MICROS 32 BITS
STM - UART

ROBINSON JIMENEZ MORENO

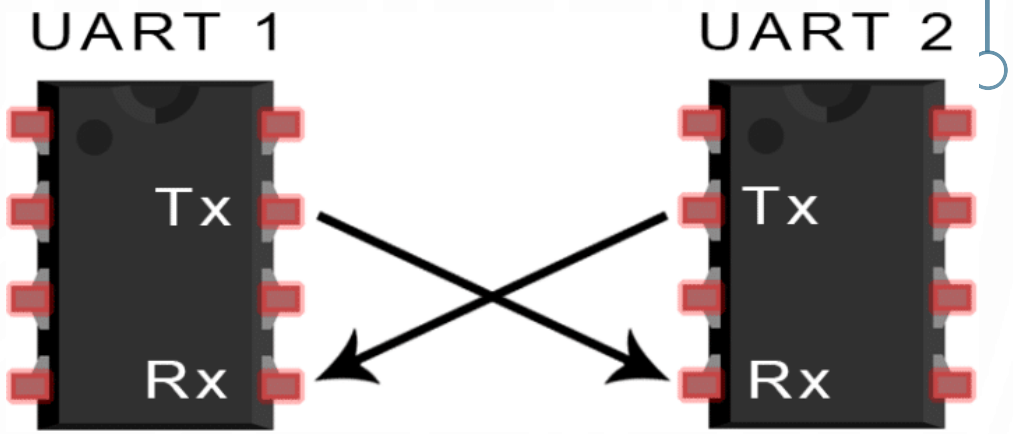
UNIVERSIDAD MILITAR NUEVA GRANADA

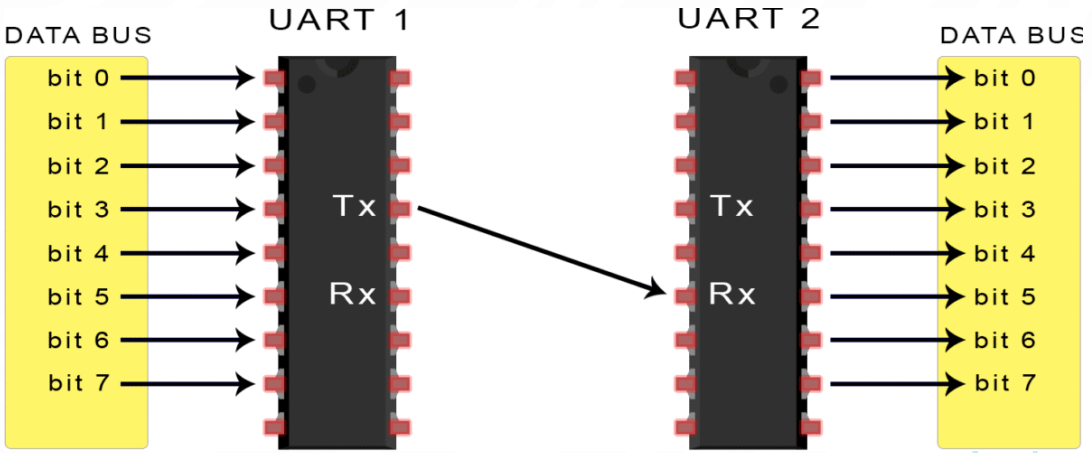
INGENIERÍA MECATRÓNICA UMNG

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, 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/

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

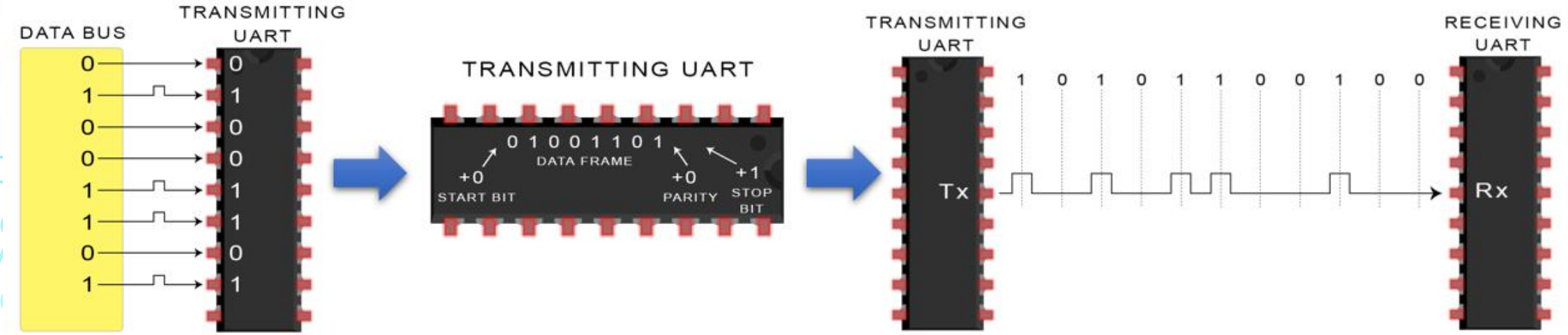
UART

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.

