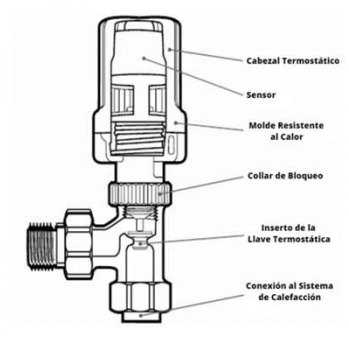
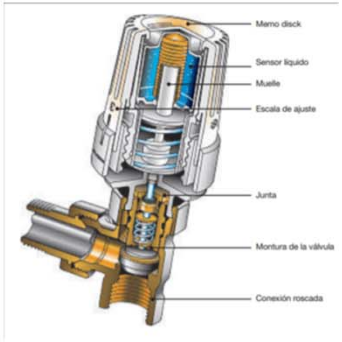
Sistemas Basados en Microprocesador

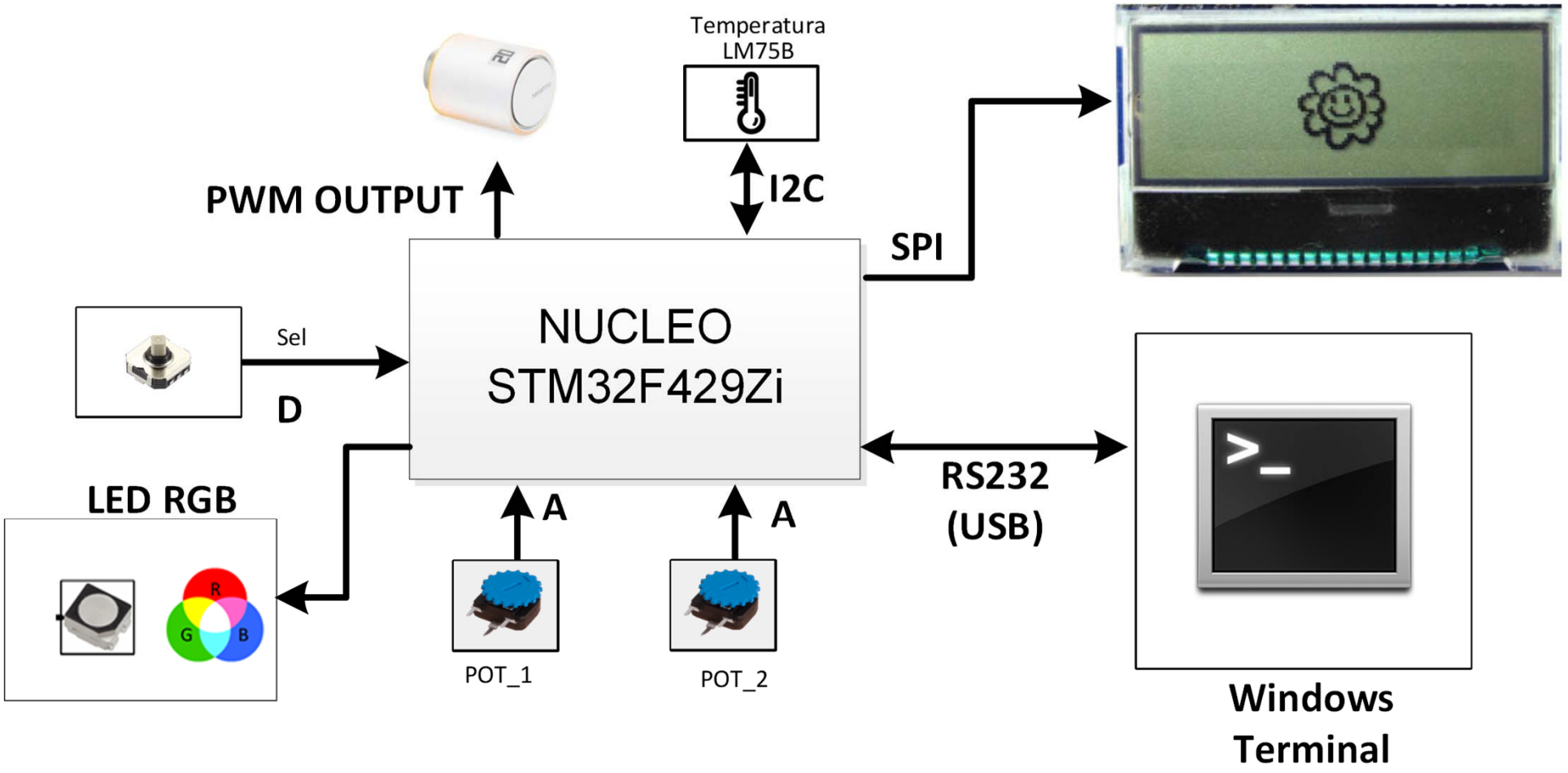
Bloque 3. DISEÑO

Objetivo:
Controlador de una válvula termostática





Esquema general del sistema

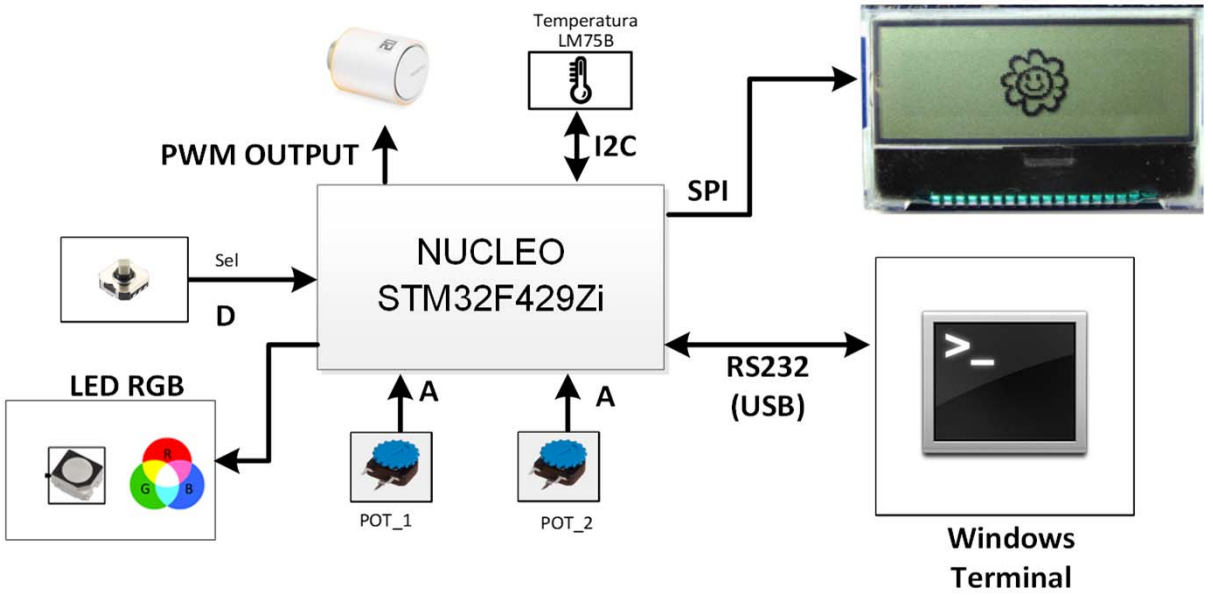


Elementos del sistema y comunicación con el micro

1. Tarjeta NUCLEO STM32F429Zi

3. Mbed app board:
   - Joystick 5 gestos → GPIO
   - Actuador TRV → PWM (Timer)
   - Sensor temperatura → **Bus I2C**
   - LCD → SPI
   - Modo test → **ADC**
   - LED RGB → **GPIO/PWM**

4. PC → **UART (USB)**



Elementos no utilizados hasta ahora:

