



MICROS 32 BITS STM - UART

REALIZADA POR: ROBINSON JIMENEZ MORENO

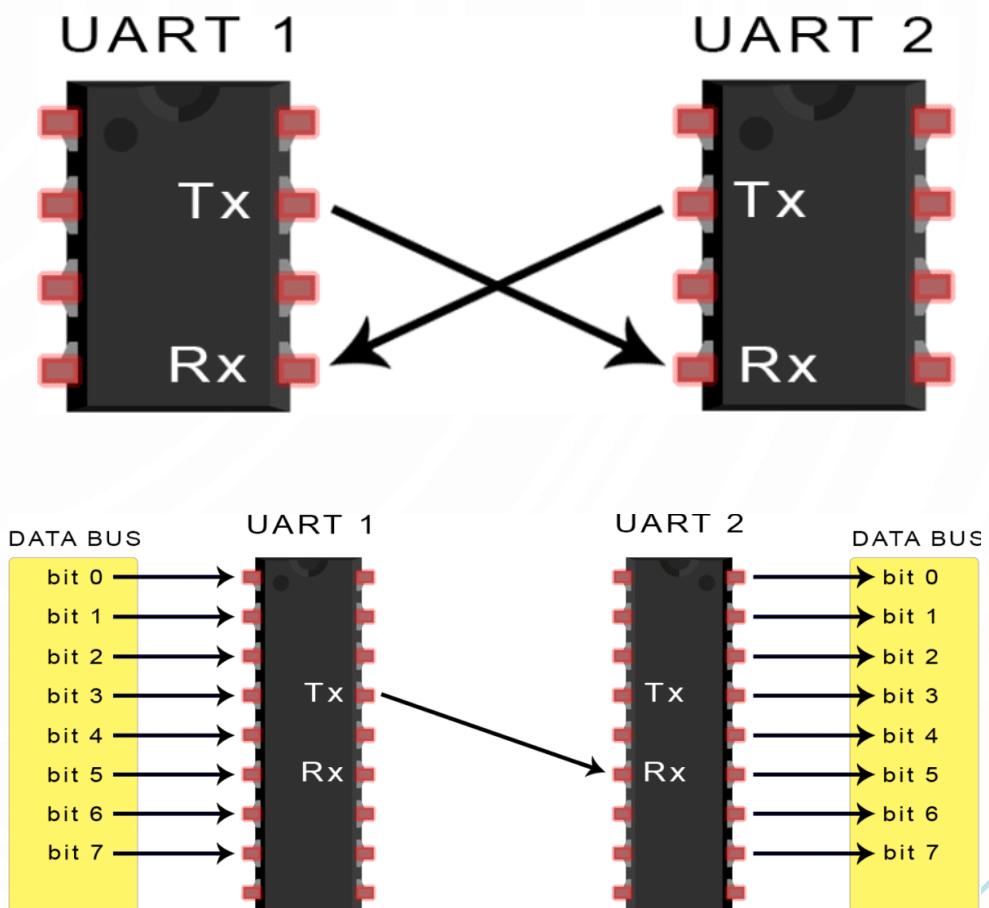
PROFESORA: LUISA FERNANDA GARCÍA



UART - 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, recibidos en serie y pasados a formato paralelo para que puedan ser procesados. El módulo UART usa solo dos cables para la transmisión de información, Tx y Rx.



UART - 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

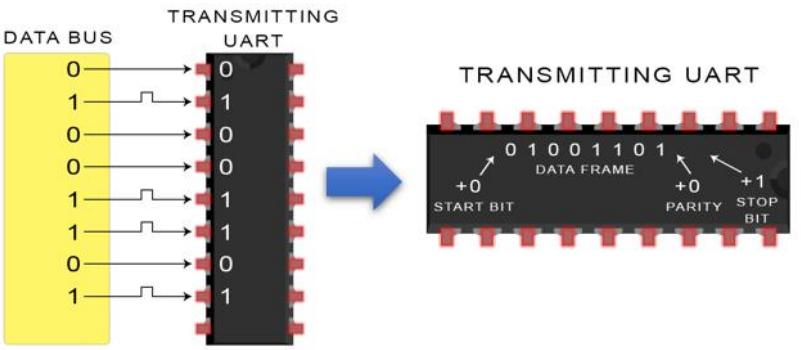
UART

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

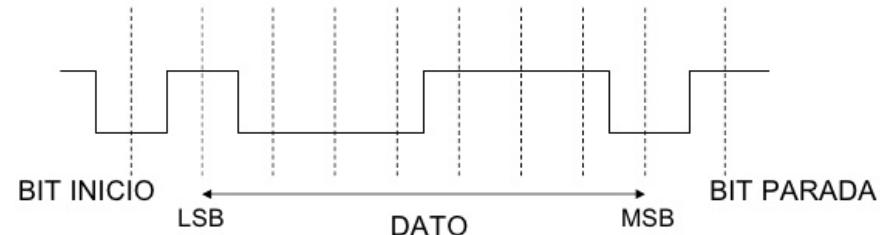
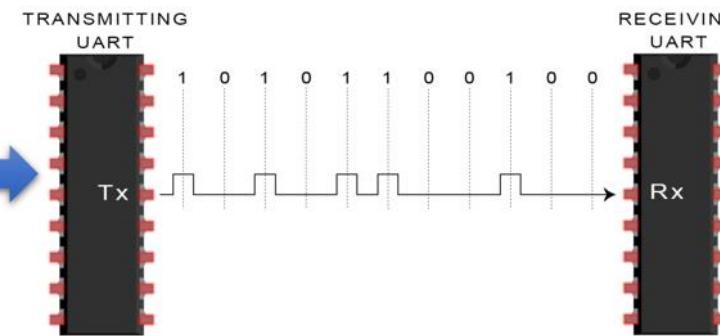
Bit de parada: Para señalar el final del paquete de datos, el UART de transmisión pasa un voltaje bajo a uno alto durante al menos dos duraciones de bit.

