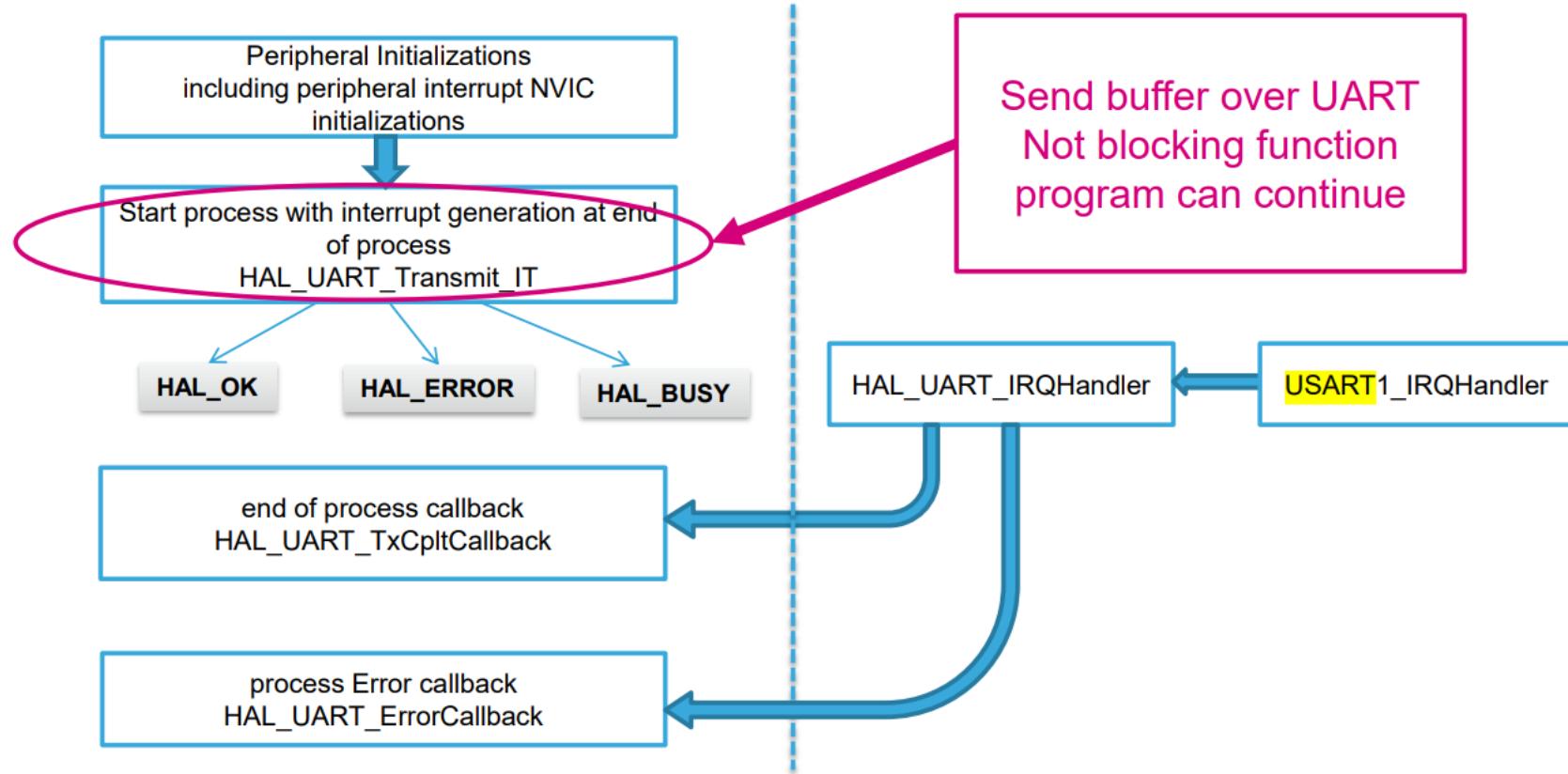
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