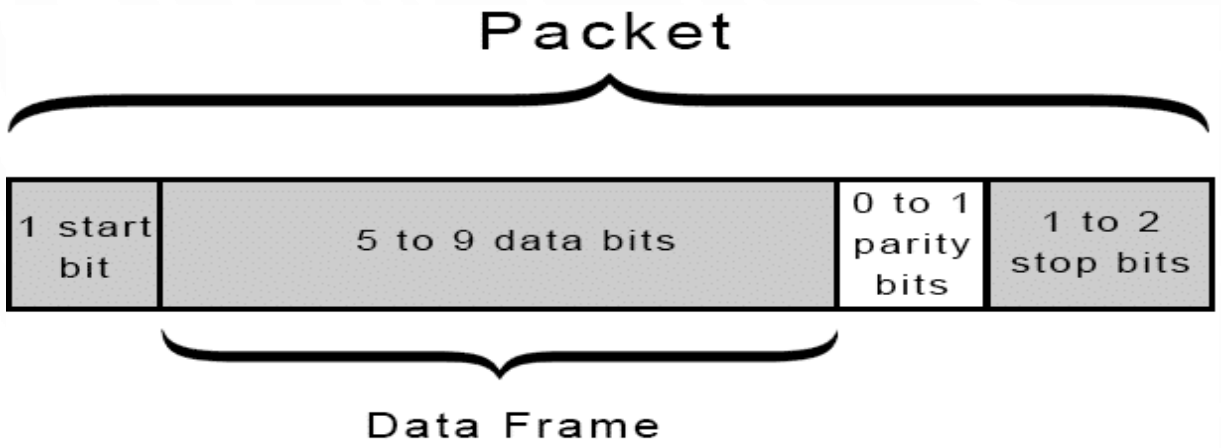
El módulo UART realiza una comunicación asíncrona, lo que significa que no requiere señal de reloj para la transmisión de la información. En lugar de ello, se agrega un bit de inicio y uno de parada al paquete de datos, los cuales indican el comienzo y el final del proceso de transferencia de datos. 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.