El módulo UART 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).

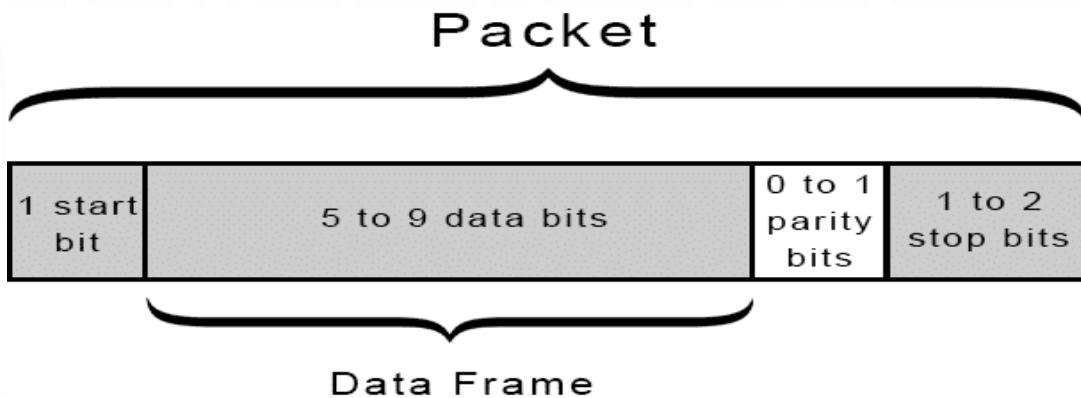
Ambos UART deben estar configurados para transmitir y recibir la misma estructura de paquetes de datos (trama) y con el mismo Baud Rate.



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



Los datos transmitidos por UART se organizan en paquetes. Cada paquete contiene 1 bit de inicio, 5 a 9 bits de datos (según el UART), 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>

Guia comunicación serial de MathLab

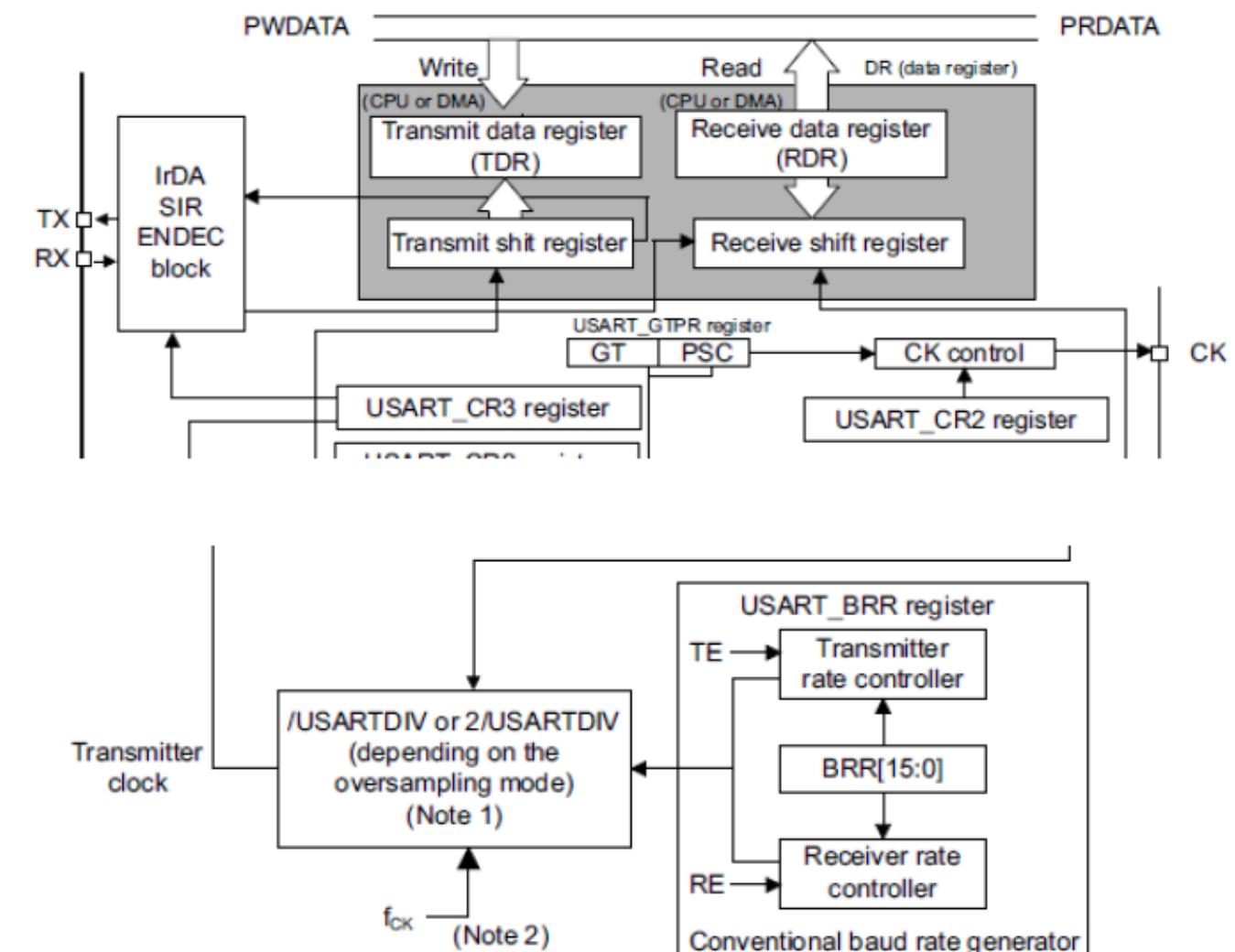
https://es.mathworks.com/help/matlab/matlab_external/overview-of-the-serial-port.html