- **Interfaz serie (UART)**
- **Bus I2C**
- **Convertidor A/D**

Temporización

	10	13/11/2023	B3 (T)		B2			**10**	17/11/2023
	11	20/11/2023	B3		B3			**11**	24/11/2023
dic	**12**	27/11/2023	Examen B2	**E1**	B3			**12**	01/12/2023
	13	04/12/2023	B3	**E2**	06-dic	07-dic	08-dic	**13**	08/12/2023
	14	11/12/2023	B3	**E3**	B3			**14**	15/12/2023
	15	18/12/2023	B3	**E4**	Examen B3			**15**	22/12/2023

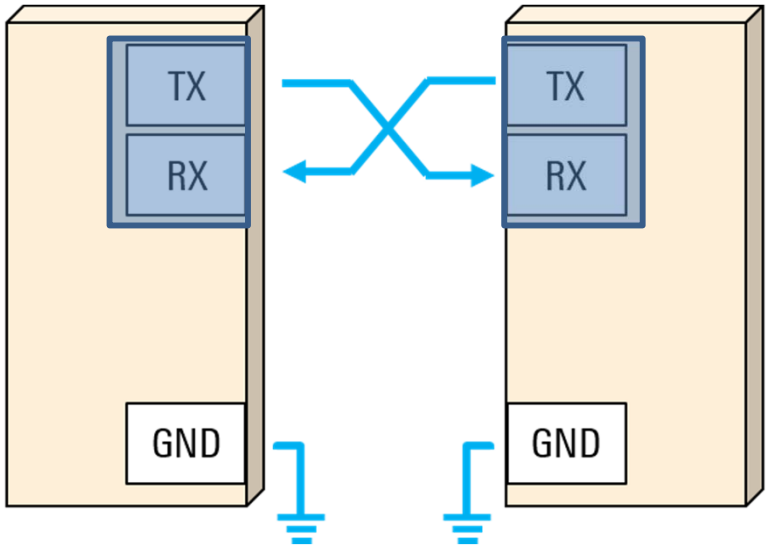
	Fecha	Módulos
E1	Martes, 28/11/2023, 21:00	JOYSTICK, HORA
E2	Martes, 5/12/2023, 21:00	TEMPERATURA, LCD
E3	Martes, 12/12/2023, 21:00	COM-PC, PWM OUTPUT, POT_1/POT_2
E4	Martes, 19/12/2023, 21:00	Proyecto FINAL y memoria técnica