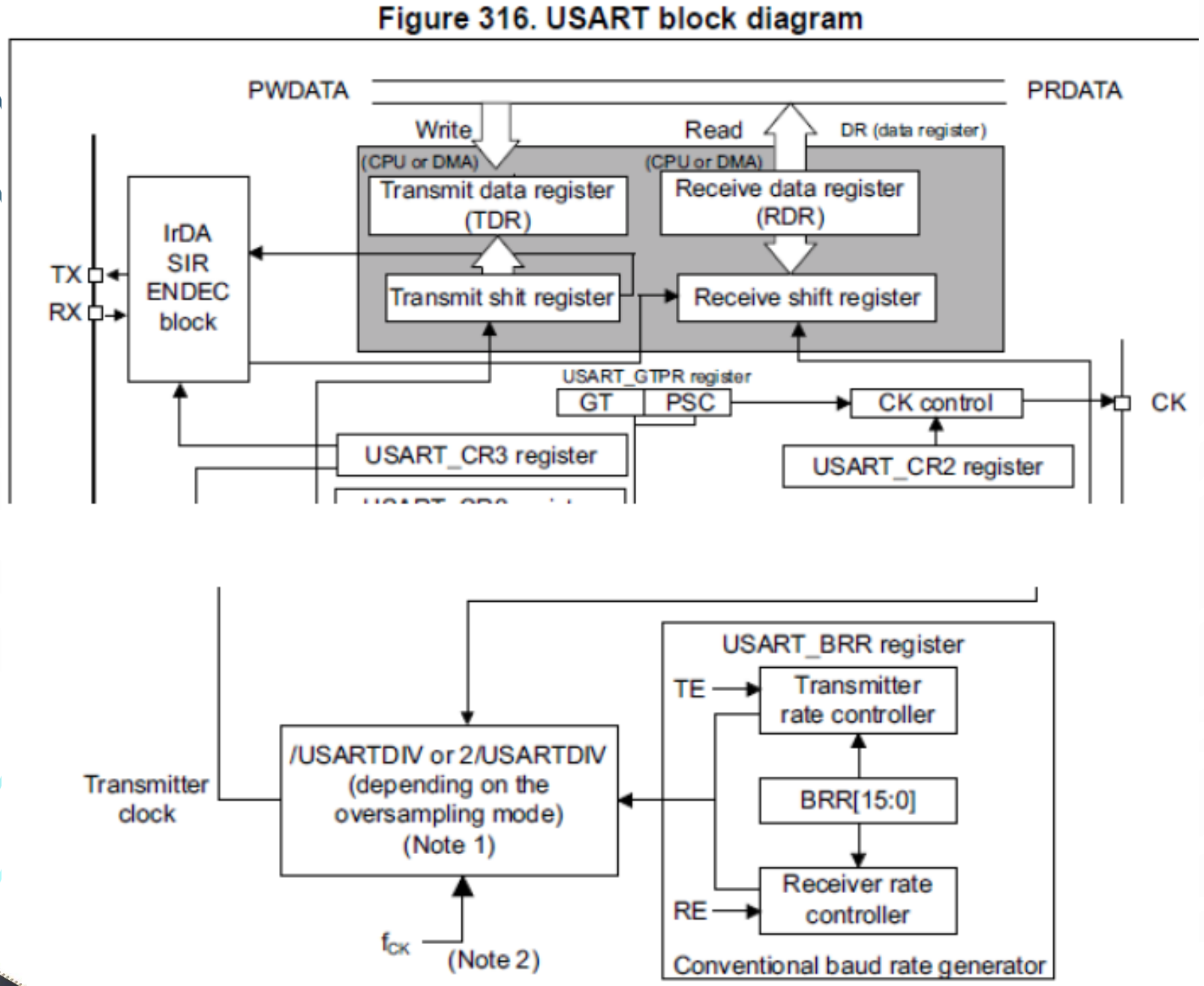


http://www.circuitbasics.com/basics-uart-communication/

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

UART - STM32F4



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

REGISTROS:

UARTx -> **CR1**
BRR
TDR
RDR

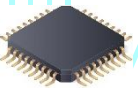
Universal synchronous/asynchronous receiver transmitters (USART)

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.

http://download.mikroe.com/documents/compilers/mikroc/arm/help/i2c_library.htm

```
//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

```
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

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

$$USARTDIV = 16Mhz/9600bps$$

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

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

```
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	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

```
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

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

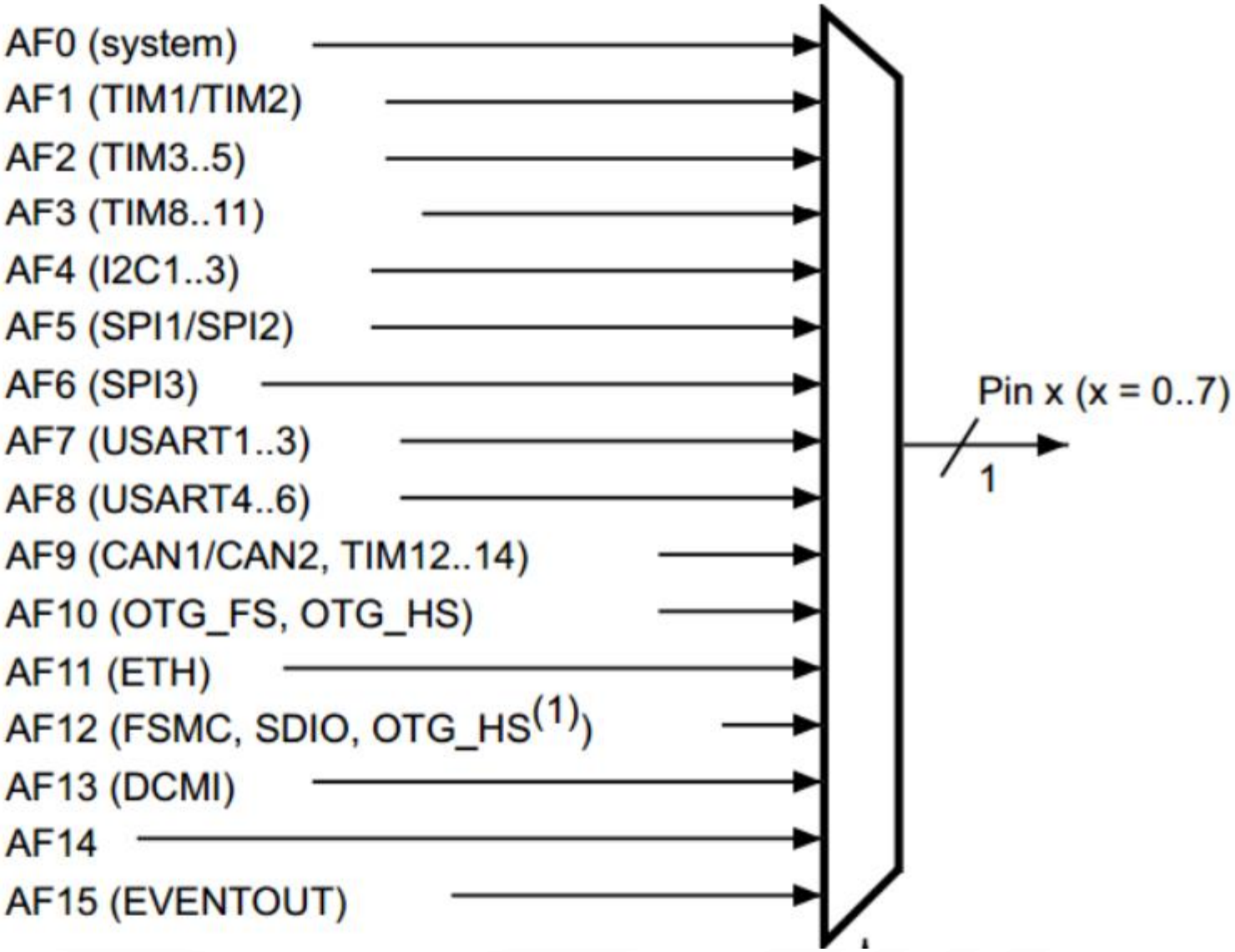
Bits 31:0 **AFRy[3:0]:** Alternate function selection for port x pin y (y = 8..15)

