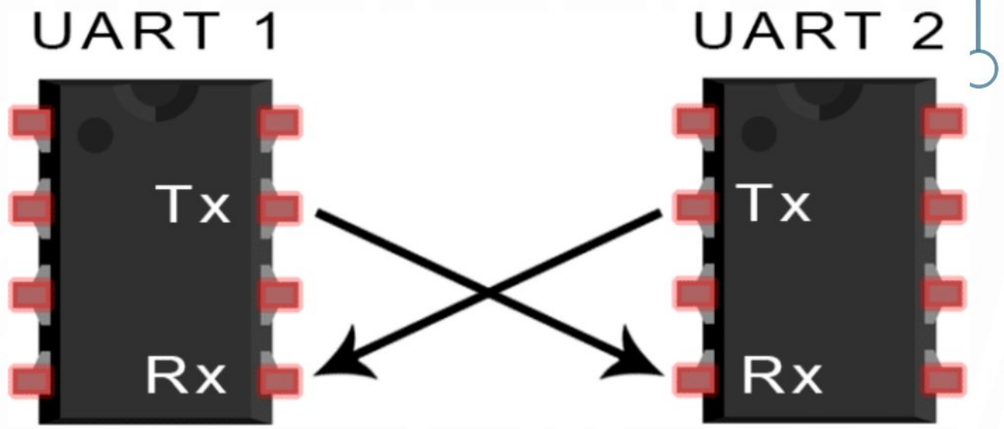
MICROS 32 BITS
STM - USART

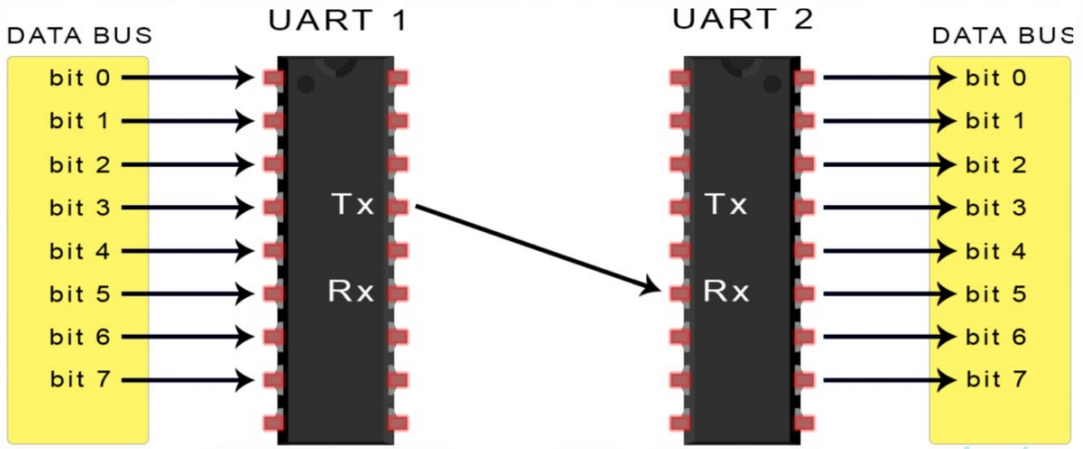
ROBINSON JIMENEZ MORENO

USART - STM32F4

Introducción:

UART (Universal Asynchronous Receiver-Transmitter) es un módulo físico instalado en la placa del microcontrolador que se encarga de controlar los puertos y dispositivos serie, cuyo objetivo principal es la transmisión y recepción de datos con un número reducido de líneas de comunicación. Sus funciones principales son: manejar las interrupciones de los dispositivos conectados al puerto serie y convertir datos en formato paralelo, a formato serie para que puedan ser transmitidos, recibirlos en serie y pasarlos a formato paralelo para que puedan ser procesados. El módulo UART usa solo dos cables para la transmisión de información, Tx y Rx.





https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter

http://www.circuitbasics.com/basics-uart-communication/

USART - STM32F4

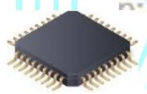
Bit de inicio: en estado de NO transmisión, el voltaje de Tx está en alto, por lo que, para iniciar la transmisión, se envía un voltaje bajo.

Paquete de datos: Contiene los datos reales a transmitir, con una longitud entre 5 y 8 bits. En la mayoría de los casos, el bit menos significativo se envía primero

UART

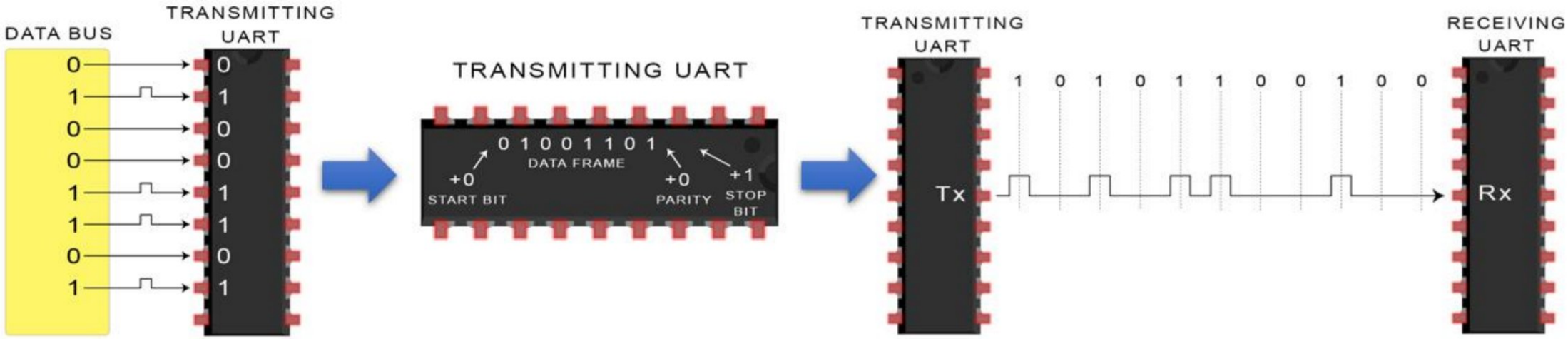
Paridad: Si el bit de paridad es 0 (paridad par), la suma de bits '1' de la trama de datos debe ser par. Si el bit de paridad es un 1 (paridad impar), la suma debe ser impar. En caso contrario, se considera que hubo un error en la transmisión

Bit de parada: Para señalar el final del paquete de datos, el UART de transmisión pasa un voltaje bajo a uno alto durante al menos dos duraciones de bit.



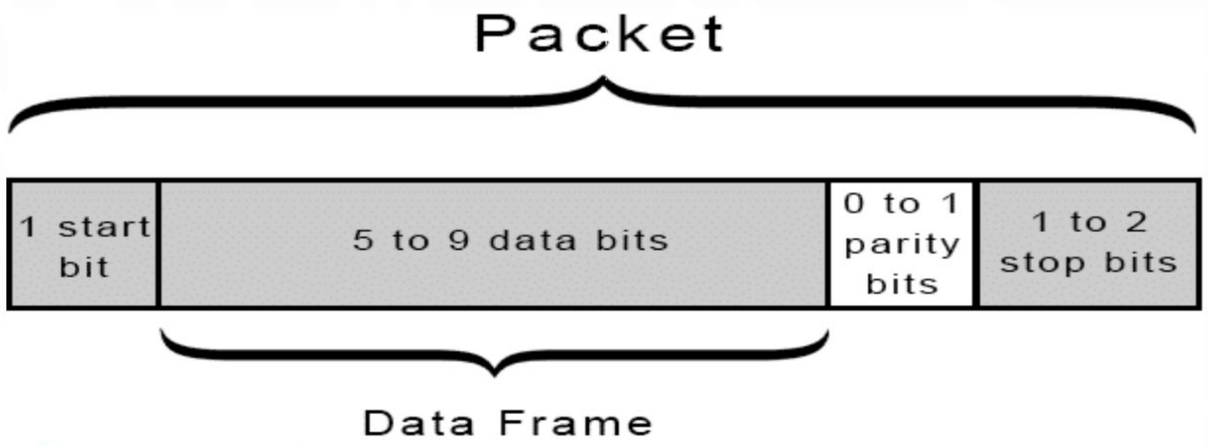
El módulo USART realiza una comunicación asíncrona, lo que significa que no requiere señal de reloj para la transmisión de la información. En lugar de ello, se agrega un bit de inicio y uno de parada al paquete de datos, los cuales indican el comienzo y el final del proceso de transferencia de datos. Cada bit se lee a una velocidad de transferencia determinada, llamada “Baud Rate”, expresada en bits por segundo (bps).

Ambos USART deben estar configurados para transmitir y recibir la misma estructura de paquetes de datos (trama) y con el mismo Baud Rate.



http://www.circuitbasics.com/basics-uart-communication/

Los datos transmitidos por USART se organizan en paquetes. Cada paquete contiene 1 bit de inicio, 5 a 9 bits de datos (según el USART), un bit de paridad (opcional) y 1 o 2 bits de parada.



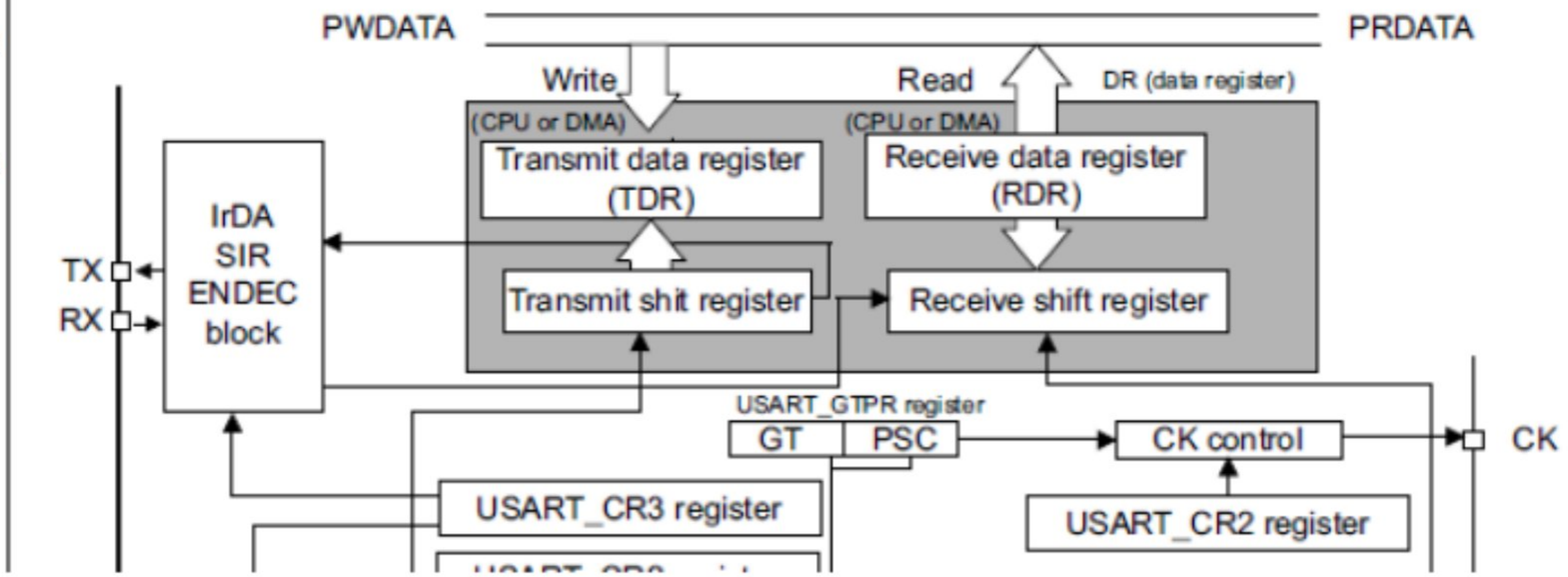
http://www.circuitbasics.com/basics-uart-communication/