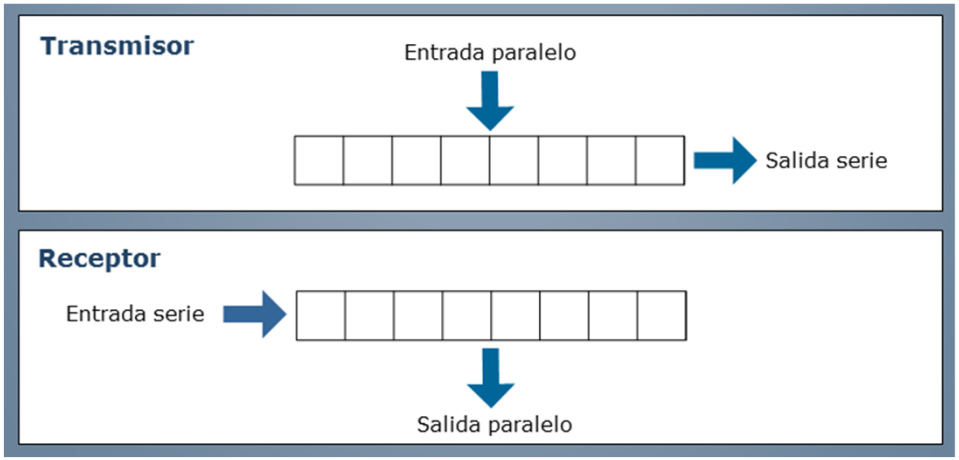
Sistemas Basados en Microprocesador

Bloque 3. DISEÑO

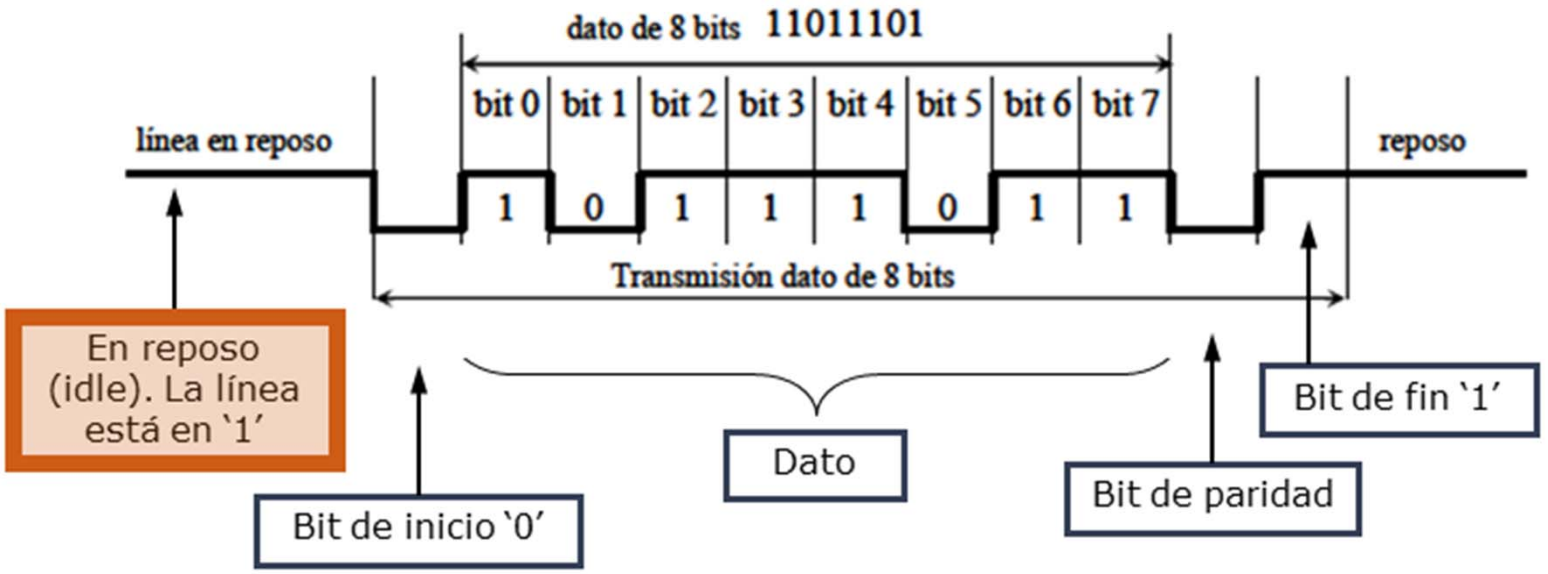
UART

Comunicación serie asíncrona. Concepto





UART (**universal asynchronous receiver / transmitte**r, por sus siglas en inglés), define un protocolo o un conjunto de normas para el intercambio de **datos en serie** entre dos dispositivos. UART es sumamente simple y utiliza solo dos hilos entre el transmisor y el receptor para transmitir y recibir en ambas direcciones. Ambos extremos tienen una conexión a masa. La comunicación en UART puede ser **simplex** (los datos se envían en una sola dirección), **semidúplex** (cada extremo se comunica, pero solo uno al mismo tiempo), o **dúplex completo** (ambos extremos pueden transmitir simultáneamente). En UART, los datos se transmiten en forma de tramas.