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.



- **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.

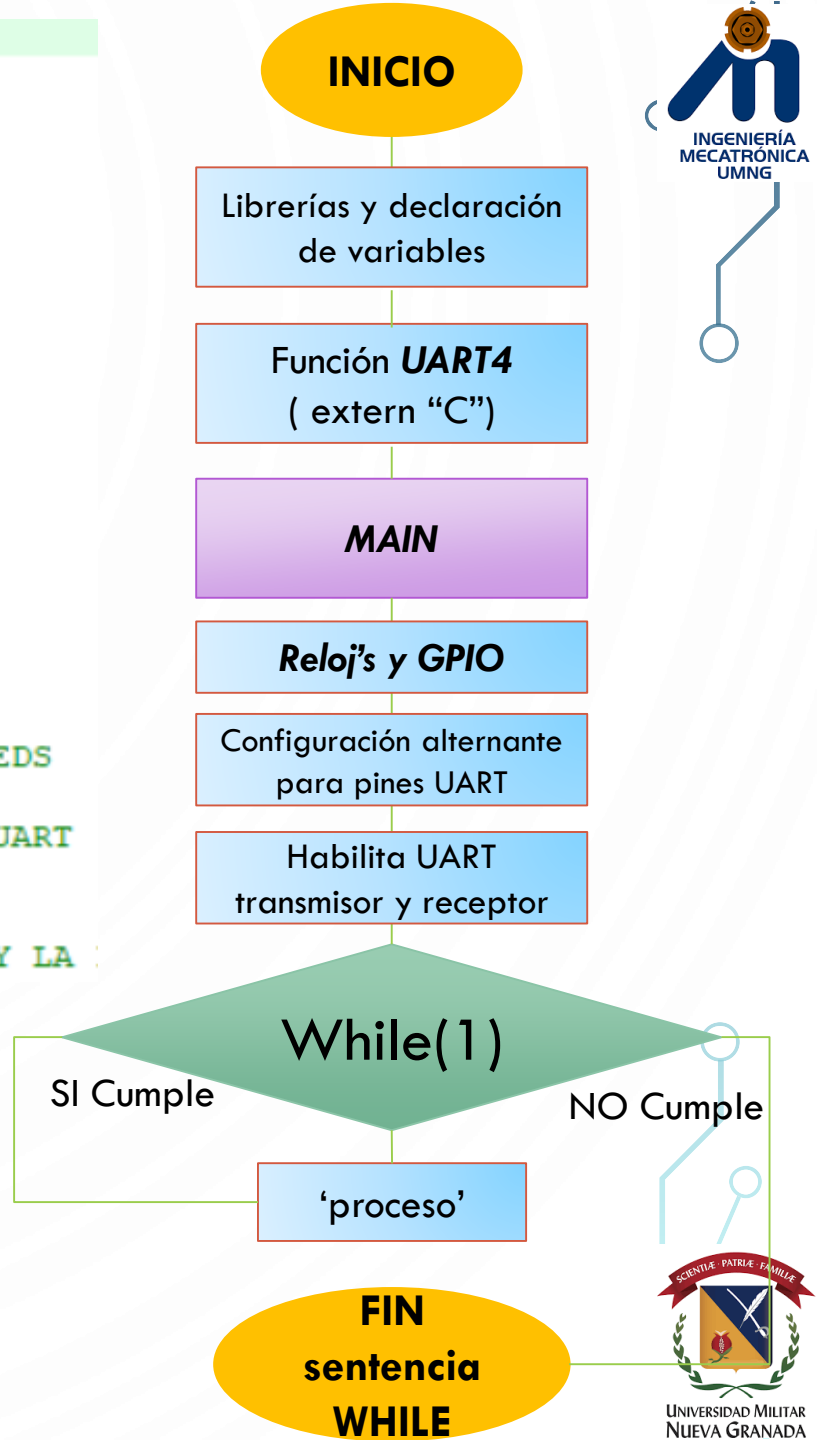
31.8.8 Interrupt and status register (USART_ISR)

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

```c
#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 -> CR1 = 0X2D; //HABILITAR EL UART, HABILITAR EL TRANSMISOR, EL RECEPTOR Y LA
                                                    INTERRUPCION POR RECEPCION

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

  while(1){

  }
}
```