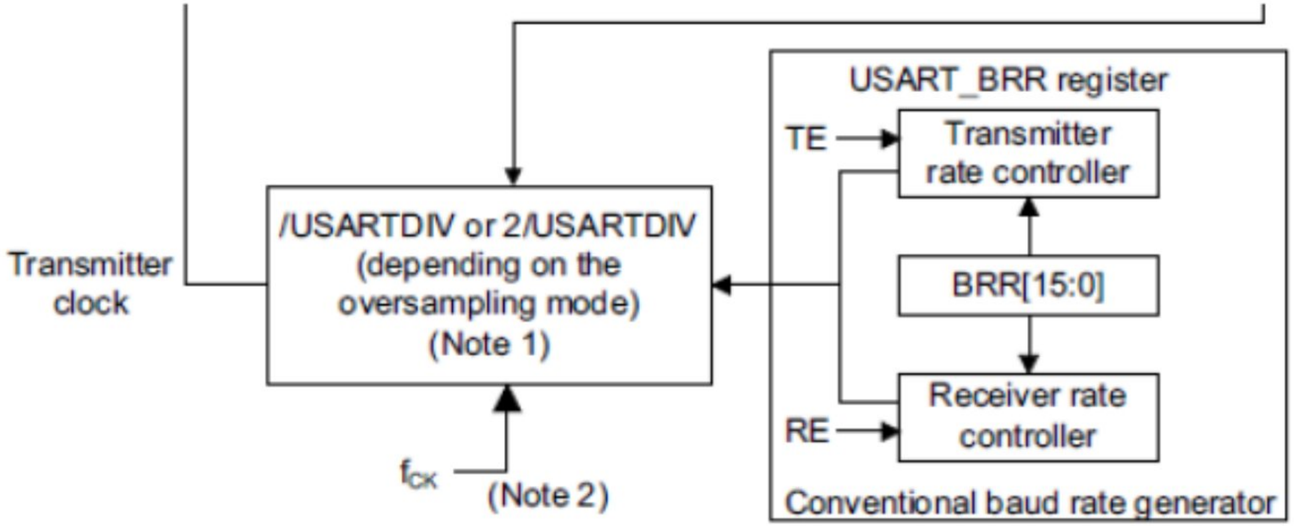
7 bits de datos	byte con bit de paridad	
	par	impar
0000000	00000000	00000001
1010001	10100011	10100010
1101001	11010010	11010011
1111111	11111111	11111110

https://sites.google.com/site/hardwarejoseangel/memoria-ram

USART - STM32F4

Figure 316. USART block diagram





La STM32F4x maneja 4 puertos: UART4, UART5, UART7 y UART8.

REGISTROS:

USARTx -> CR1
BRR
TDR
RDR

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Refer to *Section 19.6: USART registers on page 548* for the definitions of each bit.

The following pin is required to interface in synchronous mode:

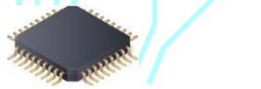
- **CK:** Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission corresponding to SPI master mode (no clock pulses on start bit and stop bit, and a software option to send a clock pulse on the last data bit). In parallel data can be received synchronously on RX. This can be used to control peripherals that have shift registers (e.g. LCD drivers). The clock phase and polarity are software programmable. In smartcard mode, CK can provide the clock to the smartcard.

The following pins are required in Hardware flow control mode:

- **CTS:** Clear To Send blocks the data transmission at the end of the current transfer when high
- **RTS:** Request to send indicates that the USART is ready to receive a data (when low).

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the USART_BRR register.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.



```
RCC->APB1ENR |= (1UL<<17);
```

19.6.4 Control register 1 (USART_CR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
USART2->CR1 = 0x200C;
```

19.6.5 Control register 2 (USART_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

19.6.3 Baud rate register (USART_BRR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
USART2->BRR = 0x683;
```

STM32F4

Pin name (function after reset)[1]	Pin type	I/O structure	Notes	Alternate functions
PD5	I/O	FT	-	USART2_TX, EVENTOUT
PD6	I/O	FT	-	SPI3_MOSI/I2S3_SD, USART2_RX, EVENTOUT

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

```
GPIOD->MODER |=  0x2801;
```

Port	AF00 SYS_AF	AF01 TIM1/TIM2	AF02 TIM3/ TIM4/ TIM5	AF03 TIM9/ TIM10/ TIM11	AF04 I2C1/I2C2/ I2C3	AF05 SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	AF06 SPI2/I2S2/ SPI3/ I2S3	AF07 SPI3/I2S3/ USART1/ USART2
PD5	-	-	-	-	-	-	-	USART2_ TX
PD6	-	-	-	-	-	SPI3_MOSI /I2S3_SD	-	USART2_ RX

8.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..E and H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	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
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
GPIOD->AFR[0] |= 0x007700000;
```

```
//CONFIGURACION UART
RCC->APB1ENR |= 0x80000; // Enable clock for UART4 (set el pin 19 )
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UART8 EN	UART7 EN	DAC EN	PWR EN	CEC EN	CAN2 EN	CAN1 EN	I2C4 EN	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	SPDIFRX EN
rw	rw	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
SPI3 EN	SPI2 EN	Res.	Res.	WWDG EN	Res.	LPTIM1 EN	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 19 **UART4EN:** UART4 clock enable

This bit is set and cleared by software.

0: UART4 clock disabled

1: UART4 clock enabled

```
RCC->APB1ENR |= (1UL<<17);
```

```
UART4->CR1 |= 0x2C;          // Tx habilitado, Rx habilitado, Interrup de Rx habilitado, 8N1
                             // CR2 se deja por defecto pues 1 bit de stop es 00
```

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	Res.	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 28 **M1**: Word length

This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.

M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits

M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits

M[1:0] = 10: 1 Start bit, 7 data bits, n stop bits

This bit can only be written when the USART is disabled (UE=0).

INGENIERÍA MECATRÓNICA UMNG

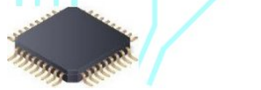
```
UART4->CR1 |= 0x2C;      // Tx habilitado, Rx habilitado, Interrup de Rx habilitado, 8N1
                         // CR2 se deja por defecto pues 1 bit de stop es 00
```

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	DLEIE	TE	RE	Res.	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 5 **RXNEIE**: RXNE interrupt enable
This bit is set and cleared by software.
0: Interrupt is inhibited
1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_ISR register

Bit 3 **TE**: Transmitter enable
This bit enables the transmitter. It is set and cleared by software.
0: Transmitter is disabled
1: Transmitter is enabled

Bit 2 **RE**: Receiver enable
This bit enables the receiver. It is set and cleared by software.
0: Receiver is disabled
1: Receiver is enabled and begins searching for a start bit



UNIVERSIDAD MILITAR NUEVA GRANADA

```
UART4->BRR = 0x683;      // 9600 Baudios, fclk=16Mhz, por defecto el HSI
```

31.8.4 Baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000

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

$$USARTDIV = FCLK_{UART}/vel\_deseada$$

$$USARTDIV = 16Mhz/9600bps$$

$$USARTDIV = 1666{,}66 \approx 1667$$

$$BRR = USARTDIV \approx 1667 = 0x683$$

19.3.4 Fractional baud rate generation

The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV.

Equation 1: Baud rate for standard USART (SPI mode included)

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

Equation 2: Baud rate in Smartcard, LIN and IrDA modes

$$\text{Tx/Rx baud} = \frac{f_{CK}}{16 \times \text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

- When OVER8=0, the fractional part is coded on 4 bits and programmed by the DIV_fraction[3:0] bits in the USART_BRR register
- When OVER8=1, the fractional part is coded on 3 bits and programmed by the DIV_fraction[2:0] bits in the USART_BRR register, and bit DIV_fraction[3] must be kept cleared.

19.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

Table 74. Error calculation for programmed baud rates at f_{PCLK} = 8 MHz or f_{PCLK} = 12 MHz, oversampling by 16(1)

Oversampling by 16 (OVER8=0)							
Baud rate7			f_{PCLK} = 8 MHz		f_{PCLK} = 12 MHz		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired) B.rate / Desired B.rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	416.6875	0	1.2 KBps	625	0
2	2.4 KBps	2.4 KBps	208.3125	0.01	2.4 KBps	312.5	0
3	9.6 KBps	9.604 KBps	52.0625	0.04	9.6 KBps	78.125	0
4	19.2 KBps	19.185 KBps	26.0625	0.08	19.2 KBps	39.0625	0
5	38.4 KBps	38.462 KBps	13	0.16	38.339 KBps	19.5625	0.16
6	57.6 KBps	57.554 KBps	8.6875	0.08	57.692 KBps	13	0.16
7	115.2 KBps	115.942 KBps	4.3125	0.64	115.385 KBps	6.5	0.16
8	230.4 KBps	228.571 KBps	2.1875	0.79	230.769 KBps	3.25	0.16
9	460.8 KBps	470.588 KBps	1.0625	2.12	461.538 KBps	1.625	0.16
10	921.6 KBps	NA	NA	NA	NA	NA	NA
11	2 MBps	NA	NA	NA	NA	NA	NA
12	3 MBps	NA	NA	NA	NA	NA	NA

1. The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

```
UART4->CR1 |= 0x1;          // Habilita UART, UE=1

NVIC_EnableIRQ(UART4_IRQn);      // Habilita la interrupción del UART4
```

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	Res.	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bit 0 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USART_ISR are set to their default values. This bit is set and cleared by software.