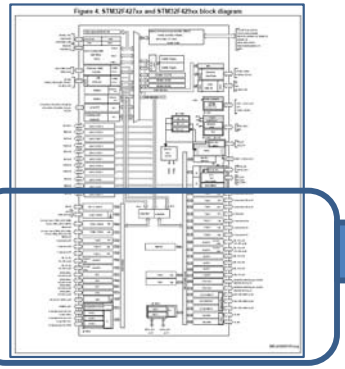


- t_b = tiempo de bit
- *BR* = *Baud Rate* = $1/t_b$ (baudios, bps)
- S: bit de arranque (0, *Start bit*)
- B_i: 5, 6, 7 u 8 bits de datos. *LSB first*
- P: bit de paridad (*Parity*).
- T: bit de parada (1, *Stop bit*)
  - Puede durar 1, 1.5 ó 2 t_b

9600–8N1

- El periférico encargado de las comunicaciones serie asíncronas RS-xxx suele denominarse UART (o USART) (*Universal Synchronous/Asynchronous Receiver-Transmitter*)
  - Genera: TxD
  - Recibe: RxD
- Permite configurar los parámetros de la conexión (baud rate, number of bits, parity, stop bits)
- Puede generar interrupciones (dato recibido, dato enviado, error de paridad, etc.)
- Suelen disponer de *buffers* FIFO y de DMA

Comunicación serie asíncrona. STM32F429



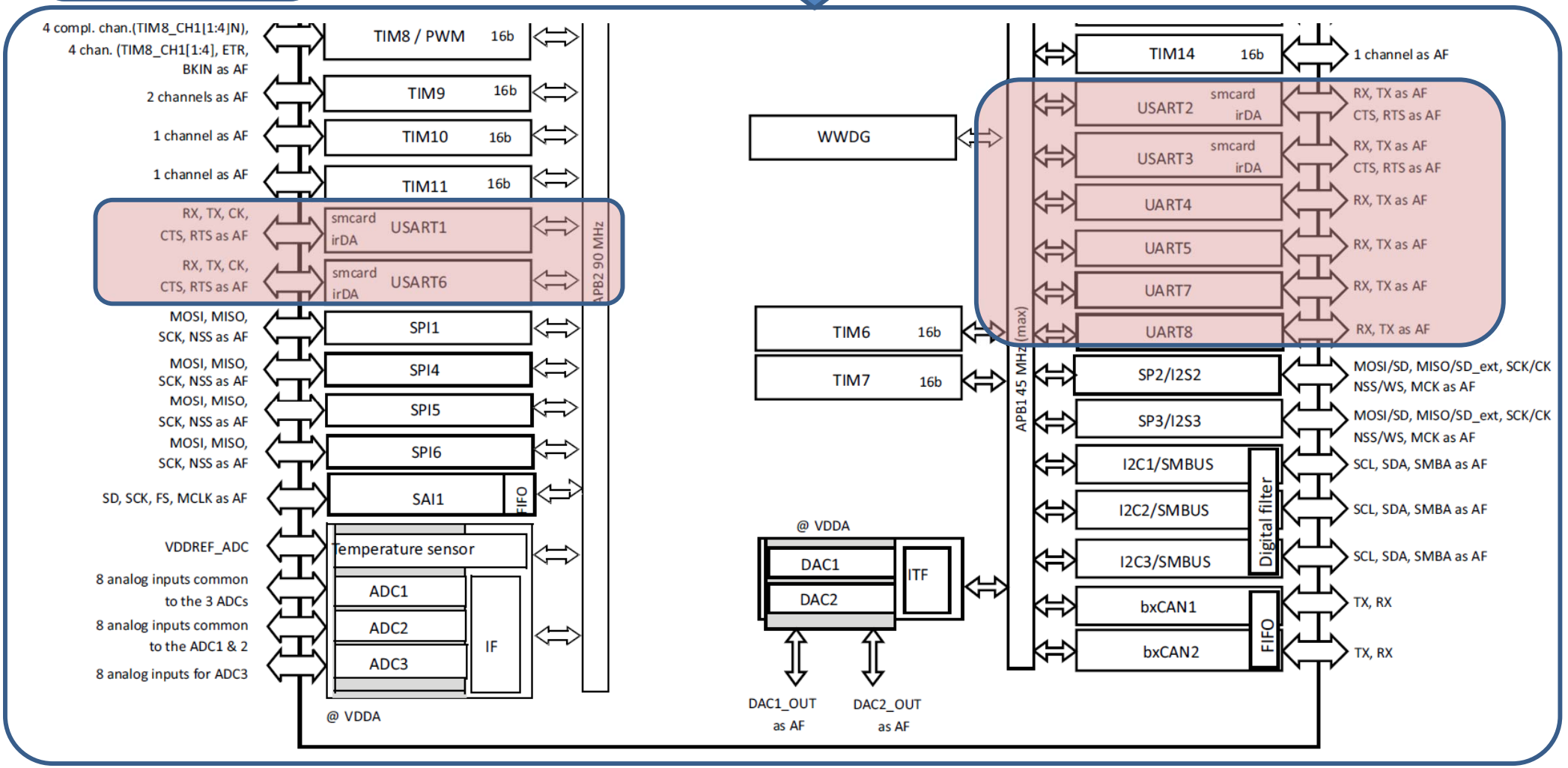
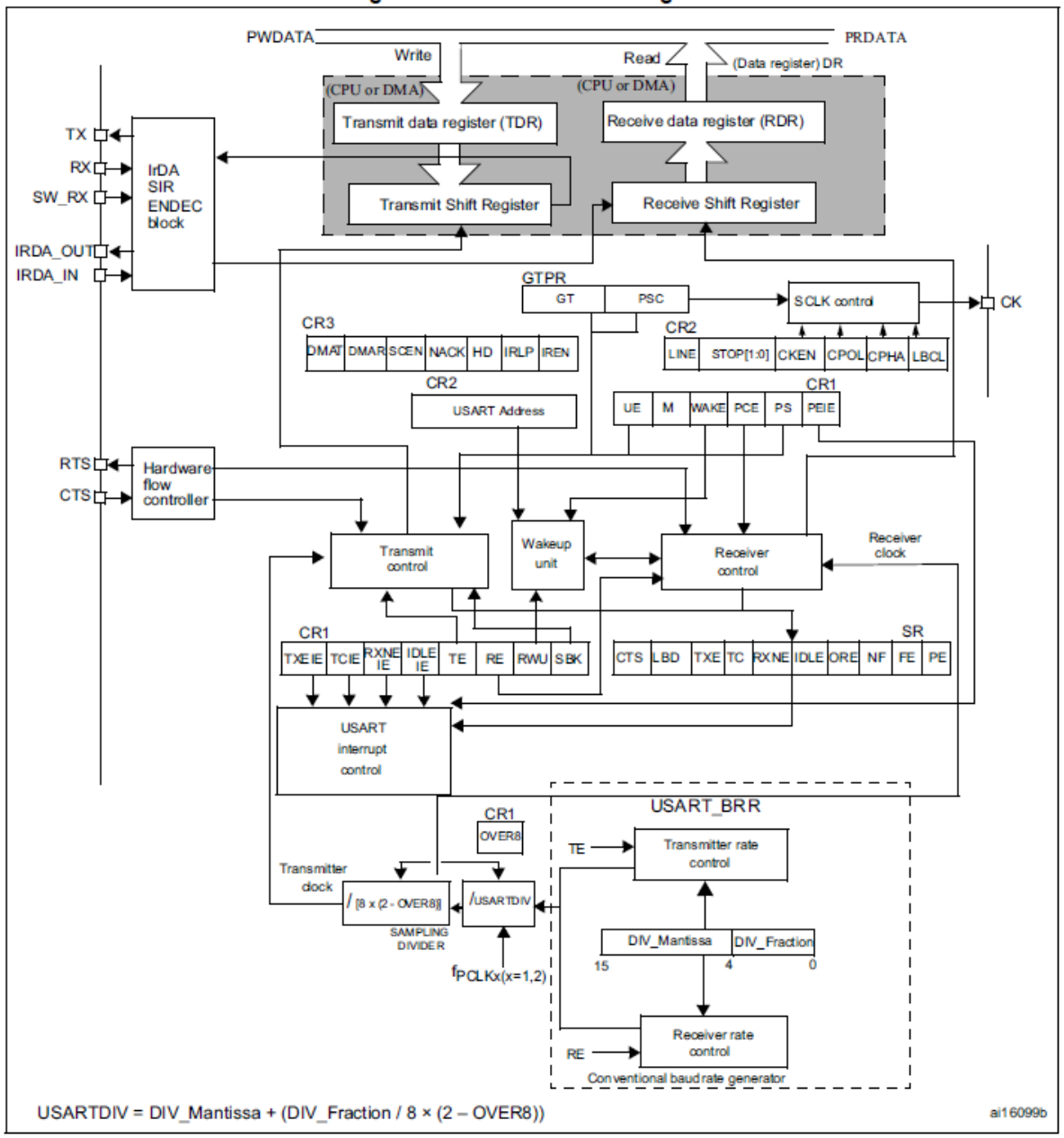