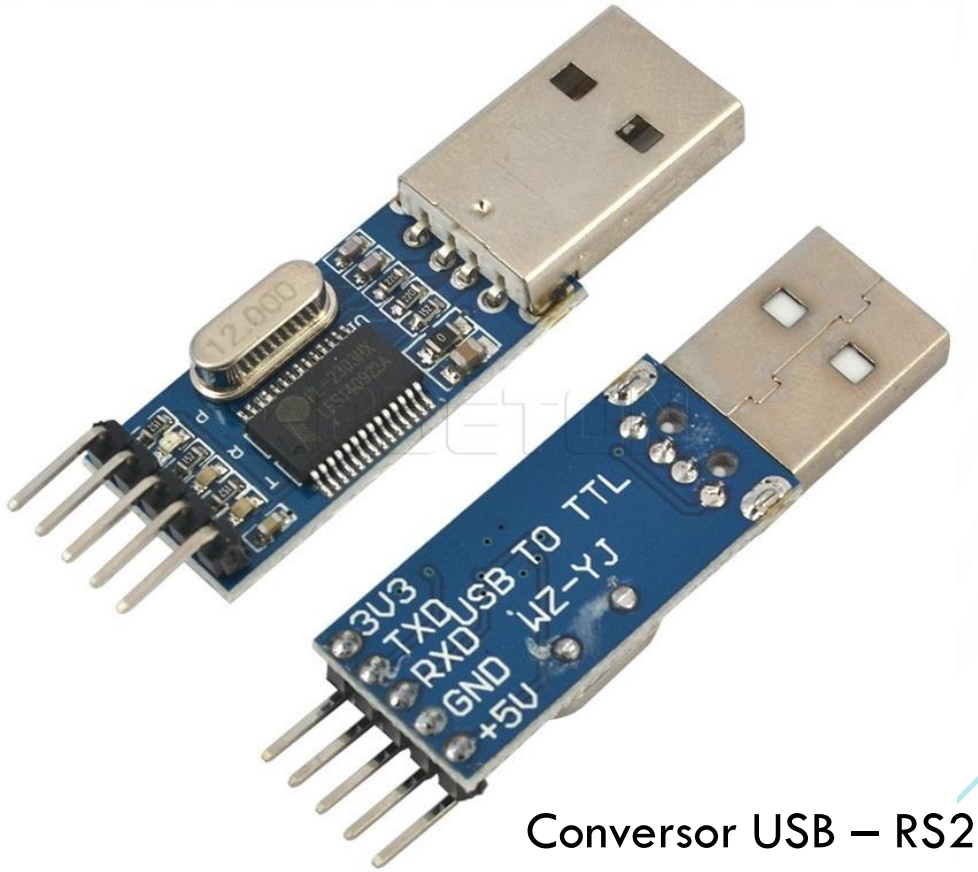


```
extern "C"{
   void UART4_IRQHandler(void)
   {
     Rx = UART4 -> RDR; //ALMACENAR EL VALOR DE RECEPTOR
     datos[i] = Rx; //Vector que almacena cada digito de entrada
     i++;
     if(i > 3){
       if(datos[0] == 'T'){
         for(int j = 1; j < 4; j++){
           numero[j-1] = datos[j];