0: USART prescaler and outputs disabled, low-power mode

1: USART enabled

31.8.2 Control register 2 (USART_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

STM32F7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD[7:4]				ADD[3:0]				RTOEN	ABRMOD[1:0]		ABREN	MSBFIRST	DATAINV	TXINV	RXINV
rw	rw	rw	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
SWAP	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw				

Bit 17 **TXINV:** TX pin active level inversion
This bit is set and cleared by software.
0: TX pin signal works using the standard logic levels (V_{DD} =1/idle, Gnd=0/mark)
1: TX pin signal values are inverted. (V_{DD} =0/mark, Gnd=1/idle).
This allows the use of an external inverter on the TX line.
This bit field can only be written when the USART is disabled (UE=0).

Bit 16 **RXINV:** RX pin active level inversion
This bit is set and cleared by software.
0: RX pin signal works using the standard logic levels (V_{DD} =1/idle, Gnd=0/mark)
1: RX pin signal values are inverted. (V_{DD} =0/mark, Gnd=1/idle).
This allows the use of an external inverter on the RX line.
This bit field can only be written when the USART is disabled (UE=0).

Bit 15 **SWAP:** Swap TX/RX pins
This bit is set and cleared by software.
0: TX/RX pins are used as defined in standard pinout
1: The TX and RX pins functions are swapped. This allows to work in the case of a cross-wired connection to another USART.
This bit field can only be written when the USART is disabled (UE=0).

Bit 14 **LINEN**: LIN mode enable
This bit is set and cleared by software.
0: LIN mode disabled
1: LIN mode enabled
The LIN mode enables the capability to send LIN synchronous breaks (13 low bits) using the SBKRQ bit in the USART_RQR register, and to detect LIN Sync breaks.
This bit field can only be written when the USART is disabled (UE=0).
Note: If the USART does not support LIN mode, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1023.

Bits 13:12 **STOP[1:0]**: STOP bits
These bits are used for programming the stop bits.
00: 1 stop bit
01: 0.5 stop bit
10: 2 stop bits
11: 1.5 stop bits
This bit field can only be written when the USART is disabled (UE=0).

Bit 11 **CLKEN**: Clock enable
This bit allows the user to enable the CK pin.
0: CK pin disabled
1: CK pin enabled
This bit can only be written when the USART is disabled (UE=0).
Note: If neither synchronous mode nor Smartcard mode is supported, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1023.
In order to provide correctly the CK clock to the Smartcard when CK is always available When CLKEN = 1, regardless of the UE bit value, the steps below must be respected:
- UE = 0
- SCEN = 1
- GTPR configuration (If PSC needs to be configured, it is recommended to configure PSC and GT in a single access to USART_ GTPR register).
- CLKEN= 1
- UE = 1

Bit 10 **CPOL**: Clock polarity
This bit allows the user to select the polarity of the clock output on the CK pin in synchronous mode. It works in conjunction with the CPHA bit to produce the desired clock/data relationship
0: Steady low value on CK pin outside transmission window
1: Steady high value on CK pin outside transmission window
This bit can only be written when the USART is disabled (UE=0).
Note: If synchronous mode is not supported, this bit is reserved and must be kept at reset value. Please refer to Section 31.4: USART implementation on page 1023.

31.8.10 Receive data register (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

Bits 8:0 **RDR[8:0]**: Receive data value

Contains the received data character.

The RDR register provides the parallel interface between the input shift register and the internal bus (see *Figure 334*).

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

31.8.11 Transmit data register (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

Bits 8:0 **TDR[8:0]**: Transmit data value

Contains the data character to be transmitted.

The TDR register provides the parallel interface between the internal bus and the output shift register (see *Figure 334*).

El UART requiere configuración de modo alterno en el GPIOx y configurar el registro alterno correspondiente AFR[x].

```
GPIOA->MODER |=   (2UL << 2*0);    //pines PA0 en modo alterno (TX)
GPIOC->MODER |=   (2UL << 2*11);   //pines PC11 en modo alterno (RX)
GPIOA->AFR[0]  |= 0x8;             // PA0 -> AF8=UART4 TX, AF8=1000 en AFR0
GPIOC->AFR[1]  |= 0x8000;          // PC11 -> AF8=UART4 RX, AF8=1000 en AFR11
```

Table 12. STM32F745xx and STM32F746xx alternate function mapping

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	DCMI	LCD	SYS
PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	SAI2_SD_B	ETH_MII_CRS	-	-	-	EVENTOUT
												ETH_MII				
PC11	-	-	-	-	-	-	SPI3_MISO	USART3_RX	UART4_RX	QUADSPI_BK2_NCS	-	-	SDMMC1_D3	DCMI_D4	-	EVENTOUT

6.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	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
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFRy[3:0]:** Alternate function selection for port x pin y (y = 0..7)
These bits are written by software to configure alternate function I/Os

AFSELy selection:

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: AF8
1001: AF9
1010: AF10
1011: AF11
1100: AF12
1101: AF13
1110: AF14
1111: AF15

6.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]			
rw	rw	rw	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
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

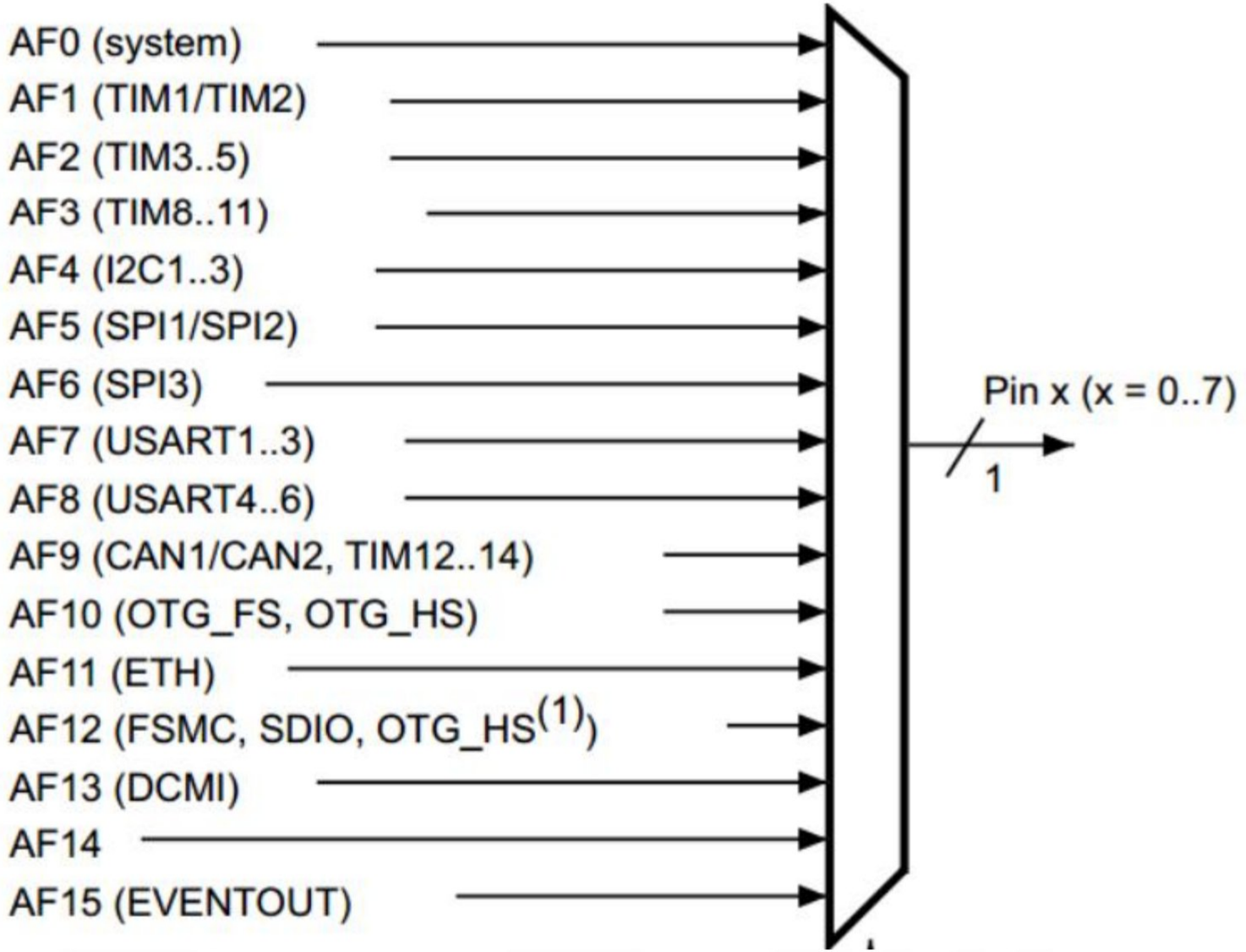
Bits 31:0 **AFRy[3:0]:** Alternate function selection for port x pin y (y = 8..15)

These bits are written by software to configure alternate function I/Os

AFSELy selection:

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: AF8
1001: AF9
1010: AF10
1011: AF11
1100: AF12
1101: AF13
1110: AF14
1111: AF15

Cada GPIO tiene multiplexadas hasta 16 funciones alternativas como se muestra a continuación.



31.8.8 Interrupt and status register (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.	Res.	Res.	Res.	Res.	TEACK	Res.	RWU	SBKF	CMF	BUSY
										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

```
if (UART4->ISR & 0x20)

while ((UART4->ISR &= 0x80)==0);
```

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the USART_TDR register has been transferred into the shift register. It is cleared by a write to the USART_TDR register.
The TXE flag can also be cleared by writing 1 to the TXFRQ in the USART_RQR register, in order to discard the data (only in Smartcard T=0 mode, in case of transmission failure).
An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register.
0: data is not transferred to the shift register
1: data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by software, writing 1 to the TCCF in the USART_ICR register or by a write to the USART_TDR register.
An interrupt is generated if TCIE=1 in the USART_CR1 register.
0: Transmission is not complete
1: Transmission is complete

Note: If TE bit is reset and no transmission is on going, the TC bit will be set immediately.

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_RDR register. It is cleared by a read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register.
An interrupt is generated if RXNEIE=1 in the USART_CR1 register.
0: data is not received
1: Received data is ready to be read.

➢ Métodos: Método que envía un solo carácter ASCII

```
void send_dato(char dt){
  UART4->TDR = dt;
  while ((UART4->ISR &= 0x80)==0);    //esperar hasta que TXE sea 1, lo que indica que se envio el dato
}
```

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the USART_TDR register has been transferred into the shift register. It is cleared by a write to the USART_TDR register.