Figure 296. USART block diagram



➢ Manejo de la UART desde CMSIS Driver

https://www.keil.com/pack/doc/CMSIS/Driver/html/group_usart_interface_gr.html

➢ Uso de 1 UART/USART:

➢ PC → USART3 (USB Virtual COM Port)

6.9 USART communication

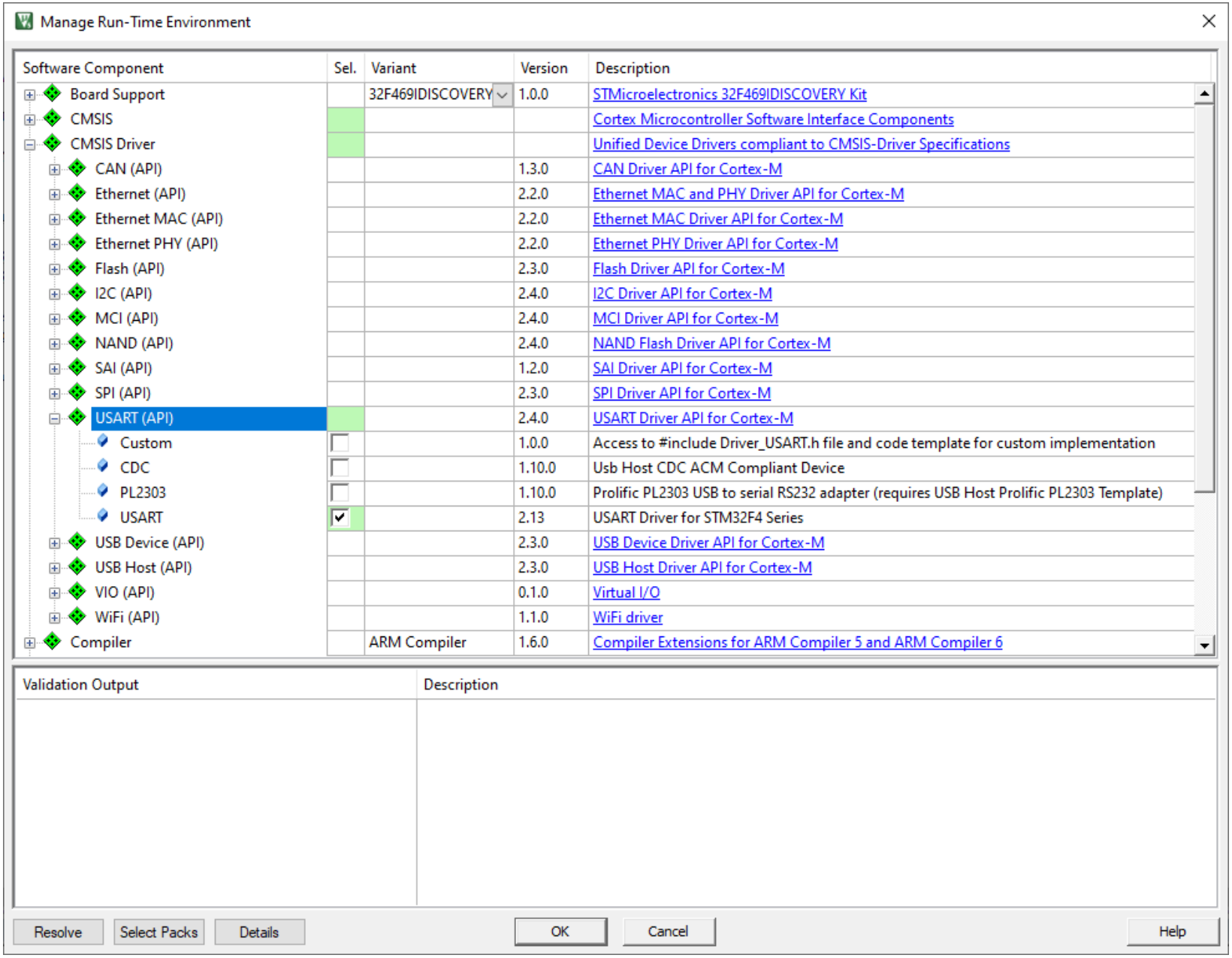
The USART3 interface available on PD8 and PD9 of the STM32 can be connected either to ST-LINK or to ST morpho connector. The choice is changed by setting the related solder bridges. By default the USART3 communication between the target STM32 and the ST-LINK is enabled, to support the virtual COM port for the mbed (SB5 and SB6 ON).

Table 9. USART3 pins

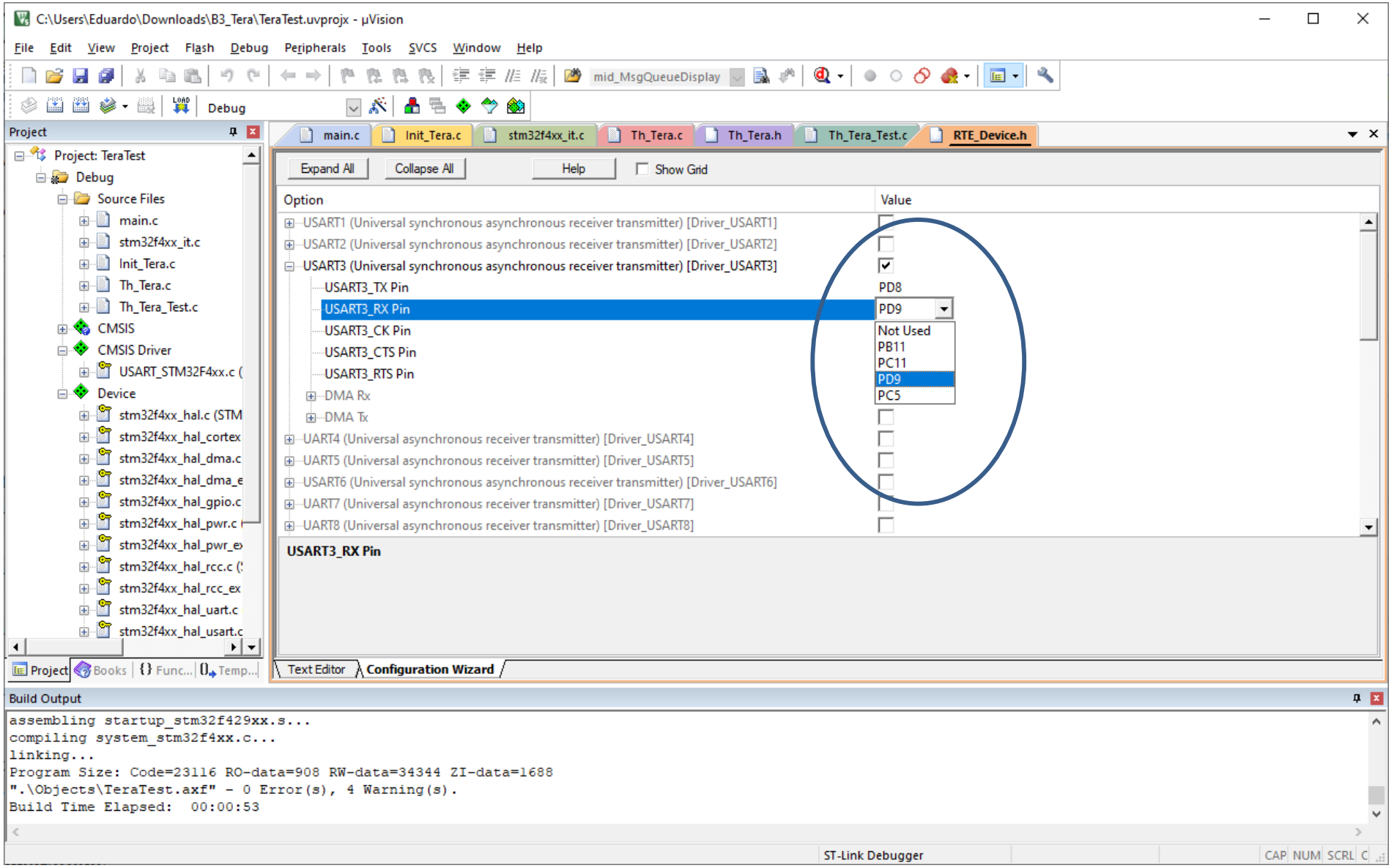
Pin name	Function	Virtual COM port (default configuration)	ST morpho connection
PD8	USART3 TX	SB5 ON and SB7 OFF	SB5 OFF and SB7 ON
PD9	USART3 RX	SB6 ON and SB4 OFF	SB6 OFF and SB4 ON