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

INGENIERÍA MECATRÓNICA UMNG

MATLAB®

The Language of Technical Computing

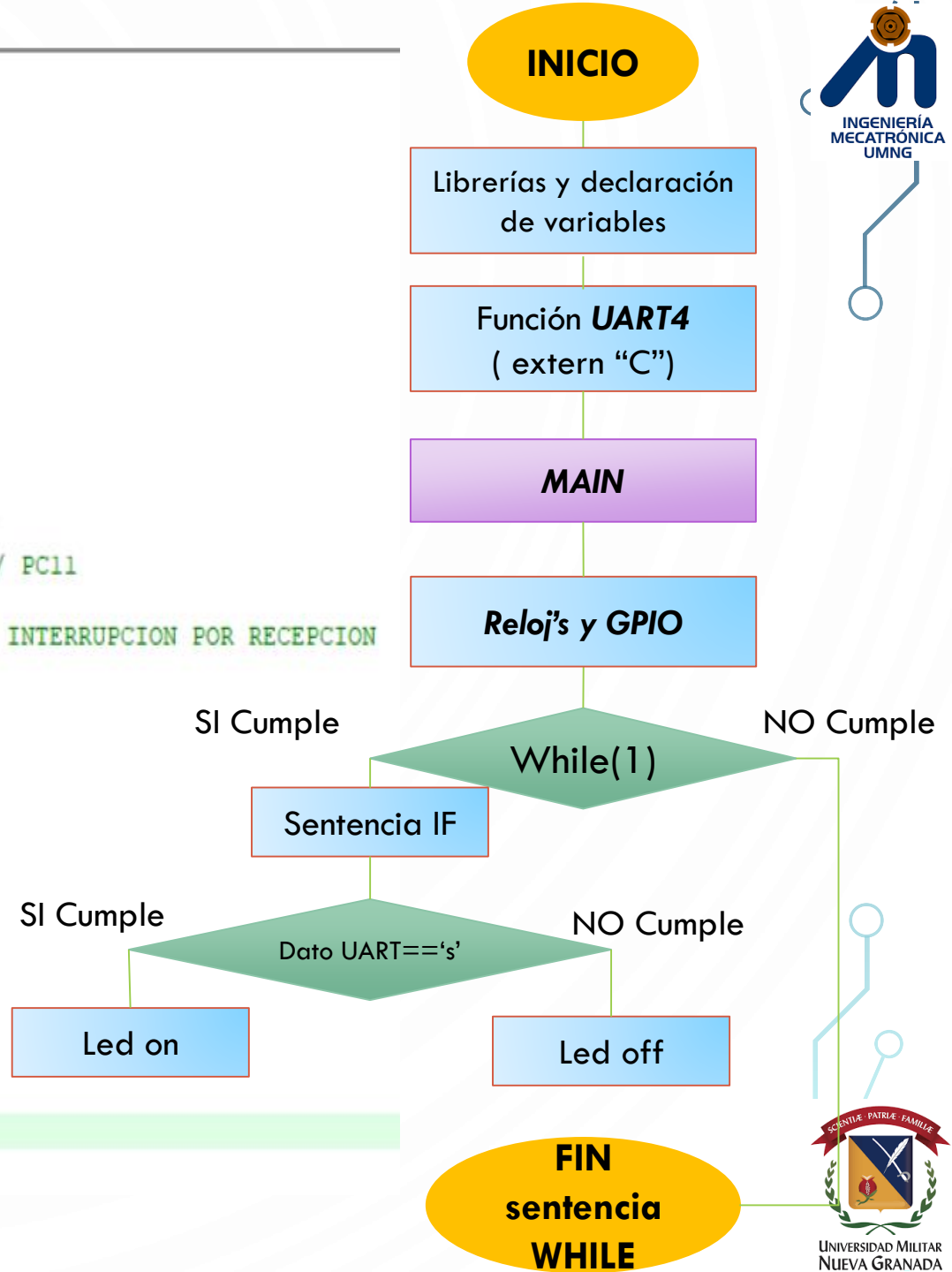


Conversor USB – RS232

UNIVERSIDAD MILITAR NUEVA GRANADA

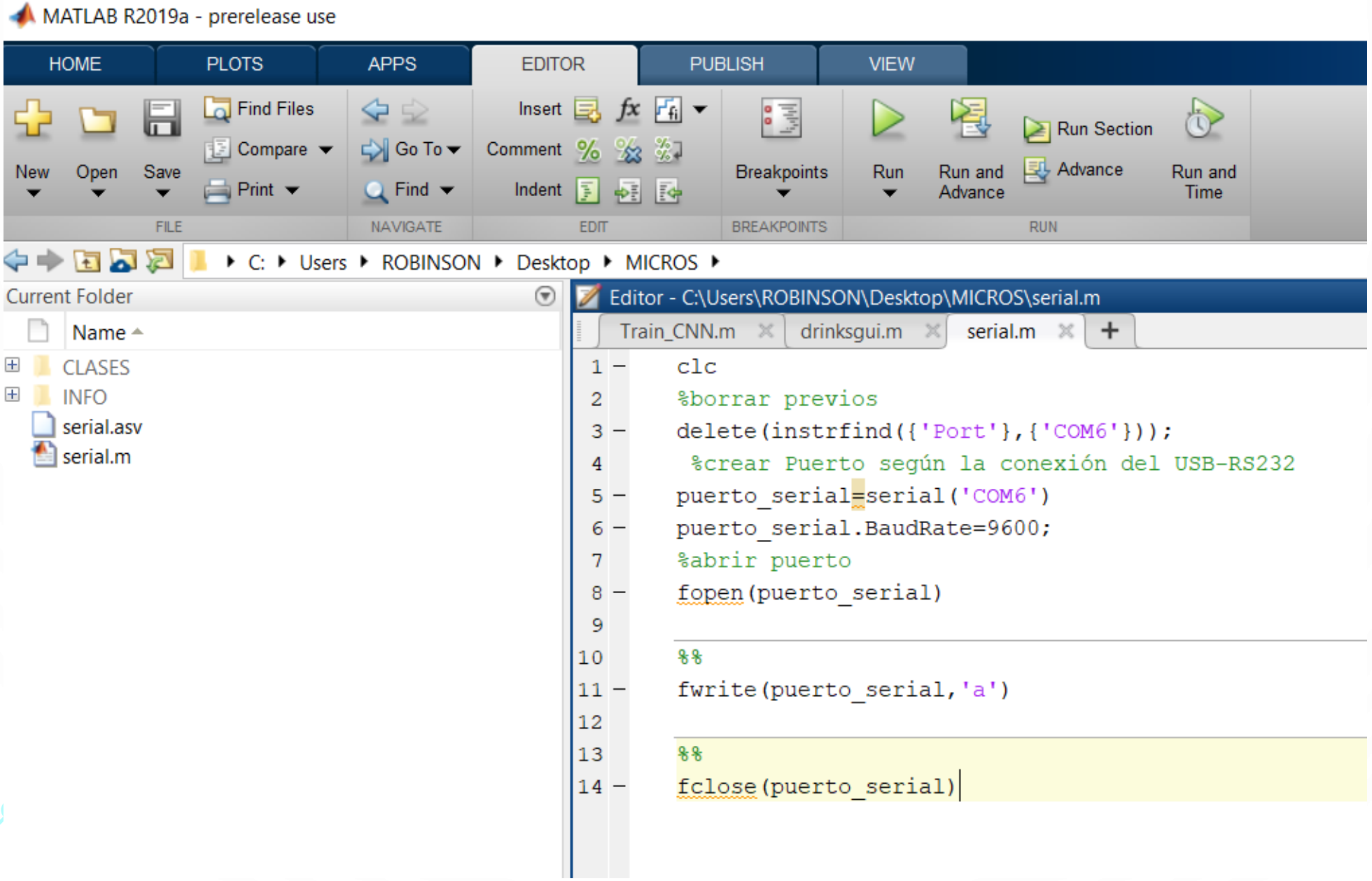
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	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