The TXE flag can also be cleared by writing 1 to the TXFRQ in the USART_RQR register, in order to discard the data (only in Smartcard T=0 mode, in case of transmission failure).

An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register.

0: data is not transferred to the shift register

1: data is transferred to the shift register)

Note: This bit is used during single buffer transmission.

31.8.8 Interrupt and status register (USART_ISR)

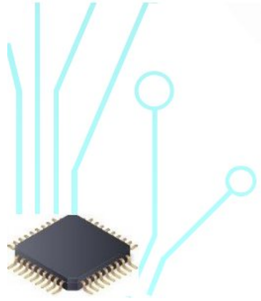
Address offset: 0x1C

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.	Res.	Res.	Res.	Res.	TEACK	Res.	RWU	SBKF	CMF	BUSY
										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

- **Métodos:** Método que envía una frase o un conjunto de caracteres en cadena.

```
char envio1[] = {"MENSAJE DE PRUEBA\n\r-"};

void send(){
  short i=0;
  for (i=0;envio!='-';i++){
      envio = envio1[i];
      UART4->TDR = envio;
      while ((UART4->ISR &= 0x80)==0);    //esperar hasta que TXE sea 1, lo que indica que se envio el dato
  }
  envio='1';
}
```



➢ ESTRUCTURA GENERAL DE LA INTERRUPCIÓN:

```
extern "C" {

  void UART4_IRQHandler(void){        //Interrupcion por dato recibido.
    if (UART4->ISR & 0x20) {          // evaluar si la bandera RXNE esta activa
        dato = UART4->RDR;
    }
  }

}
```

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_RDR register. It is cleared by a read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register.
An interrupt is generated if RXNEIE=1 in the USART_CR1 register.
0: data is not received
1: Received data is ready to be read.

31.8.8 Interrupt and status register (USART_ISR)

Address offset: 0x1C

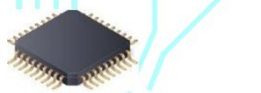
Reset value: 0x0200 00C0

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

Table 12. STM32F745xx and STM32F746xx alternate function mapping (continued)

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	DCMI	LCD	SYS
Port C	PC4	-	-	-	-	-	I2S1_MCK	-	-	SPDIFRX_IN2	-	-	ETH_MII_RXD0/ETH_RMII_RXD0	FMC_SDNE0	-	-	EVENTOUT
	PC5	-	-	-	-	-	-	-	-	SPDIFRX_IN3	-	-	ETH_MII_RXD1/ETH_RMII_RXD1	FMC_SDCKE0	-	-	EVENTOUT
	PC6	-	-	TIM3_CH1	TIM8_CH1	-	I2S2_MCK	-	-	USART6_TX	-	-	-	SDMMC1_D6	DCMI_D0	LCD_HSYNC	EVENTOUT
	PC7	-	-	TIM3_CH2	TIM8_CH2	-	-	I2S3_MCK	-	USART6_RX	-	-	-	SDMMC1_D7	DCMI_D1	LCD_G6	EVENTOUT
	PC8	TRACED1	-	TIM3_CH3	TIM8_CH3	-	-	-	UART5_RTS	USART6_CK	-	-	-	SDMMC1_D0	DCMI_D2	-	EVENTOUT
	PC9	MCO2	-	TIM3_CH4	TIM8_CH4	I2C3_SDA	I2S_CKIN	-	UART5_CTS	-	QUADSPI_BK1_IO0	-	-	SDMMC1_D1	DCMI_D3	-	EVENTOUT
	PC10	-	-	-	-	-	-	SPI3_SCK/I2S3_CK	USART3_TX	UART4_TX	QUADSPI_BK1_IO1	-	-	SDMMC1_D2	DCMI_D8	LCD_R2	EVENTOUT
	PC11	-	-	-	-	-	-	SPI3_MISO	USART3_RX	UART4_RX	QUADSPI_BK2_NCS	-	-	SDMMC1_D3	DCMI_D4	-	EVENTOUT

```
#include "stm32f7xx.h"

int i,Rx;
char datos[5]={120,15,36,3};
float valor;

int main(void){
  RCC -> AHB1ENR |= 0X6; //PUERTOS B Y C
  RCC -> APB1ENR |= 0X80000; //HABILITAR EL UART4

  GPIOB -> MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS

  GPIOC -> MODER = 0XA00000; //COLOCAR LOS PINES EN MODO ALTERNANTE PARA USAR EL UART
  GPIOC -> AFR[1] = 0X8800; //DEFINIR LA FUNCION ALTERNANTE PARA EL MODULO UART PC10 / PC11
  UART4 -> BRR = 0X683; //VELOCIDAD DE 9600 BAUDIOS
  UART4 -> CR1 = 0X2D; //HABILITAR EL UART, HABILITAR EL TRANSMISOR, EL RECEPTOR Y LA INTERRUPCION POR RECEPCION

  NVIC_EnableIRQ(UART4_IRQn); //HABILITAR LA INTERRUPCION DEL UART4
  GPIOB -> ODR=1;
  while(1){
    if ((GPIOC -> IDR &= 0x2000)==0x2000){GPIOB -> ODR=0xfff;
    for (i=0;i<5;i++){UART4 -> TDR =datos[i];
    while((UART4 -> ISR &=0x80)==0);
    }
  }
  GPIOB -> ODR=1;}
}
```