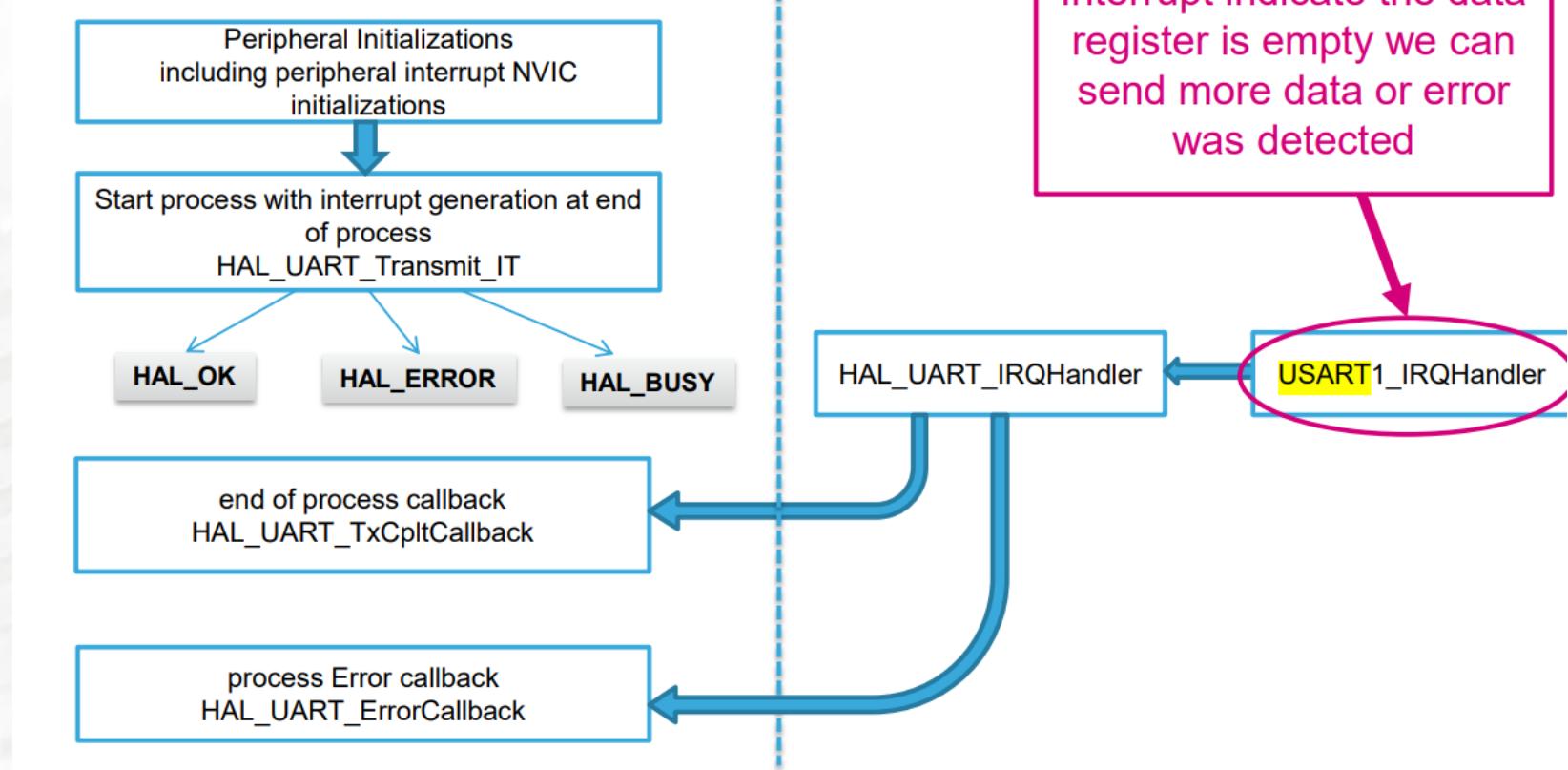
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