Serial data are transmitted and received through these pins in normal USART mode.

The frames are comprised of:

- An Idle Line prior to transmission or reception
 - A start bit
 - A data word (7, 8 or 9 bits) least significant bit first
 - 0.5, 1, 1.5, 2 stop bits indicating that the frame is complete
 - The USART interface uses a baud rate generator
-
- A status register (`USART_ISR`)
 - Receive and transmit data registers (`USART_RDR`, `USART_TDR`)
 - A baud rate register (`USART_BRR`)

Figure 316. USART block diagram



UART - STM32F4

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

REGISTROS:

UARTx -> CR1
BRR
TDR
RDR

Universal synchronous/asynchronous receiver transmitters (USART)

The device embeds USART. Refer to [Table 8: USART implementation](#) for the features implementation.

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format.

The USART peripheral supports:

- Full-duplex asynchronous communications
- Configurable oversampling method by 16 or 8 to give flexibility between speed and clock tolerance
- Dual clock domain allowing convenient baud rate programming independent from the PCLK reprogramming
- A common programmable transmit and receive baud rate of up to 27 Mbit/s when USART clock source is system clock frequency (Max is 216 MHz) and oversampling by 8 is used.
- Auto baud rate detection
- Programmable data word length (7 or 8 or 9 bits) word length
- Programmable data order with MSB-first or LSB-first shifting
- Programmable parity (odd, even, no parity)
- Configurable stop bits (1 or 1.5 or 2 stop bits)
- Synchronous mode and clock output for synchronous communications
- Single-wire half-duplex communications

Character transmission procedure

1. Program the M bits in **USART_CR1** to define the word length.
2. Select the desired baud rate using the **USART_BRR** register.
3. Program the number of stop bits in **USART_CR2**.
4. Enable the USART by writing the UE bit in **USART_CR1** register to 1.
5. Select DMA enable (DMAT) in **USART_CR3** if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
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_TDR** 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_TDR** 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.

//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	USART3 EN	USART2 EN	SPDIFRX EN

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

Bit 19 **UART4EN**: UART4 clock enable

This bit is set and cleared by software.

0: UART4 clock disabled

1: UART4 clock enabled

```
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.
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

$$USARTDIV = FCLK_{UART} / vel_deseada$$

$$USARTDIV = 16Mhz / 9600bps$$

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

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

```
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).

```
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										
			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

```

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]											
			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

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	MSBFI RST	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 ($V_{DD} = 1/\text{idle}$, Gnd=0/mark)

1: TX pin signal values are inverted. ($V_{DD} = 0/\text{mark}$, 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 ($V_{DD} = 1/\text{idle}$, Gnd=0/mark)

1: RX pin signal values are inverted. ($V_{DD} = 0/\text{mark}$, 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
RDR[8:0]															
Res.	Res.	Res.	Res.	Res.	Res.	Res.	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