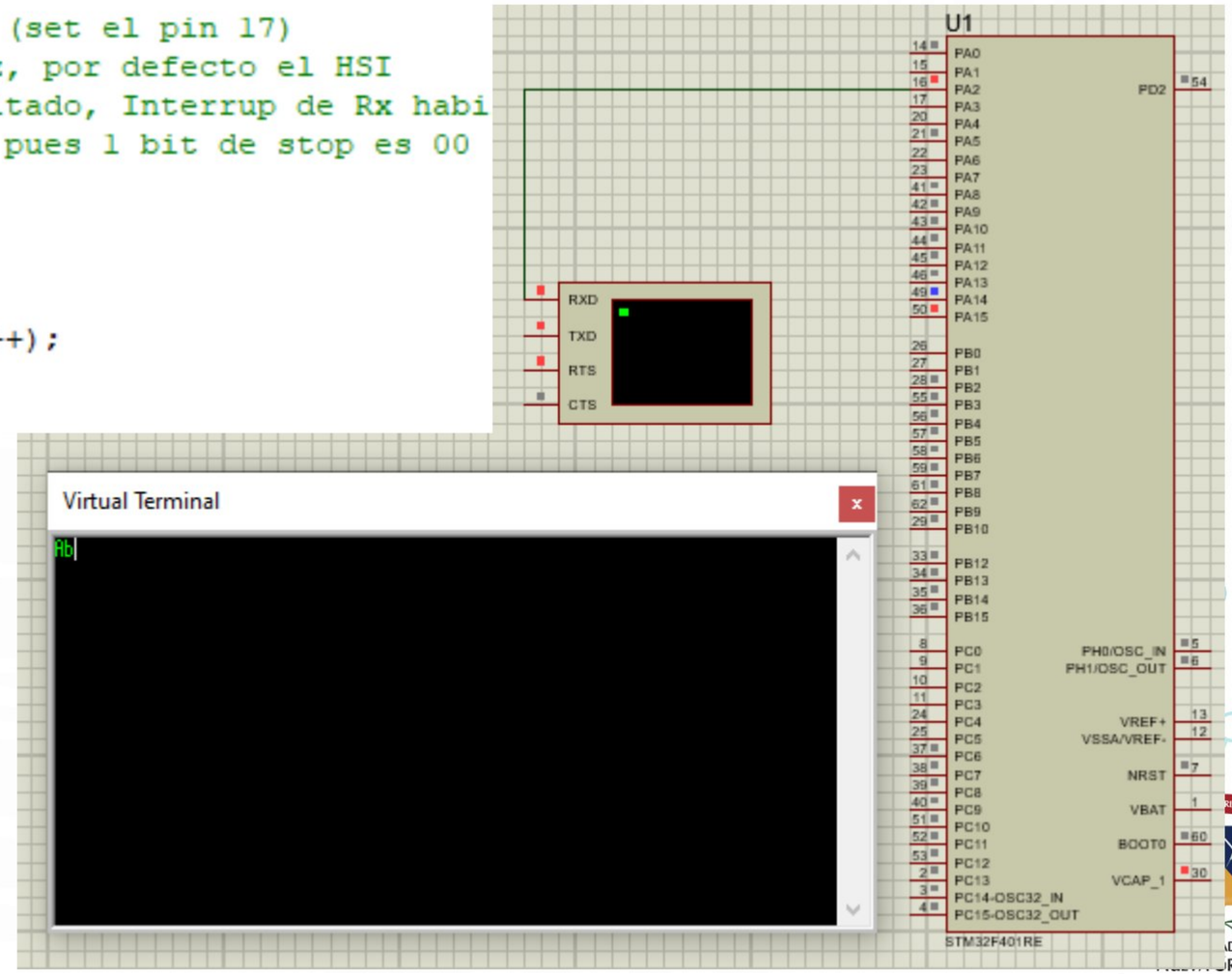


STM32F4

Table 9. Alternate function mapping

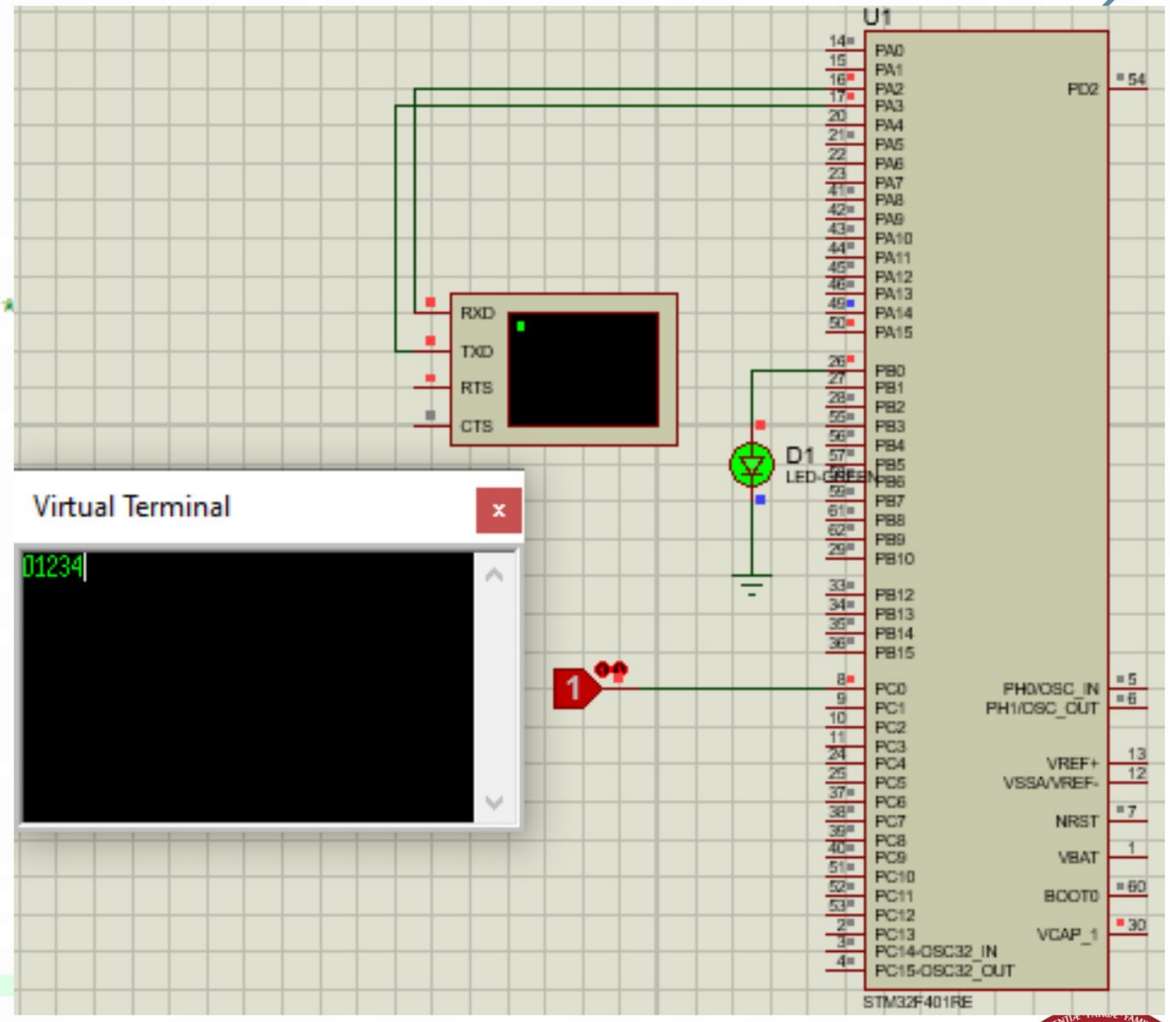
Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF1
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	SPI2/I2S2/ SPI3/ I2S3	SPI3/I2S3/ USART1/ USART2	USART6	I2C2/ I2C3	OTG1_FS	
Port A	PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	-	-	-	-
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_ TX	-	-	-	-
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-		-	USART2_ RX	-	-	-	-
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	USART2_ CK	-	-	-	-
	PA5	-	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_SCK	-	-	-	-	-	-
	PA6	-	TIM1_BKIN	TIM3_CH1	-	-	SPI1_ MISO	-	-	-	-	-	-
	PA7	-	TIM1_CH1N	TIM3_CH2	-	-	SPI1_ MOSI	-	-	-	-	-	-
	PA8	MCO_1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-
	PA9	-	TIM1_CH2	-	-	I2C3_ SMBA	-	-	USART1_ TX	-	-	OTG_FS_ VBUS	-

```
#include "STM32F4xx.h"

char envio[4] = {'A','b'};

int main(void){
  RCC->AHB1ENR =0x7; //Puertos A,B,C
  GPIOB->MODER |=  1;
//*********************************************************************************
  //CONFIGURACION UART
  RCC->APB1ENR |= (1UL<<17); // Enable clock for USART2 (set el pin 17)
  USART2->BRR = 0x683;        // 9600 Baudios, fclk=16Mhz, por defecto el HSI
  USART2->CR1 = 0x012C;       // Tx habilitado, Rx habilitado, Interrup de Rx habi
  USART2->CR1 |= 0x2000;      // CR2 se deja por defecto pues 1 bit de stop es 00

  GPIOA->MODER |=  0x20;     //pin PA2  alterno
  GPIOA->AFR[0] |= 0x700;    // PA2 -> AF7=USART2 TX,

    USART2->DR = envio[0];   for (long iX=0;iX<400000;iX++);
    USART2->DR = envio[1];
}
```



INGENIERÍA MECATRÓNICA UMNG

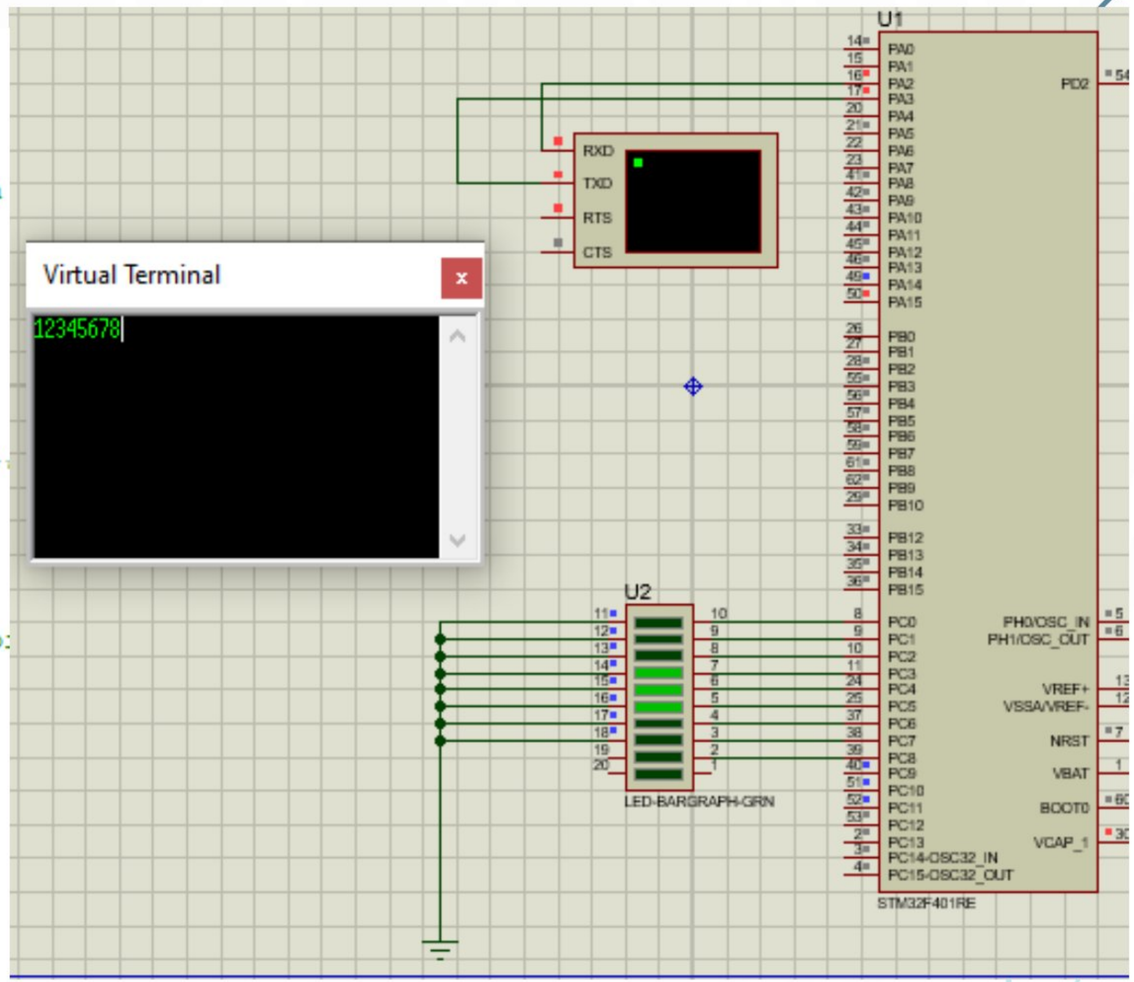
```
#include "STM32F4xx.h"
int d=0;
int main(void){
  RCC->AHB1ENR =7; //Puertos A,B,C,D
  GPIOB->MODER |=  1; //LED PRUEBA

//*************************************************************
  //CONFIGURACION UART
  RCC->APB1ENR |= (1UL<<17);

  USART2->BRR = 0x683;
  USART2->CR1 = 0x200C;

  //CONFIGURACION AF PTO_A
  GPIOA->MODER |=  0xA0;     //pin PA2-3  alterno
  GPIOA->AFR[0] |= 0x7700;    // PA2 -> AF7=USART2 TX,

  while(true){GPIOB->ODR =0;
  if((GPIOC->IDR & 1)==1){GPIOB->ODR =1;
  USART2->DR = d+0x30;  d++; }
  while ((GPIOC->IDR & 1)==1){__ASM("NOP");}
}}
```

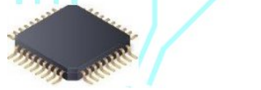


STM32F4

INGENIERÍA MECATRÓNICA UMNG

```
#include "STM32F4xx.h"

extern "C" {
  void USART2_IRQHandler(void){       //Interrupcion por dato recibido.
    if (USART2->SR & 0x20) {         // evaluar si la bandera RXNE esta activa
        GPIOC->ODR  = USART2->DR;    }
  }
}
int main(void){
  RCC->AHB1ENR =7; //Puertos A,B,C,D
  GPIOC->MODER |=  0x555555; //LED PRUEBA

//*********************************************************************************
  //CONFIGURACION UART
  RCC->APB1ENR |= (1UL<<17);

  USART2->BRR = 0x683;
  USART2->CR1 = 0x012C;   // Tx habilitado, Rx habilitado, Interrup de Rx hab
  USART2->CR1 |= 0x2000;
   NVIC_EnableIRQ(USART2_IRQn);
  //CONFIGURACION AF PTO_A
  GPIOA->MODER |=  0xA0;    //pin PA2-3  alterno
  GPIOA->AFR[0] |= 0x7700;   // PA2 -> AF7=USART2 TX,
  GPIOB->ODR =0;
  while(true){

}}
```