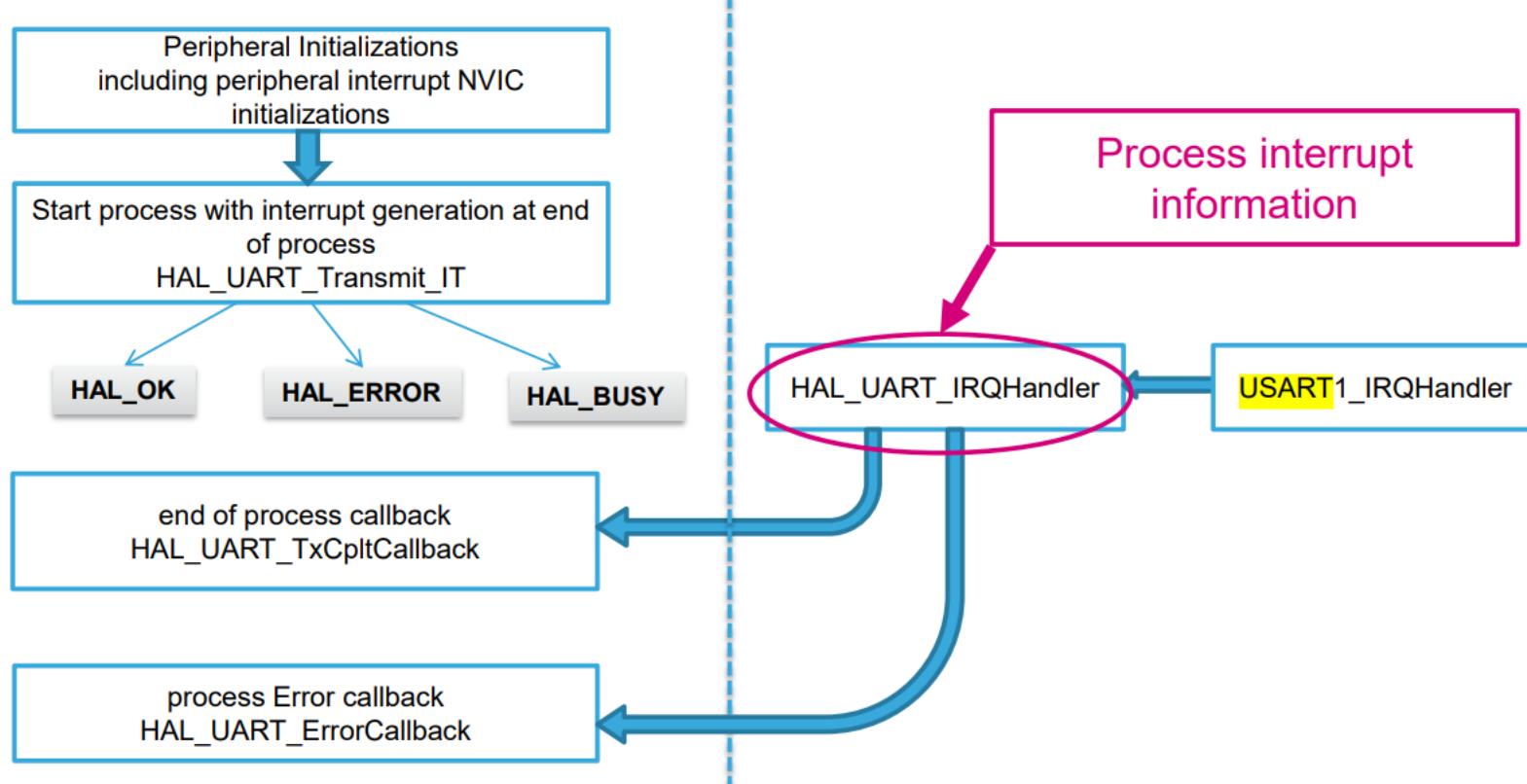
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