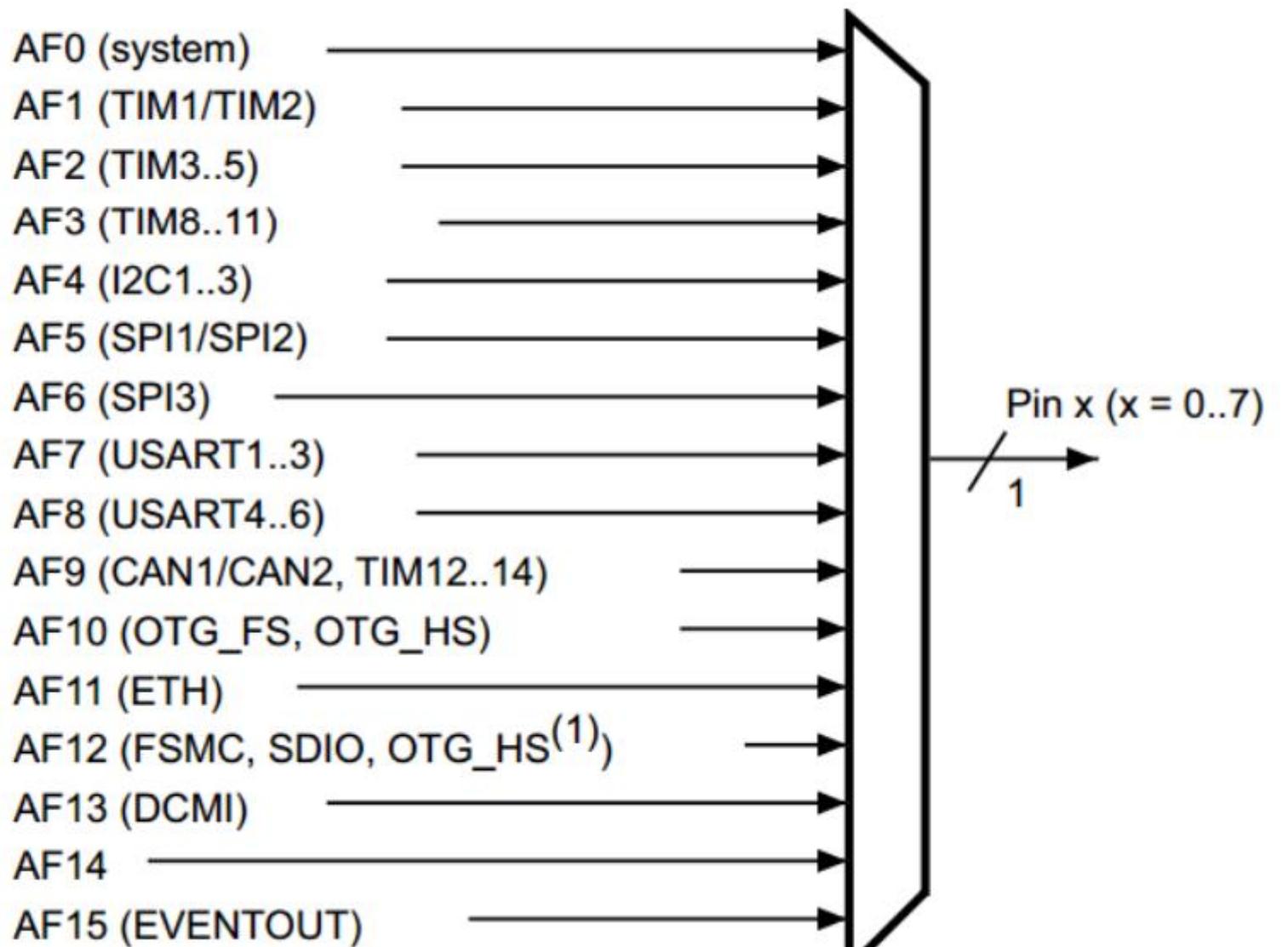
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TDR[8:0]															
Res.	Res.	Res.	Res.	Res.	Res.	Res.	rw								

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](#)).

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



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/LPTIM 1/CEC	I2C1/2/3/ 4/CEC	SPI1/2/3/ 4/5/6	SPI3/ SAI1	SPI2/3/U SART1/2/ 3/UART5/ SPDIFRX	SAI2/US ART6/UA RT4/5/7/8 /SPDIFRX	CAN1/2/T IM12/13/ 14/QUAD SPI/LCD	SAI2/QU ADSPi/O TG2_HS/ OTG1_FS	ETH/ OTG1_FS	FMC/SD MMC1/O TG2_FS	DCMI	LCD	SYS	
PA0	-	TIM2_C H1/TIM2 _ETR	TIM5_C H1	TIM8_ET R	-	-	-	USART2 _CTS	UART4_ TX	-	SAI2_SD_ B	ETH_MII_ CRS	-	-	-	EVEN TOUT	
PC11	-	-	-	-	-	-	-	SPI3_M	USART3 _RX	UART4_ RX	QUADSP _BK2_N CS	-	-	SDMMC 1_D3	DCMI_D 4	-	EVEN TOUT

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												
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												

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

1000: AF8

0001: AF1

1001: AF9

0010: AF2

1010: AF10

0011: AF3

1011: AF11

0100: AF4

1100: AF12

0101: AF5

1101: AF13

0110: AF6

1110: AF14

0111: AF7

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												
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												

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

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.	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.

Enviar un carácter

```
void send_data(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 TXIE 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
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

Enviar una frase

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

```

48 extern "C" {
49
50     void UART4_IRQHandler(void){
51         if (UART4 -> ISR &= 0x20) {           //Interrupcion por dato recibido.
52             dato = UART4 -> RDR;            // evaluar si la bandera RXNE esta activa
53         }
54     }
55 }
```

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.	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

```

#include "stm32f7xx.h"
#include "stdio.h"
#include <string.h>
#include <math.h>
#include <stdlib.h>

int i,Rx;
char datos[5]={0}, numero[4] = {0};
float valor;

int main(void){
    RCC -> AHB1ENR = 0X6; //PUERTOS B Y C
    RCC -> APB1ENR = 0X80000; //HABILITAR EL UART4
    RCC -> APB2ENR |= 0X3; //TIMER 3

    GPIOB -> MODER = 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS
    GPIOB -> OTYPER = 0X0; //PUSH PULL
    GPIOB -> OSPEEDR = 0X10004001; //VELOCIDAD MEDIA
    GPIOB -> PUPDR = 0X10004001; //PULL UPGPIOB -> ODR = 0X4081; //ENCENDER LOS 3 LEDS