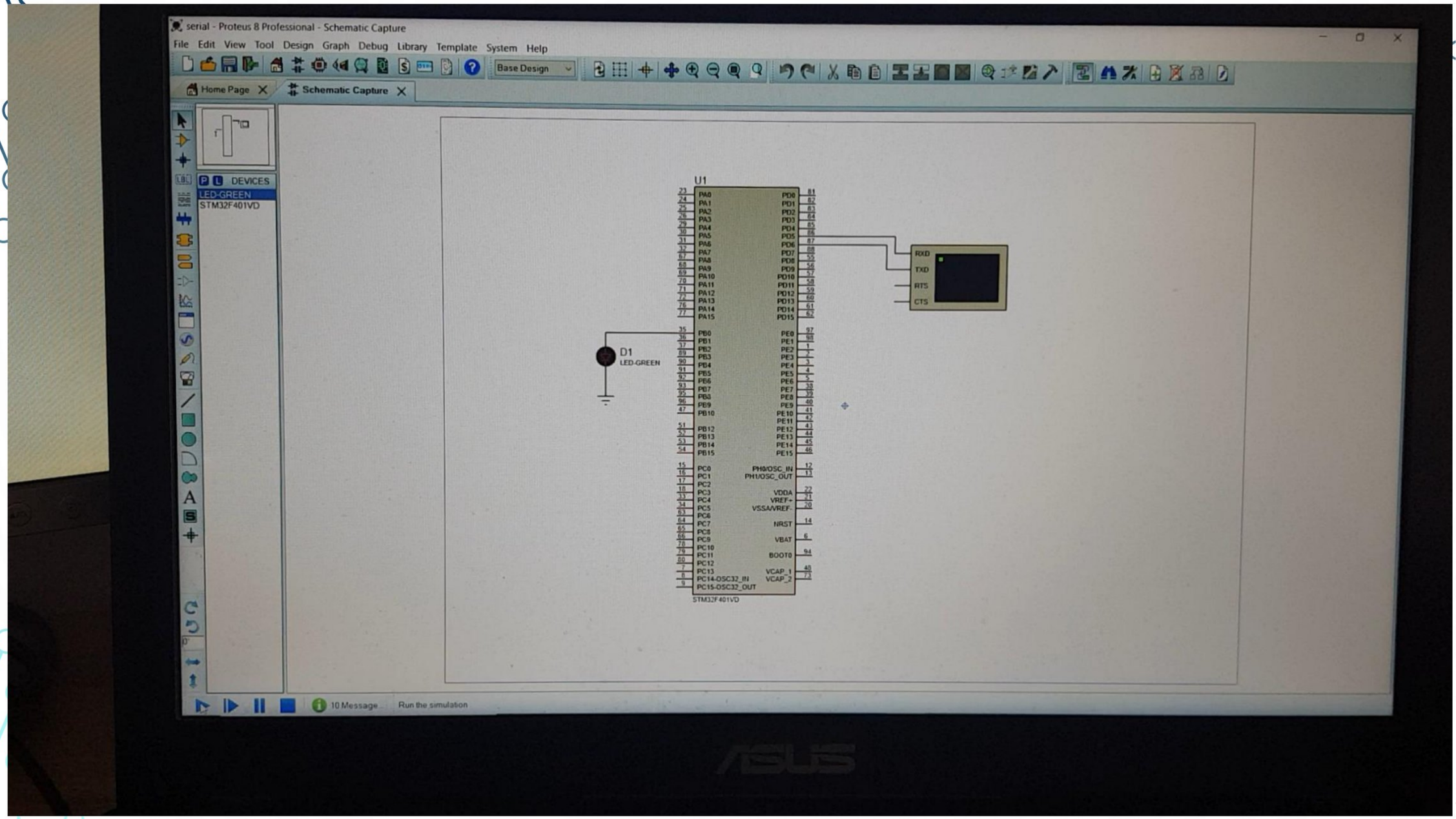


UNIVERSIDAD MILITAR NUEVA GRANADA

STM32F4

Serial STM32F4.cpp*

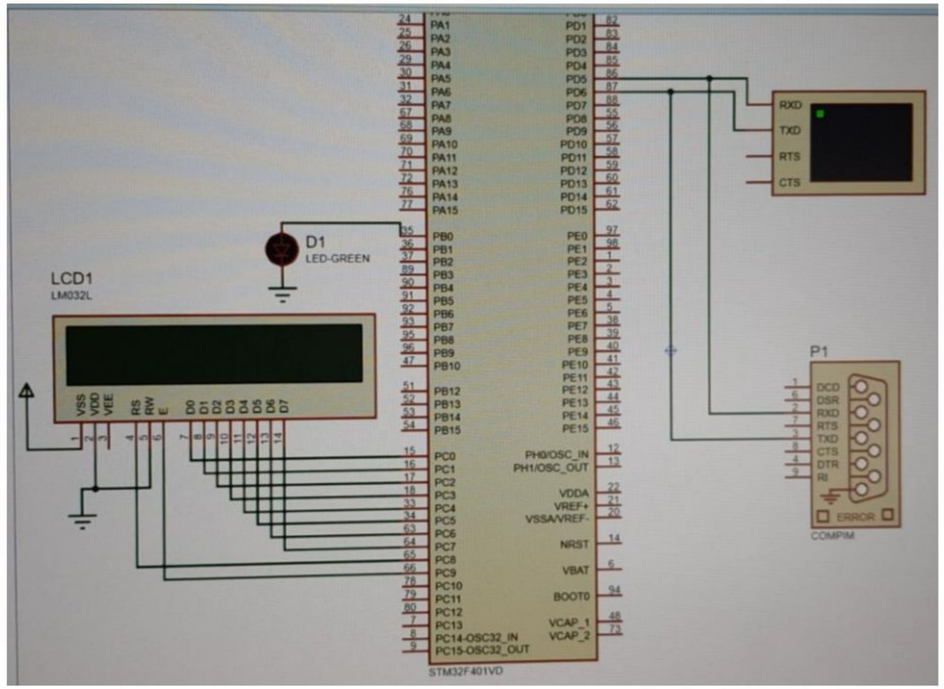
```
#include "STM32F4xx.h"
char dato;
char enviol[] = {"Conteo= \n\r-"};

extern "C" {
  void USART2_IRQHandler(void){      //Interrupcion por dato recibido.
    if (USART2->SR & 0x20) {        // evaluar si la bandera RXNE esta activa
        dato = USART2->DR;  }
  }
}

void send(){
    for ( int d=0;d<10;d++){
    for ( int u=0;u<10;u++){
    for ( int i=0;i<7;i++){
      USART2->DR = enviol[i]; while ((USART2->SR &= 0x80)==0); }
      USART2->DR = d+0x30;  while ((USART2->SR &= 0x80)==0);
      USART2->DR = u+0x30;  while ((USART2->SR &= 0x80)==0);
      USART2->DR = 0x0d;     while ((USART2->SR &= 0x80)==0);
      for(int x=0;x<500000;x++);dato=0;
  } }
    dato=0;
}

int main(void){
  RCC->AHB1ENR =0xFF; //Puertos A,B,C,D,E,F,G,H
  GPIOB->MODER |=  1;
//*********************************************************************************************
  //CONFIGURACION UART
  RCC->APB1ENR |= (1UL<<17); // Enable clock for USART2 (set el pin 17)
                            // 8N1-> M=00, no es necesario escribir el registro
  USART2->BRR = 0x683;      // 9600 Baudios, fclk=16Mhz, por defecto el HSI
  USART2->CR1 = 0x012C;      // Tx habilitado, Rx habilitado, Interrup de Rx habilitado, 8N1
  USART2->CR1 |= 0x2000;                              // CR2 se deja por defecto pues 1 bit de stop es 00

  NVIC_EnableIRQ(USART2_IRQn);      // Llama interrupción del USART2

  GPIOD->MODER |=  0x2800;  //pines PD5  PD6 en modo alterno (RX)
  GPIOD->AFR[0] |= 0x007700000;            // PD5 -> AF7=USART2 TX,
                                          // PD6 -> AF7=USART2 RX,

  while(true){
  if(dato=='s'){GPIOB->ODR=1;send();}
  else GPIOB->ODR=0;
}}
```



INGENIERÍA MECATRÓNICA UMNG

UNIVERSIDAD MILITAR NUEVA GRANADA

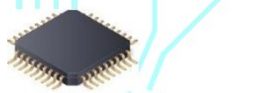
```c
#include "STM32F4xx.h"

char set = 0x38;
char disp_on= 0x0E;
char mode=0x06;
char ddram2L=0xC0;
char datoT[8]={'D','a','t','o','s',' ','R','x'};

void send_comando(char a){
  GPIOC->ODR = a |(1UL<<9);
  for(int dl=0;dl<10000;dl++);
  GPIOC->ODR &=~(1UL<<9);}
void send_dato(char b){
  GPIOC ->ODR = b |(3<<8);
  for(int dl=0;dl<5000;dl++);
  GPIOC->ODR &=~(1UL<<9); }
extern "C" {
  void USART2_IRQHandler(void){ //Interrupcion por dato recibido.
    if (USART2->SR & 0x20){
    send_comando(ddram2L+10);send_dato(USART2->DR);};
    }
}
```




```c
int main(void){
  RCC -> AHB1ENR = 0XF; //PUERTOS
  GPIOB -> MODER = 0X55555555; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS
  GPIOC -> MODER = 0X55555555;
//*********************************************************************************************
  //CONFIGURACION UART
  RCC->APB1ENR |= (1UL<<17); // Enable clock for USART2 (set el pin 17)
                             // 8N1-> M=00, no es necesario escribir el registro
  USART2->BRR = 0x683;     // 9600 Baudios, fclk=16Mhz, por defecto el HSI
  USART2->CR1 = 0x012C;    // Tx habilitado, Rx habilitado, Interrup de Rx habilitado, 8N1
  USART2->CR1 |= 0x2000;   // CR2 se deja por defecto pues 1 bit de stop es 00

  NVIC_EnableIRQ(USART2_IRQn);      // Llama interrupción del USART2

  GPIOD->MODER |=  0x2800;  //pines PD5  PD6 en modo alterno (RX)
  GPIOD->AFR[0] |= 0x007700000; // PD5 -> AF7=USART2 TX,PD6 -> AF7=USART2 RX,
  //*********************************************************************************************
  //CONFIGURACION LCD
  send_comando(set);
  send_comando(disp_on);
  send_comando(mode);
  for (int ix=0;ix<8;ix++){
  send_dato(datoT[ix]); }

  while(true){
}}
```