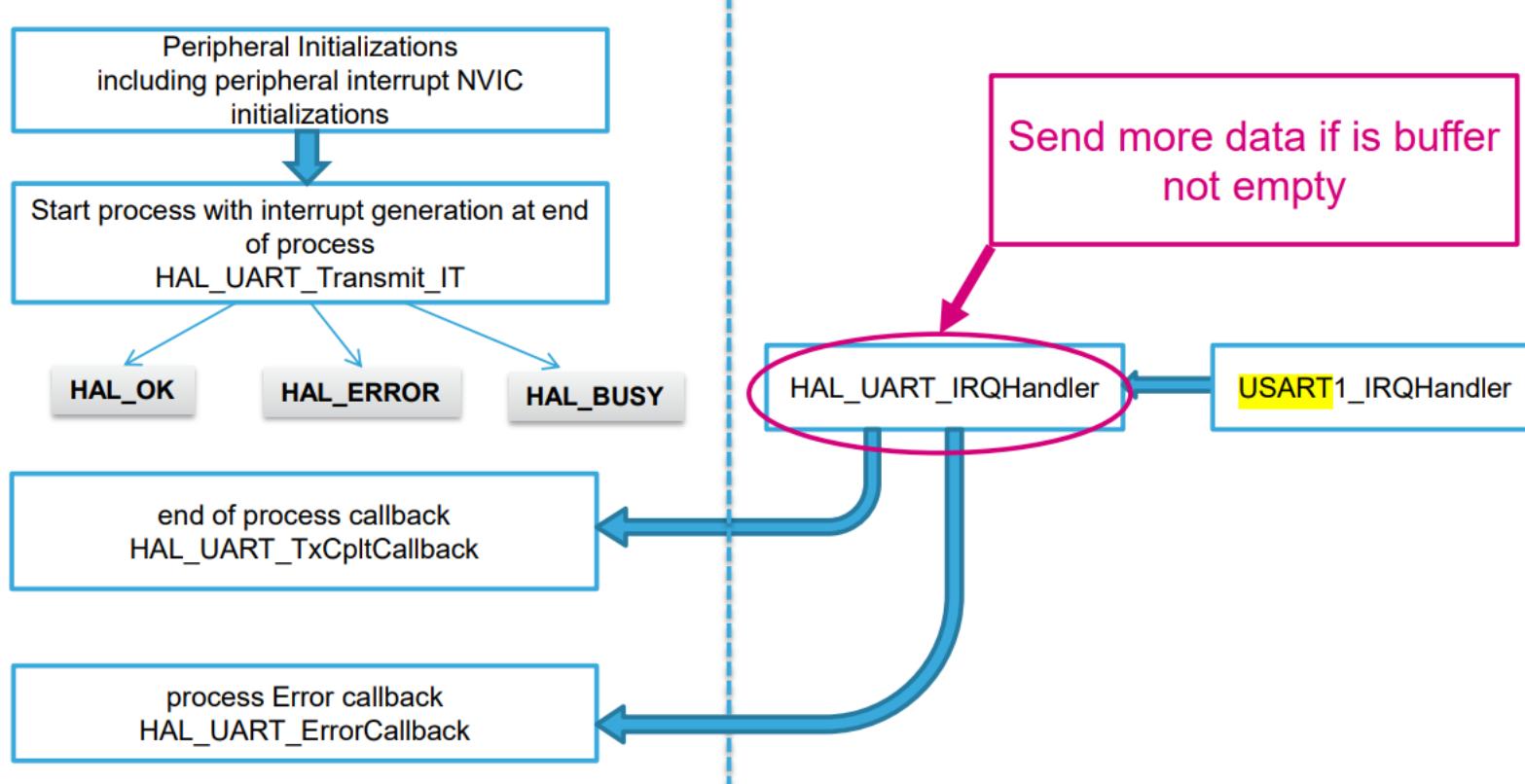
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



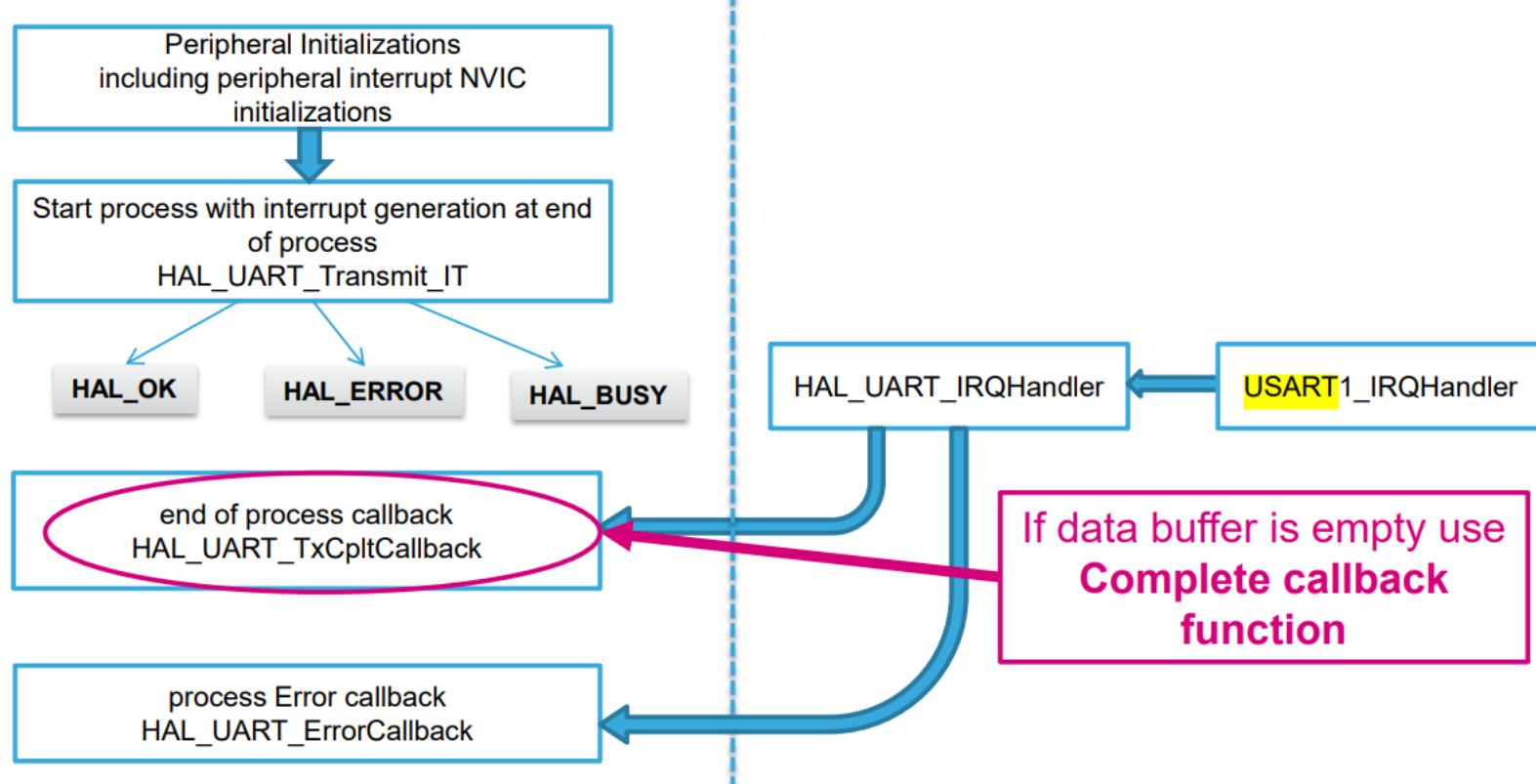
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