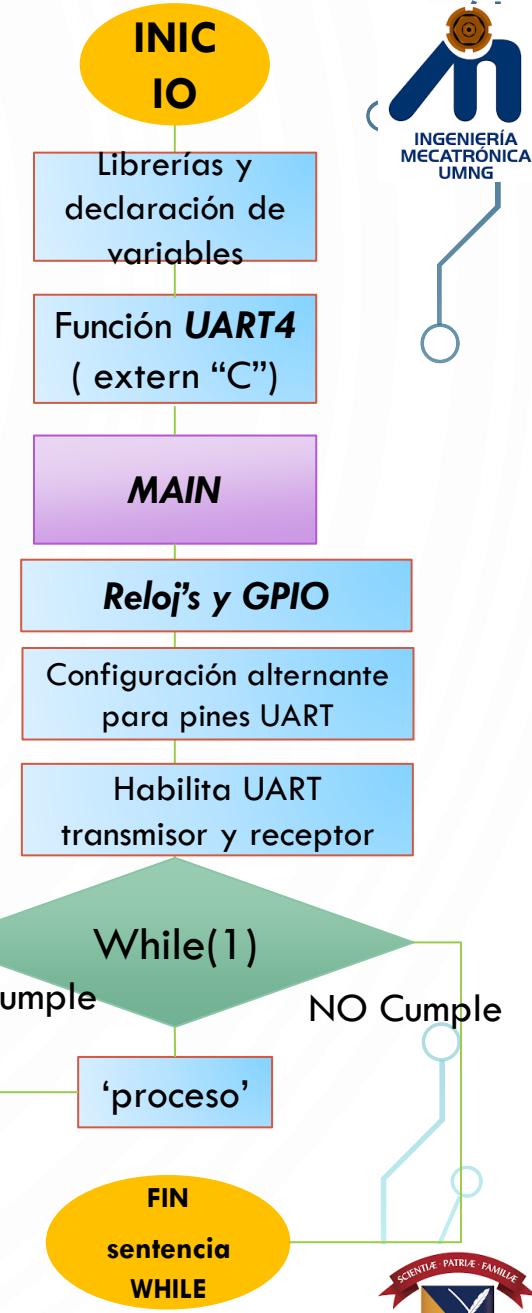
    GPIOC -> MODER = 0XA00000; //COLOCAR LOS PINES EN MODO ALTERNANTE PARA USAR EL UART
    GPIOC -> AFR[1] = 0X8800; //DEFINIR LA FUNCION ALTERNANTE PARA EL MODULO UART

    UART4 -> CRL = 0X2D; //HABILITAR EL UART, HABILITAR EL TRANSMISOR, EL RECEPTOR Y LA
                           //INTERRUPCIÓN POR RECEPCIÓN

    UART4 -> BRR = 0X683; //VELOCIDAD DE 9600 BAUDIOS
    NVIC_EnableIRQ(UART4_IRQn); //HABILITAR LA INTERRUPCIÓN DEL UART4

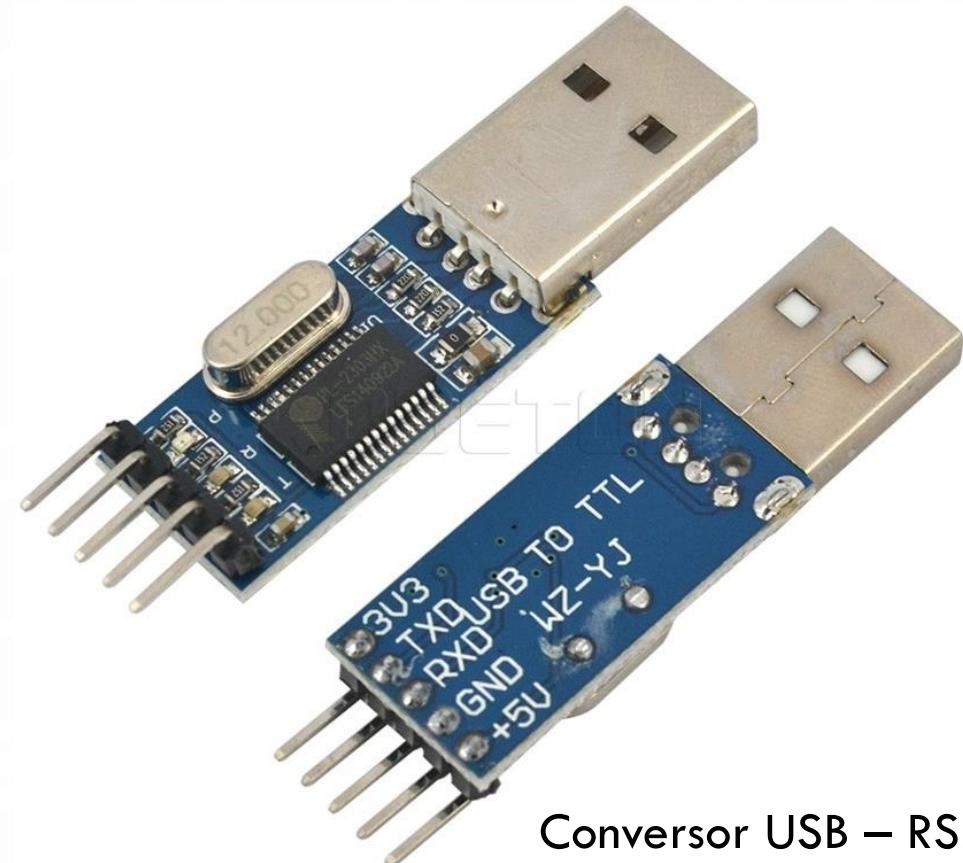
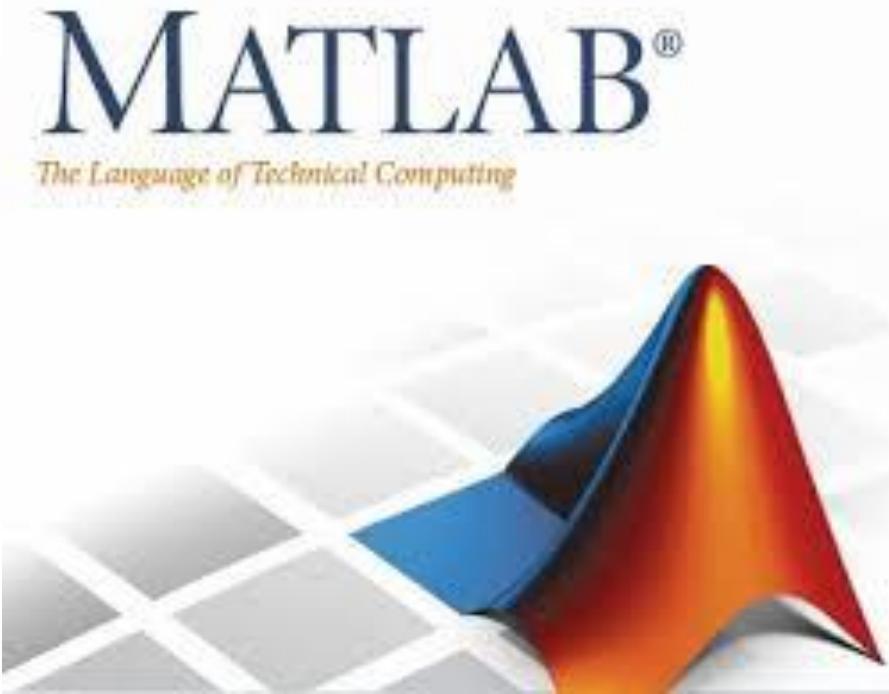
    while(1){
    }
}

```