Ejercicio de Clase

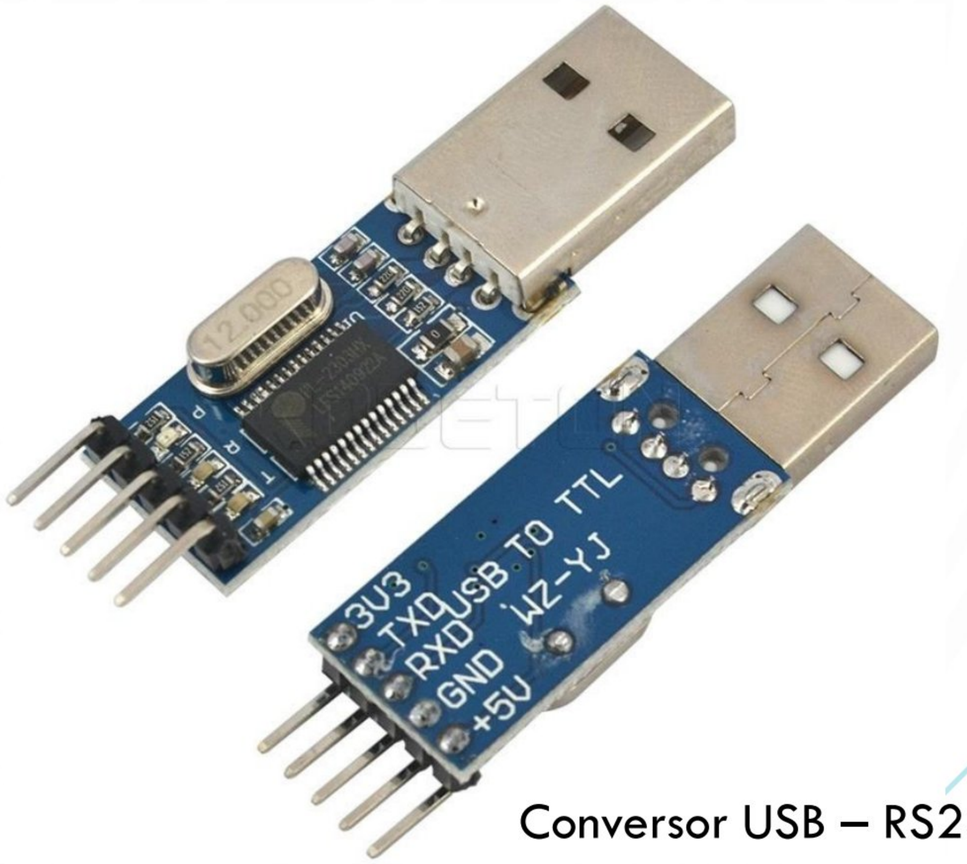
Realizar un programa para microcontrolador que reciba por puerto serial el limite de cuenta para un contador de dos cifras, el micro debe reenviar el valor de la cuenta por comunicación serial, hasta el limite establecido.



INGENIERÍA MECATRÓNICA UMNG

MATLAB®

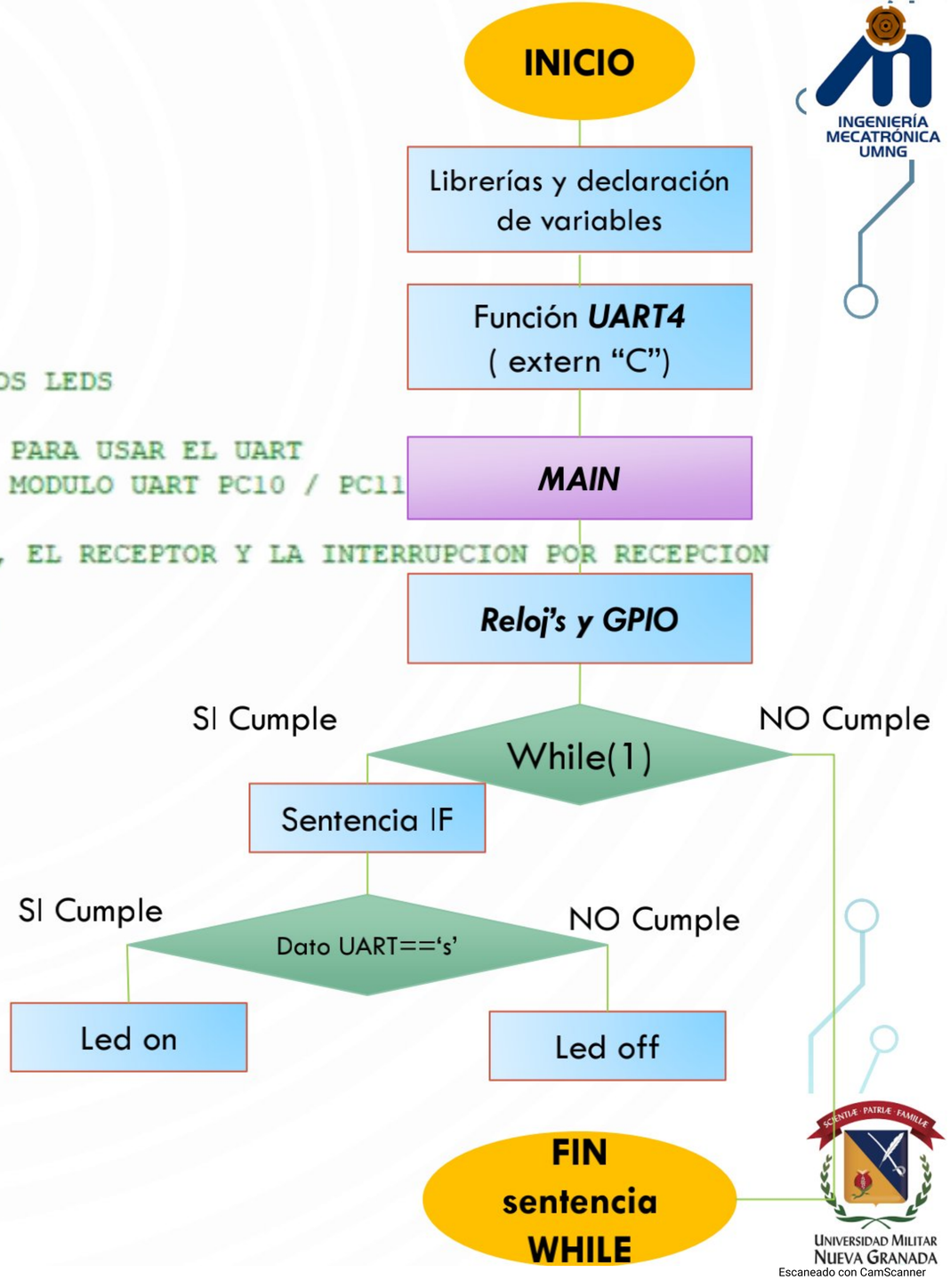
The Language of Technical Computing



Conversor USB – RS232

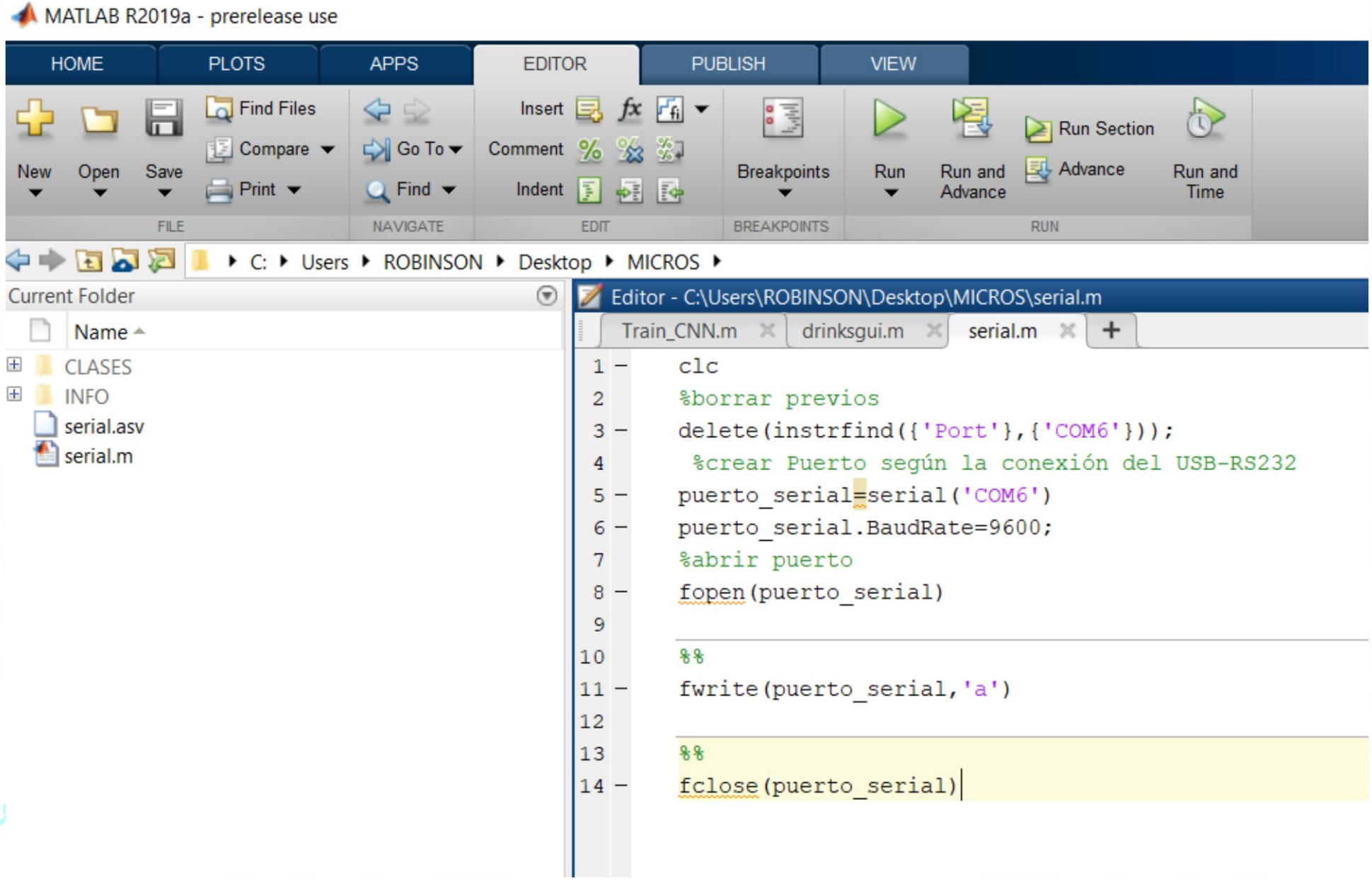
UNIVERSIDAD MILITAR NUEVA GRANADA

```c
#include "stm32f7xx.h"
int valor;
int main(void){
  RCC -> AHB1ENR |= 0X6; //PUERTOS B Y C
  RCC -> APB1ENR |= 0X80000; //HABILITAR EL UART4
  GPIOB -> MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS

  GPIOC -> MODER = 0XA00000; //COLOCAR LOS PINES EN MODO ALTERNANTE PARA USAR EL UART
  GPIOC -> AFR[1] = 0X8800; //DEFINIR LA FUNCION ALTERNANTE PARA EL MODULO UART PC10 / PC11
  UART4 -> BRR = 0X683; //VELOCIDAD DE 9600 BAUDIOS
  UART4 -> CR1 = 0X2D; //HABILITAR EL UART, HABILITAR EL TRANSMISOR, EL RECEPTOR Y LA INTERRUPCION POR RECEPCION

  NVIC_EnableIRQ(UART4_IRQn); //HABILITAR LA INTERRUPCION DEL UART4

  while(1){
    if(UART4 -> RDR=='s')
    {GPIOB->ODR=1;}
    else GPIOB->ODR =0;
  }
}
extern "C"{
  void UART4_IRQHandler(void){
    valor = UART4 -> RDR; //ALMACENAR EL VALOR DE RECEPTOR
    if (valor>5)  {GPIOB->ODR=128;}
    else GPIOB->ODR =0;
  }
}
```