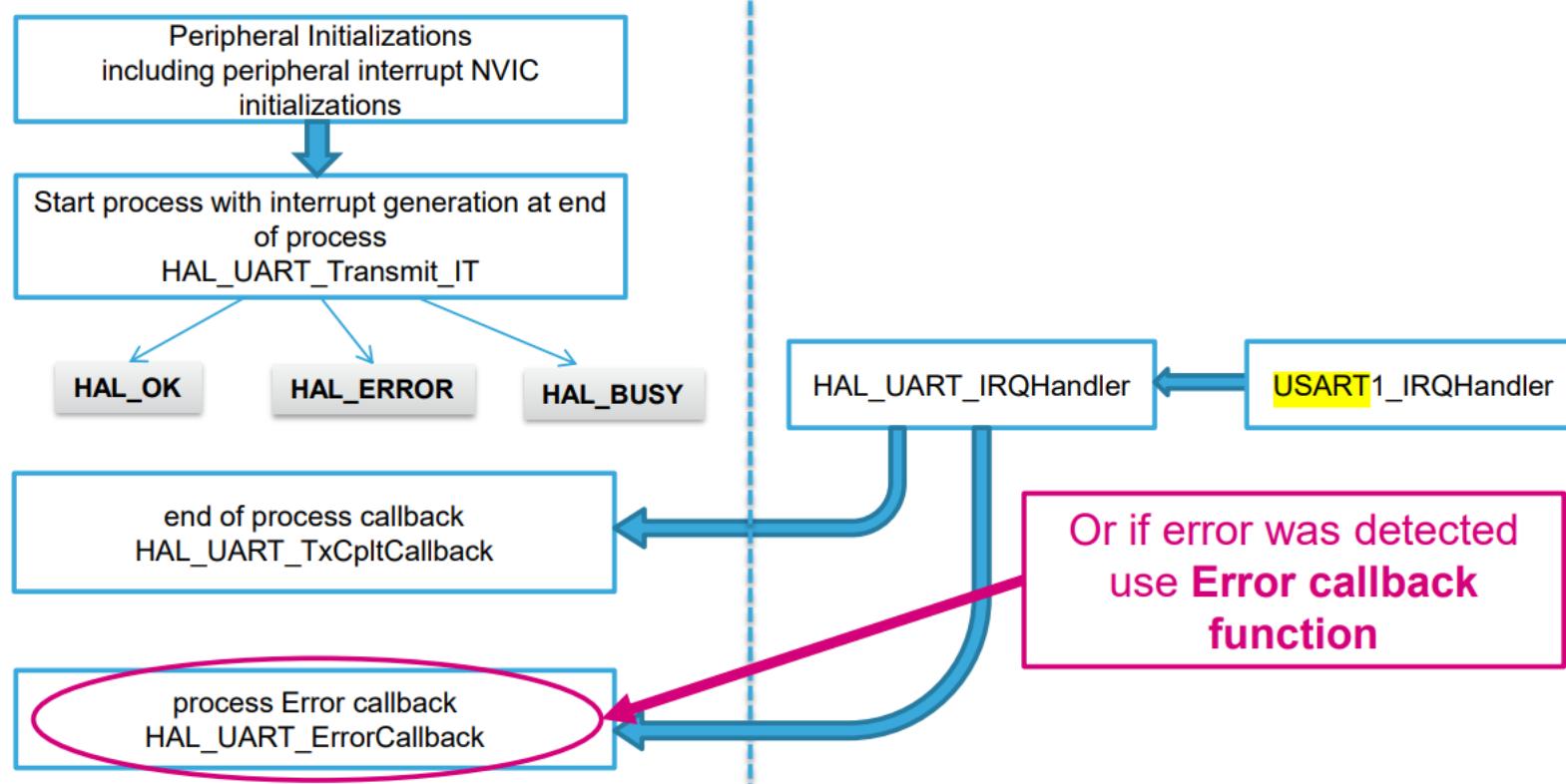
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