```
extern "C"{
    void UART4_IRQHandler(void)
    {
        Rx = UART4 -> RDR; //ALMACENAR EL VALOR DE RECEPTOR
        datos[i] = Rx; //Vector que almacena cada digito de entrada
        i++;
        if(i > 3){
            if(datos[0] == 'T'){
                for(int j = 1; j < 4; j++){
                    numero[j-1] = datos[j];
                }
            }
            valor = atoi(numero);
            i = 0;
        }
    }
}
```

Atoi
Convierte una cadena a su valor numérico



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	I2C1/2/3/4/5/6	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/UART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/TG2_HSI/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/O/TG2_FS	DCMI	LCD	SYS	
Port C	PC4	-	-	-	-	-	I2S1_MCK	-	-	SPDIFRX_IN2	-	-	ETH_MII_RXD0/ETH_RMII_RXD0	FMC_SDNE0	-	-	EVEN TOUT
	PC5	-	-	-	-	-	-	-	-	SPDIFRX_IN3	-	-	ETH_MII_RXD1/ETH_RMII_RXD1	FMC_SDCKE0	-	-	EVEN TOUT
	PC6	-	-	TIM3_CH1	TIM8_CH1	-	I2S2_MCK	-	-	USART6_TX	-	-	-	SDMMC1_D6	DCMI_D0	LCD_HSYNC	EVEN TOUT
	PC7	-	-	TIM3_CH2	TIM8_CH2	-	-	I2S3_MCK	-	USART6_RX	-	-	-	SDMMC1_D7	DCMI_D1	LCD_G6	EVEN TOUT
	PC8	TRACE_D1	-	TIM3_CH3	TIM8_CH3	-	-	-	UART5_RTS	USART6_CK	-	-	-	SDMMC1_D0	DCMI_D2	-	EVEN TOUT
	PC9	MCO2	-	TIM3_CH4	TIM8_CH4	I2C3_SD_A	I2S_CKIN	-	UART5_CTS	-	QUADSP1_BK1_IO0	-	-	SDMMC1_D1	DCMI_D3	-	EVEN TOUT
	PC10	-	-	-	-	-	-	SPI3_SCK/I2S3_CK	USART3_TX	USART4_TX	QUADSP1_BK1_IO1	-	-	SDMMC1_D2	DCMI_D8	LCD_R2	EVEN TOUT
	PC11	-	-	-	-	-	-	SPI3_MISO	USART3_RX	USART4_RX	QUADSP1_BK2_NCS	-	-	SDMMC1_D3	DCMI_D4	-	EVEN TOUT

```
#include <math.h>
#include <stdlib.h>

int i,Rx;
char datos[5]={0}, numero[4] = {0};
float valor;