```c
#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 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
  }
  }
```

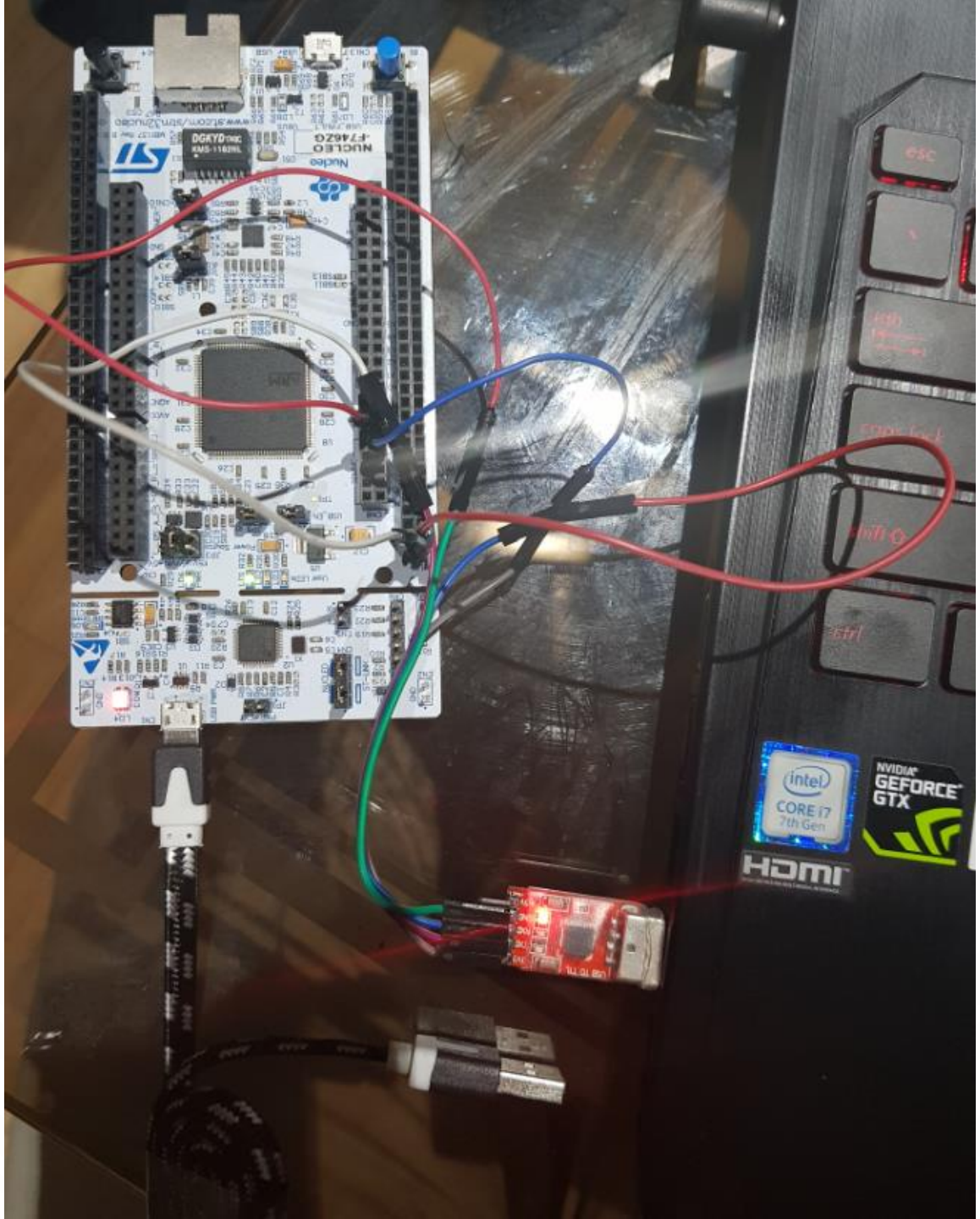


CONFIGURACIÓN DE MATLAB

INGENIERÍA MECATRÓNICA UMNG



UNIVERSIDAD MILITAR NUEVA GRANADA

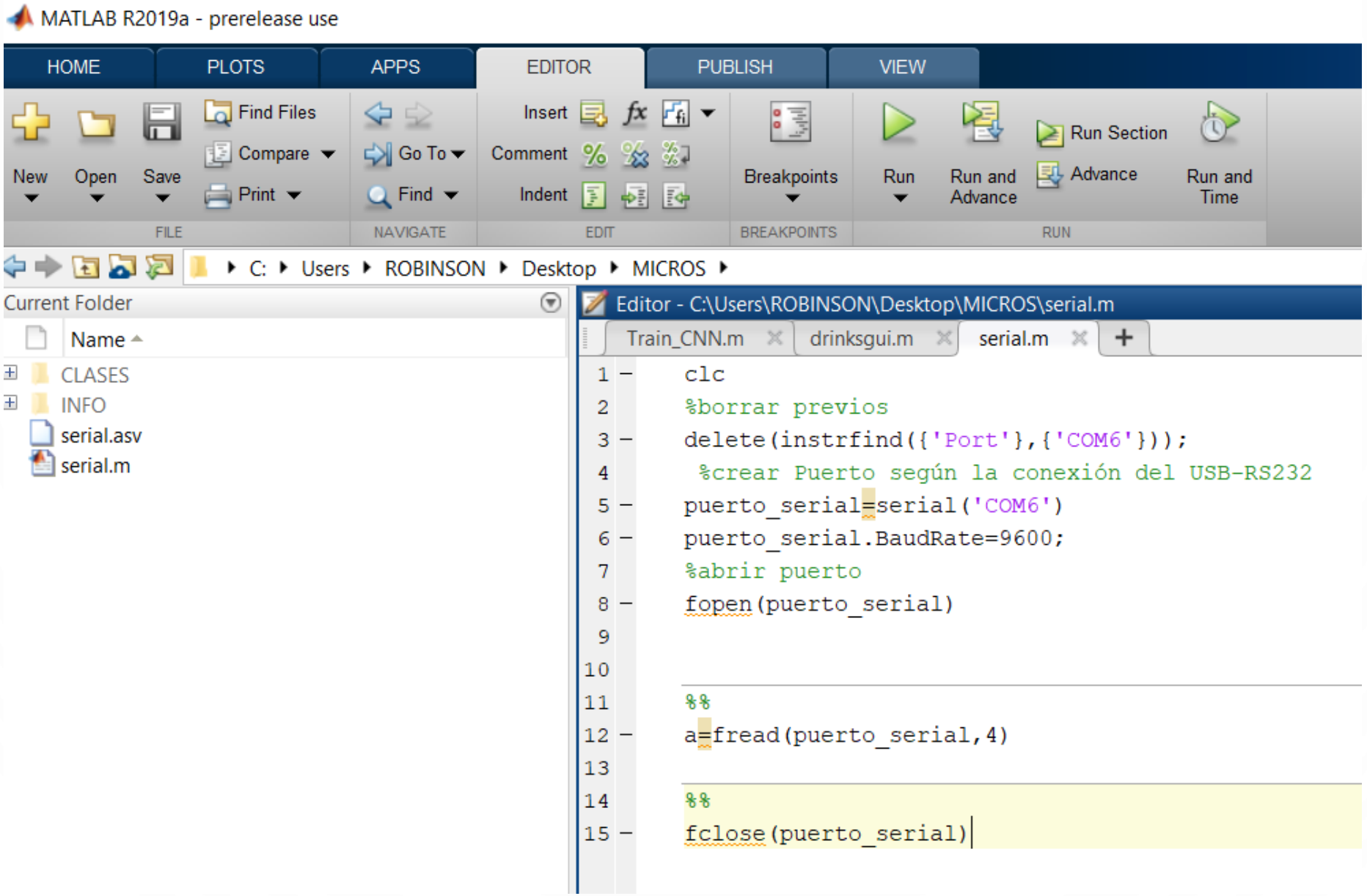


INGENIERÍA MECATRÓNICA UMNG

UNIVERSIDAD MILITAR NUEVA GRANADA

```c
#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 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;}
}
```

CONFIGURACIÓN DE MATLAB



INGENIERÍA MECATRÓNICA UMNG

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 LCD.

UNIVERSIDAD MILITAR NUEVA GRANADA