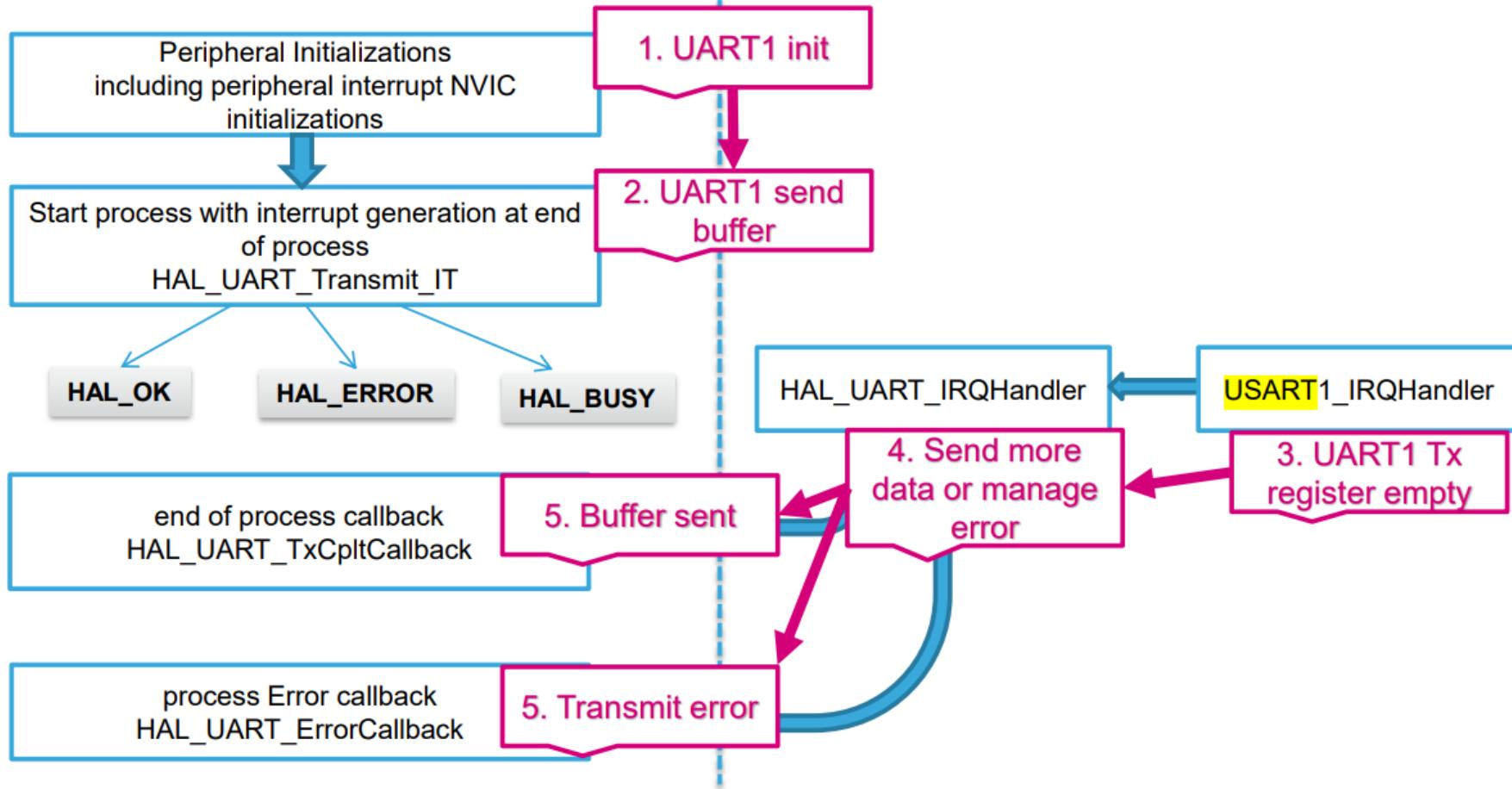
# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