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

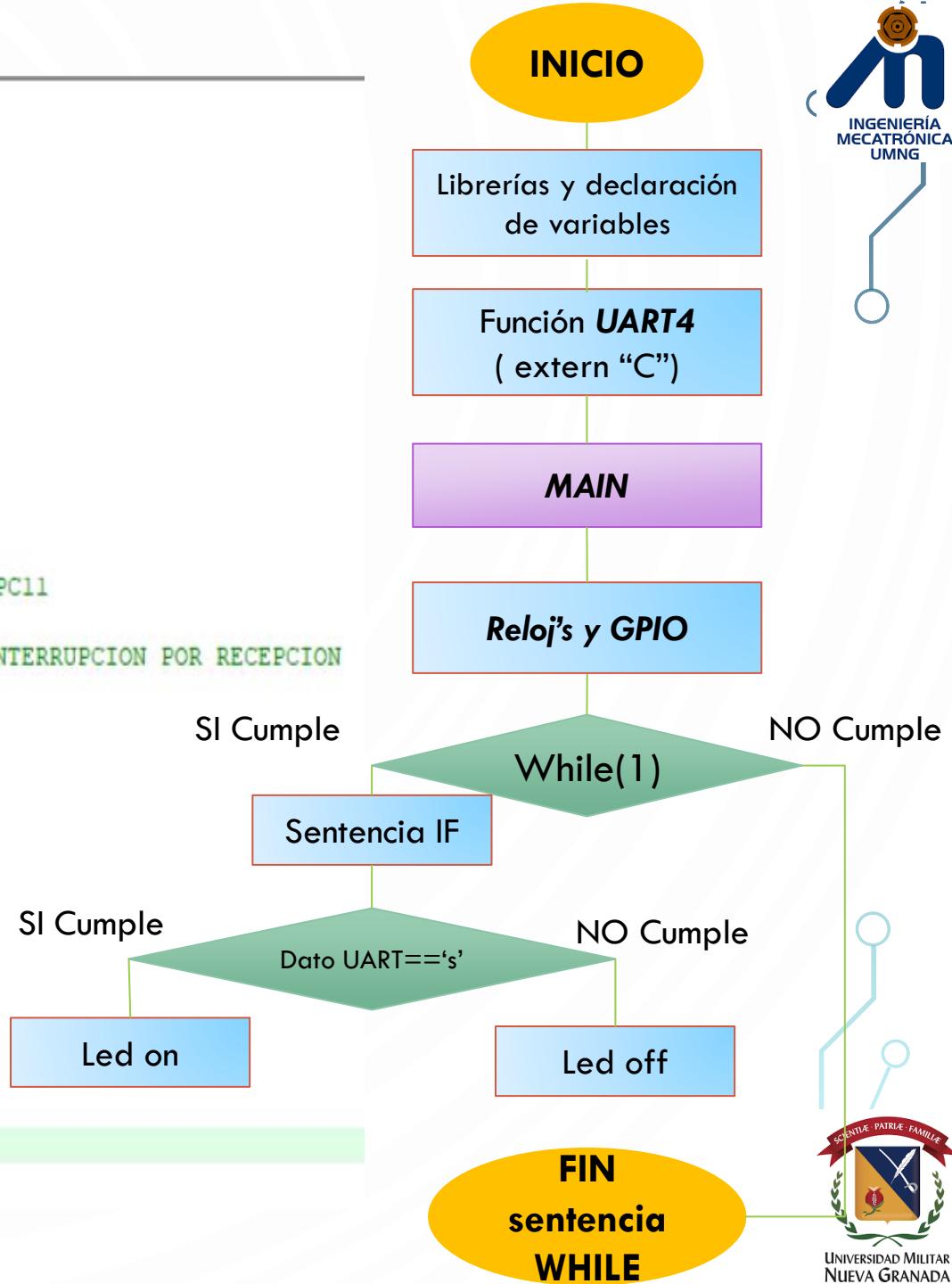
    GPIOB ->MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS

    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 INTERRUPCIÓN POR RECEPCION

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

    while(1){
        if(UART4 ->RDR=='s') //Se puede preguntar directamente sobre valor ?
            {GPIOB->ODR=1;}
        else GPIOB->ODR =0;
    }
}

extern "C"
{
    void UART4_IRQHandler(void)
    {
        valor = UART4 ->RDR; //ALMACENAR EL VALOR DE RECEPTOR
    }
}
```



MATLAB R2019a - prerelease use

HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Find Files Compare Go To Print Find

FILE NAVIGATE EDIT BREAKPOINTS RUN

C:\Users\ROBINSON\Desktop\MICROS\serial.m

Current Folder

Name CLASES INFO serial.asv serial.m

```
1 - clc
2 %borrar previos
3 - delete(instrfind({'Port'},{'COM6'}));
4 %crear Puerto según la conexión del USB-RS232
5 puerto_serial=serial('COM6')
6 puerto_serial.BaudRate=9600;
7 %abrir puerto
8 fopen(puerto_serial)

%%
11 - fwrite(puerto_serial,'a')

%%
13 - fclose(puerto_serial)|

delete (puerto_serial);
```

```
#include "stm32f7xx.h"
#include "stdio.h"
#include <string.h>
#include <math.h>
#include <stdlib.h>