UM1974-NUCLEO-F429ZI-USER MANUAL

Configuración de UART en Keil. Run-Time Environment



Configuración de UART en Keil. RTE_Device.h



Manejo de la UART por callback

```c
#include "Driver_USART.h"
#include "cmsis_os.h" /* ARM::CMSIS:RTOS:Keil RTX */
#include <stdio.h>
#include <string.h>
void myUART_Thread(void const *argument);
osThreadId tid_myUART_Thread;
/* USART Driver */
extern ARM_DRIVER_USART Driver_USART3;

void myUSART_callback(uint32_t event)
{
uint32_t mask;
mask = ARM_USART_EVENT_RECEIVE_COMPLETE |
ARM_USART_EVENT_TRANSFER_COMPLETE |
ARM_USART_EVENT_SEND_COMPLETE |
ARM_USART_EVENT_TX_COMPLETE ;
if (event & mask) {
/* Success: Wakeup Thread */
osSignalSet(tid_myUART_Thread, 0x01);
}
if (event & ARM_USART_EVENT_RX_TIMEOUT) {
__breakpoint(0); /* Error: Call debugger or replace
with custom error handling */
}
if (event & (ARM_USART_EVENT_RX_OVERFLOW |
ARM_USART_EVENT_TX_UNDERFLOW)) {
__breakpoint(0); /* Error: Call debugger or replace
with custom error handling */
}
}
/* CMSIS-RTOS Thread - UART command thread */
```

osThreadFlagsSet

http://www.keil.com/pack/doc/CMSIS/Driver/html/group__usart__interface__gr.html

```c
void myUART_Thread(const void* args)
{
static ARM_DRIVER_USART * USARTdrv = &Driver_USART3;
ARM_DRIVER_VERSION version;
ARM_USART_CAPABILITIES drv_capabilities;
char cmd;
#ifdef DEBUG
version = USARTdrv->GetVersion();
if (version.api < 0x200) /* requires at minimum API version 2.00 or
higher */
{ /* error handling */
return;
}
drv_capabilities = USARTdrv->GetCapabilities();
if (drv_capabilities.event_tx_complete == 0)
{ /* error handling */
return;
}
#endif
/*Initialize the USART driver */
USARTdrv->Initialize(myUSART_callback);
/*Power up the USART peripheral */
USARTdrv->PowerControl(ARM_POWER_FULL);
/*Configure the USART to 4800 Bits/sec */
USARTdrv->Control(ARM_USART_MODE_ASYNCHRONOUS |
ARM_USART_DATA_BITS_8 |
ARM_USART_PARITY_NONE |
ARM_USART_STOP_BITS_1 |
ARM_USART_FLOW_CONTROL_NONE, 4800);
/* Enable Receiver and Transmitter lines */
USARTdrv->Control (ARM_USART_CONTROL_TX, 1);
USARTdrv->Control (ARM_USART_CONTROL_RX, 1);
USARTdrv->Send("\nPress Enter to receive a message", 34);
osSignalWait(0x01, osWaitForever);
while (1)
{
USARTdrv->Receive(&cmd, 1); /* Get byte from UART */
osSignalWait(0x01, osWaitForever);
if (cmd == 13) /* CR, send greeting */
{
USARTdrv->Send("\nHello World!", 12);
osSignalWait(0x01, osWaitForever);
}
}
}
```

osThreadFlagsWait

Conexión tarjeta NUCLEO con PC (TeraTerm)