# إنشاء اتصال عبر المنفذ التسلسلي باستخدام نمط الـ interrupt

## HAL Library UART with IT receive flow



# دوال مكتبة HAL المستخدمة ل التعامل مع منفذ الاتصال التسلسلي في وضع الـ Interrupt

سنستخدم دالتين رئيسيتين ل التعامل مع المنفذ التسلسلي إحداهما لإرسال والأخرى ل الاستقبال:

دالة الإرسال:

```
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 إضافة IT في اسم الدالة وأيضاً تم إزالة Timeout من بارامترات هذه الدالة مقارنة بالدالة المستخدمة في نمط الـ Polling، لأنه لم يعد هناك زمن انتظار في نمط المقاطعة.

# دوال مكتبة HAL المستخدمة للتعامل مع منفذ الاتصال التسلسلي في وضع الـ interrupt

□ دالة الاستقبال: إذا أردنا استقبال بيانات على UART باستخدام وصفات HAL ومكتبات interrupt يقوم باستدعاء الدالة التالية:

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

حيث:

□ 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

# التطبيق العملي 1: استخدام المنفذ التسلسلي UART2 في نمط الـ Polling لطباعة قيمة متحوال على النافذة التسلسليّة

نقوم بضبط إعدادات المنفذ التسلسلي:

SX \*uart.ioc

Pinout & Configuration      Clock Configuration      Project Manager      Tools

Software Packs

Pinout

Pinout view      System view

Categories A-Z

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- SYS
- WWDG

Analog

Timers

Connectivity

- I2C1
- I2C2
- IRTIM
- LPUART1
- SPI1
- SPI2
- UCPD1
- UCPD2
- USART1
- USART2**
- USART3
- USART4

Multimedia

Computing

Middleware

Utilities

USART2 Mode and Configuration

Mode

Hardware Flow Control (RS232) Disable

Hardware Flow Control (RS485)

Slave Select(NSS) Management Disable

Configuration

Reset Configuration

NVIC Settings   DMA Settings   GPIO Settings

Parameter Settings   User Constants

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

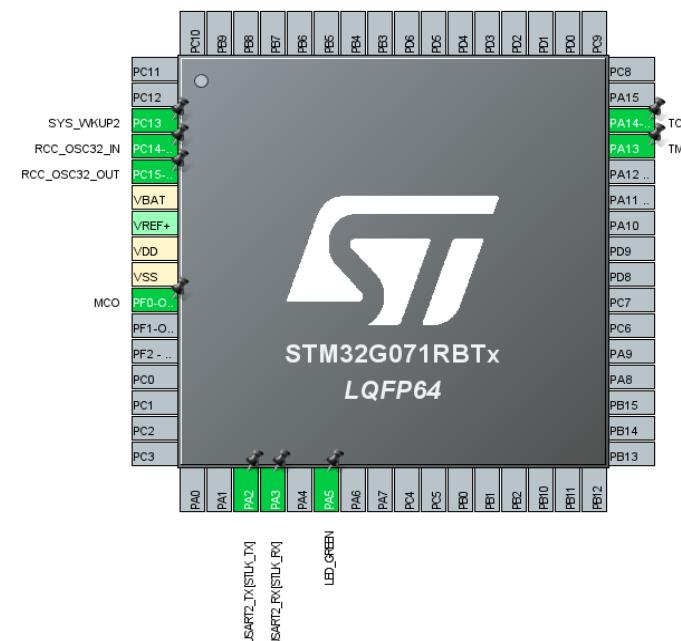
- Baud Rate: 115200
- Word Length: 8 Bits (including Parity)
- Parity: None
- Stop Bits: 1

Advanced Parameters

- Data Direction: Receive and Transmit
- Over Sampling: 16 Samples
- Single Sample: Disable
- ClockPrescaler: 1
- Fifo Mode: Disable
- Txfifo Threshold: 1 eighth full configuration
- Rx fifo Threshold: 1 eighth full configuration

Advanced Features

- Auto Baudrate: Disable
- TX Pin Active Level Inversion: Disable
- RX Pin Active Level Inversion: Disable
- Data Inversion: Disable
- TX and RX Pins Swapping: Disable
- Overrun: Enable
- DMA on RX Error: Enable
- MSB First: Disable



# التطبيق العملي 1 : استخدام المنفذ التسلسلي UART2 في نمط الـ Polling لطباعة قيمة متتحول على النافذة التسلسليّة

□ يصبح الكود بالشكل التالي:

```
#include "main.h"

UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
int main(void)
{
```

# التطبيق العملي 1: استخدام المنفذ التسلسلي UART2 في نمط الـ طباعة قيمة متتحول على النافذة التسلسليّة Polling

```
uint8_t MSG[35] = {'\0'};  
uint8_t X = 0;  
HAL_Init();  
SystemClock_Config();  
MX_GPIO_Init();  
MX_USART2_UART_Init();
```

# التطبيق العملي 1 : استخدام المنفذ التسلسلي UART2 في نمط الـ Polling لطباعة قيمة متتحول على النافذة التسلسليّة

```
while (1)
```

```
{
```

```
    sprintf(MSG, "Hello Dudes! Tracing X =  
%d\r\n", X);
```

```
    HAL_UART_Transmit(&huart2, MSG,  
sizeof(MSG), 100);
```

```
    HAL_Delay(500);
```

```
    X++;
```

```
}
```

**التطبيق العملي 2 : استخدام المنفذ التسلسلي UART2 في نمط الـ interrupt لطباعة قيمة متتحول على النافذة التسلسليّة**

**نقوم بضبط إعدادات المنفذ التسلسلي:**

# التطبيق العملي 2 : استخدام المنفذ التسلسلي UART2 في نمط الـ interrupt لطباعة قيمة متتحول على النافذة التسلسليّة

نقوم بضبط إعدادات مقاطعة المنفذ التسلسلي:

The screenshot shows the MX Configuration tool interface for a USART2 interrupt setup. The left sidebar lists various peripheral categories like DMA, GPIO, IWDG, NVIC, RCC, SYS, WWDG, Analog, Timers, Connectivity, I2C1, I2C2, IRTIM, LPUART1, SPI1, SPI2, UCPD1, UCPD2, USART1, USART2, USART3, USART4, Multimedia, Computing, and Middleware. The USART2 item is selected and highlighted with a blue bar. The main window has two tabs: 'Pinout & Configuration' and 'Clock Configuration'. The 'Pinout & Configuration' tab is active, displaying the 'USART2 Mode and Configuration' section under 'Mode'. It shows the mode set to 'Asynchronous', hardware flow control (RS232) disabled, and RS485 management disabled. Below this is the 'Configuration' section, which includes 'Reset Configuration' and tabs for 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings', and 'User Constants'. Under 'NVIC Interrupt Table', it shows 'USART2 global interrupt / USART2 wake-up interrupt thr...' with 'Enabled' checked and 'Preemption Priority' set to 0.

# التطبيق العملي 1 : استخدام المنفذ التسلسلي UART2 في نمط الـ interrupt لطباعة قيمة متتحول على النافذة التسلسليّة

□ يصبح الكود بالشكل التالي:

```
#include "main.h"

UART_HandleTypeDef huart2;
uint8_t rx_buffer[4];
uint8_t tx[]='hello';
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
```

# التطبيق العملي 1 : استخدام المنفذ التسلسلي UART2 في نمط الـ interrupt لطباعة قيمة متتحول على النافذة التسلسليّة

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    HAL_UART_Transmit_IT(&huart2, tx, 5);
    HAL_UART_Receive_IT(&huart2, rx_buffer, 4);
```

# التطبيق العملي1 : استخدام المنفذ التسلسلي UART2 في نمط الـ interrupt لطباعة قيمة متتحول على النافذة التسلسليّة

```
while (1)
```

```
{  
}
```

```
Void
```

```
HAL_UART_RxCpltCallback(UART_HandleTypeDef
```

```
*huart)
```

```
{
```

```
HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5) ;
```

```
HAL_UART_Receive_IT(&huart2, rx_buffer, 4);
```

```
HAL_UART_Transmit_IT(&huart2, rx_buffer, 4);
```

```
}
```

Thank you for listening