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

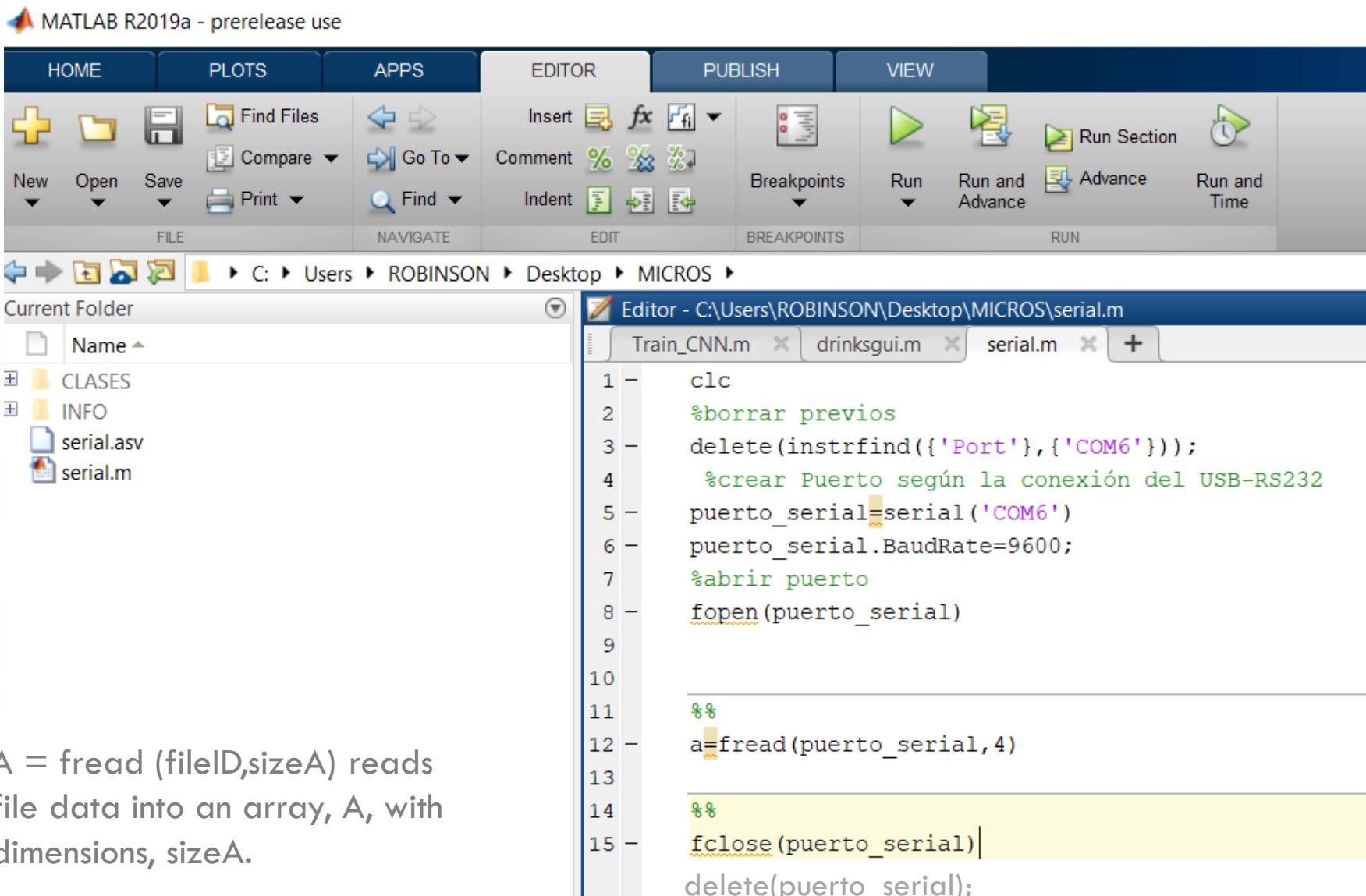
int main(void){
    RCC -> AHB1ENR |= 0X6; //PUERTOS B Y C
    RCC -> APB1ENR |= 0X80000; //HABILITAR EL UART4

    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 INTERRUPCIÓN POR RECEPCION

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

CONFIGURACIÓN DE MATLAB



A = fread (fileID,sizeA) reads file data into an array, A, with dimensions, sizeA.

```
Editor - C:\Users\ROBINSON\Desktop\MICROS\serial.m
1 - clc
2 - %borrar previos
3 - delete(instrfind({'Port'},{'COM6'}));
4 - %crear Puerto según la conexión del USB-RS232
5 - puerto_serial=serial('COM6')
6 - puerto_serial.BaudRate=9600;
7 - %abrir puerto
8 - fopen(puerto_serial)

9

10
11
12 - a=fread(puerto_serial,4)
13
14
15 - fclose(puerto_serial)
delete(puerto_serial);
```

```
1 void send_data(char dt) {
2     UART4->TDR = dt;
3     while ((UART4->ISR &= 0x80)==0); //esperar hasta que TXE sea 1,
4     //lo que indica que se envio el dato
5 }
6
7 void UART4_IRQHandler(void) {      //Interrupcion por dato recibido.
8     if (UART4->ISR & 0x20) {          // evaluar si la bandera RXNE esta activa
9         datoU = UART4->RDR;
10    }
```



TAREA:

Realizar una aplicación de transmisión serial UART entre MATLAB o su programa de preferencia y la STM. En la STM se debe llevar el conteo del número de veces que se presione un pulsador, conteo que se debe mostrar en una pantalla LCD. Desde MATLAB se debe indicar el valor límite del conteo para que cuando se llegue a ese valor se prenda un LED.

Se debe realizar en los grupos de trabajo, documentar el código, generar un video por medio del cual se sustente el funcionamiento del sistema y se explique el diseño del código, subir el video a la actividad. Adjuntar el código en un archivo aparte.