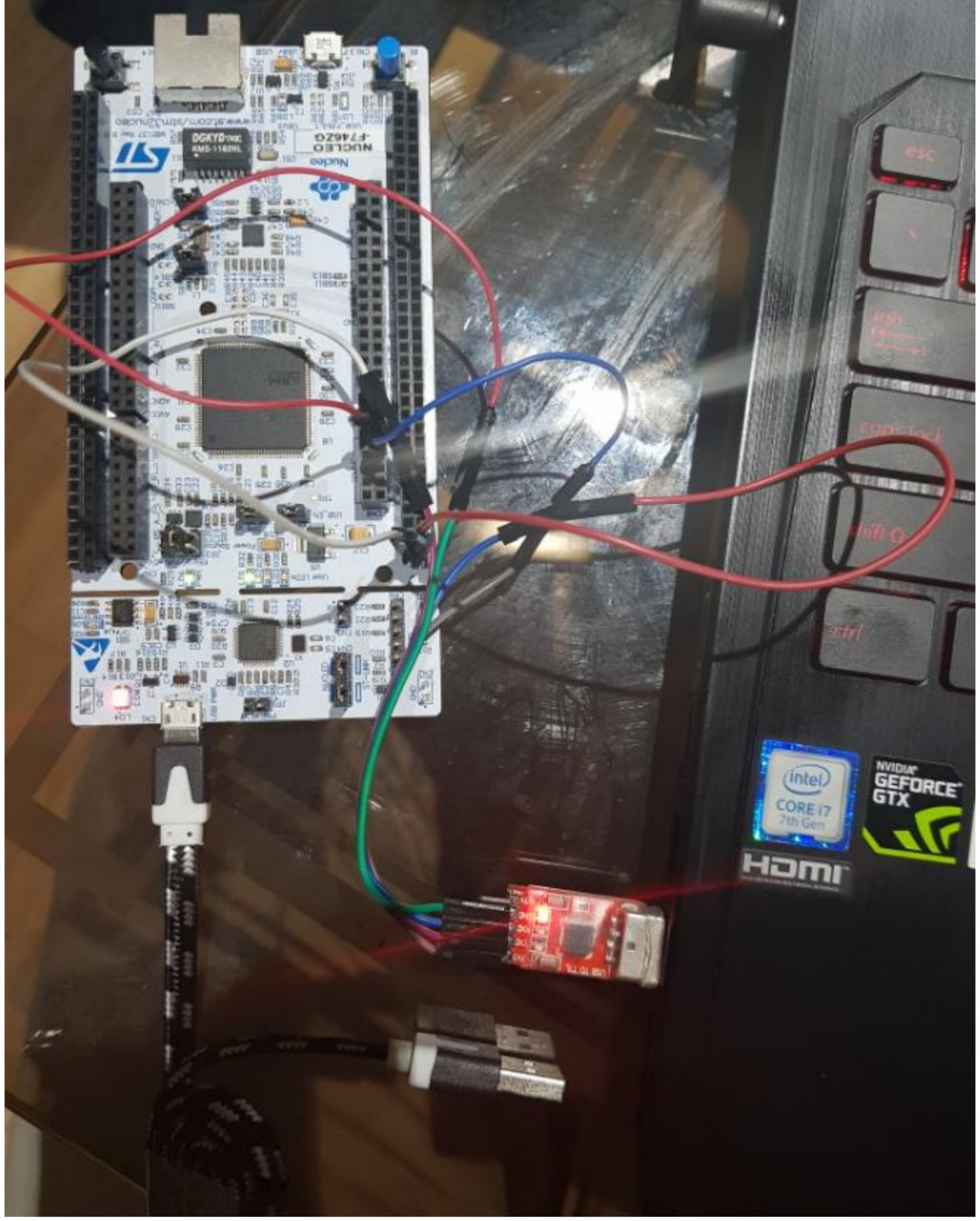


CONFIGURACIÓN DE MATLAB

INGENIERÍA MECATRÓNICA UMNG



UNIVERSIDAD MILITAR NUEVA GRANADA

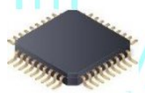


INGENIERÍA MECATRÓNICA UMNG

UNIVERSIDAD MILITAR NUEVA GRANADA

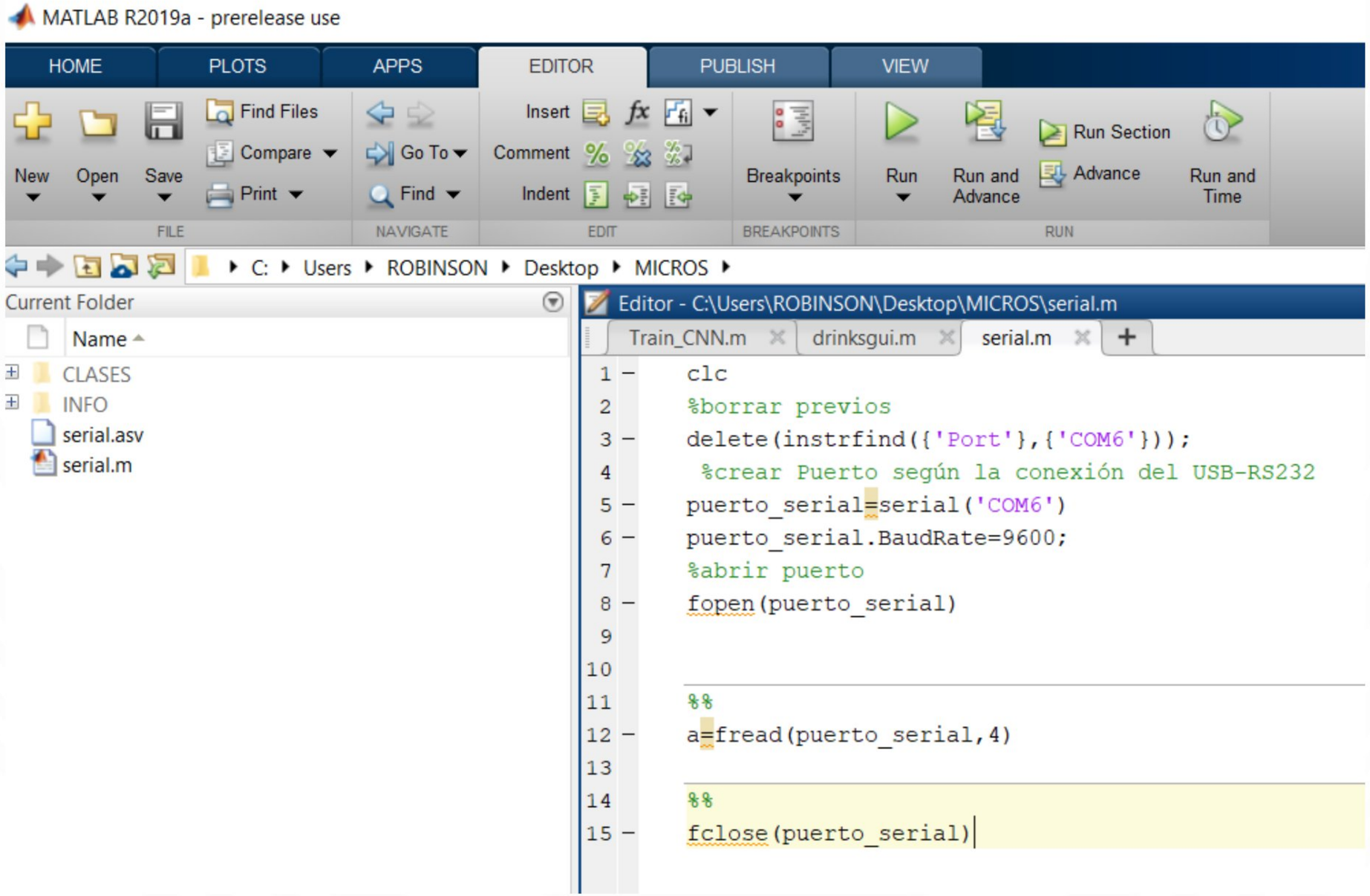
INGENIERÍA MECATRÓNICA UMNG

```c
#include "stm32f7xx.h"

int i,Rx;
char datos[5]={120,15,36,3};
float valor;

int main(void){
  RCC -> AHB1ENR |= 0X6; //PUERTOS B Y C
  RCC -> APB1ENR |= 0X80000; //HABILITAR EL UART4

  GPIOB -> MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS

  GPIOC -> MODER = 0XA00000; //COLOCAR LOS PINES EN MODO ALTERNANTE PARA USAR EL UART
  GPIOC -> AFR[1] = 0X8800; //DEFINIR LA FUNCION ALTERNANTE PARA EL MODULO UART PC10 / PC11
  UART4 -> BRR = 0X683; //VELOCIDAD DE 9600 BAUDIOS
  UART4 -> CR1 = 0X2D; //HABILITAR EL UART, HABILITAR EL TRANSMISOR, EL RECEPTOR Y LA INTERRUPCION POR RECEPCION

  NVIC_EnableIRQ(UART4_IRQn); //HABILITAR LA INTERRUPCION DEL UART4
  GPIOB -> ODR=1;
  while(1){
    if ((GPIOC -> IDR &= 0x2000)==0x2000){GPIOB -> ODR=0xfff;
    for (i=0;i<5;i++){UART4 -> TDR =datos[i];
    while((UART4 -> ISR &=0x80)==0);
    }
  }
  GPIOB -> ODR=1;}
}
```



UNIVERSIDAD MILITAR NUEVA GRANADA

CONFIGURACIÓN DE MATLAB



Tarea:

Realizar una GUIDE en MATLAB que visualice el valor de cuenta de un contador de eventos programado en el microcontrolador, desde Matlab se debe enviar el limite hasta el cual debe contar, a lo cual prendera un led, localmente el micro muestra la cuenta en una GLCD.