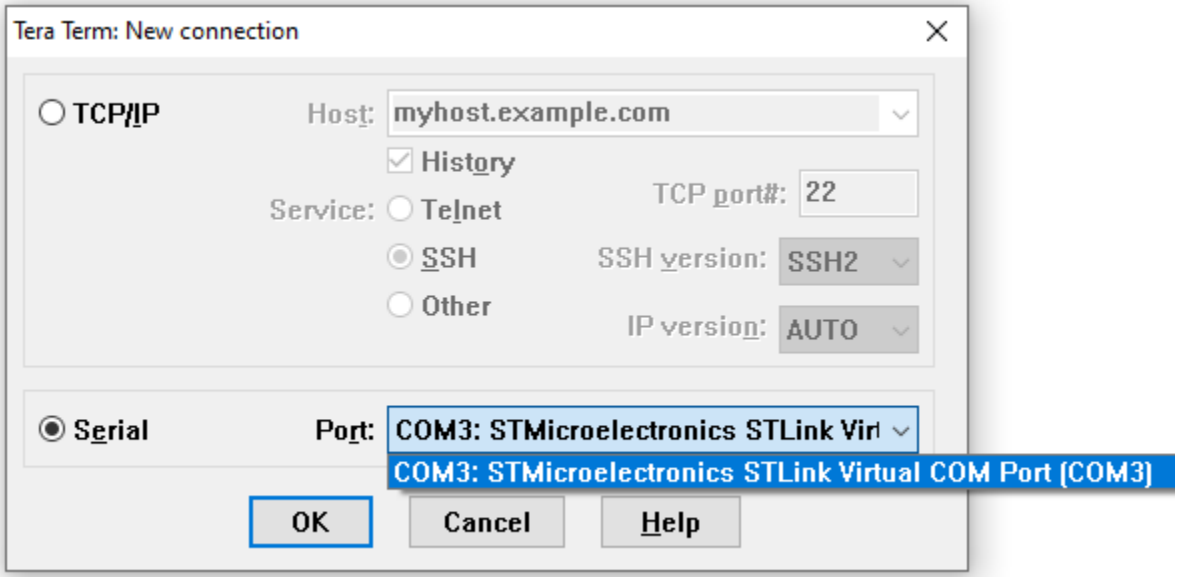
Tera Term (Windows)

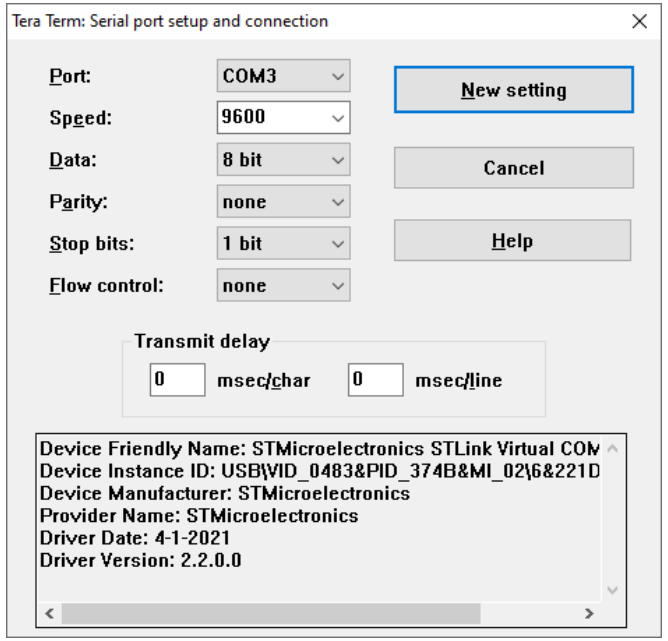
Tera Term is one of the more popular Windows terminal programs. It's been around for years, it's open source, and it's simple to use. For Windows users, it's one of the best options out there.

You can download a copy from here. Once you have Tera Term installed, open up it up, and let's poke around.

Making a Connection

You should initially be presented with a "TeraTerm: New connection" pop-up within the program. Here, you can select which serial port you'd like to open up. Select the "Serial" radio button. Then select your port from the drop-down menu. (If this window doesn't open when you start TeraTerm, you can get here by going to ****File > New connection..."**.)





That'll open up the port. TeraTerm defaults to setting the baud rate at 9600 bps (8-N-1). If you need to adjust the serial settings, go up to **Setup > Serial Port**. You'll see a window pop up with a lot of familiar looking serial port settings. Adjust what you need to and hit "OK".

Sistemas Basados en Microprocesador

Bloque 3. DISEÑO

I2C

Características del bus I2C

– Bus de comunicación síncrono
  - La comunicación es controlada por una señal de reloj común

– Bus formado por 2 hilos
  - SDA (***S**erial **DA**ta Line*): datos
  - SCL (***S**erial **CL**ock line*): reloj
  - También es necesaria una referencia común de masa

– Velocidad de transmisión
  - *Standard*: hasta 100 Kbits/s
  - *Fast*: hasta 400 Kbits/s
  - *High-speed*: hasta 3,4 Mbits/s

– Cada dispositivo del bus tiene una dirección única

– Distancia y número de dispositivos
  - Limitado por la capacidad del bus (inferior a 400pF). Normalmente 2 o 3 metros

– Protocolo de acceso al bus:
  - Maestro –esclavo
  - I_2C soporta protocolo multimaestro

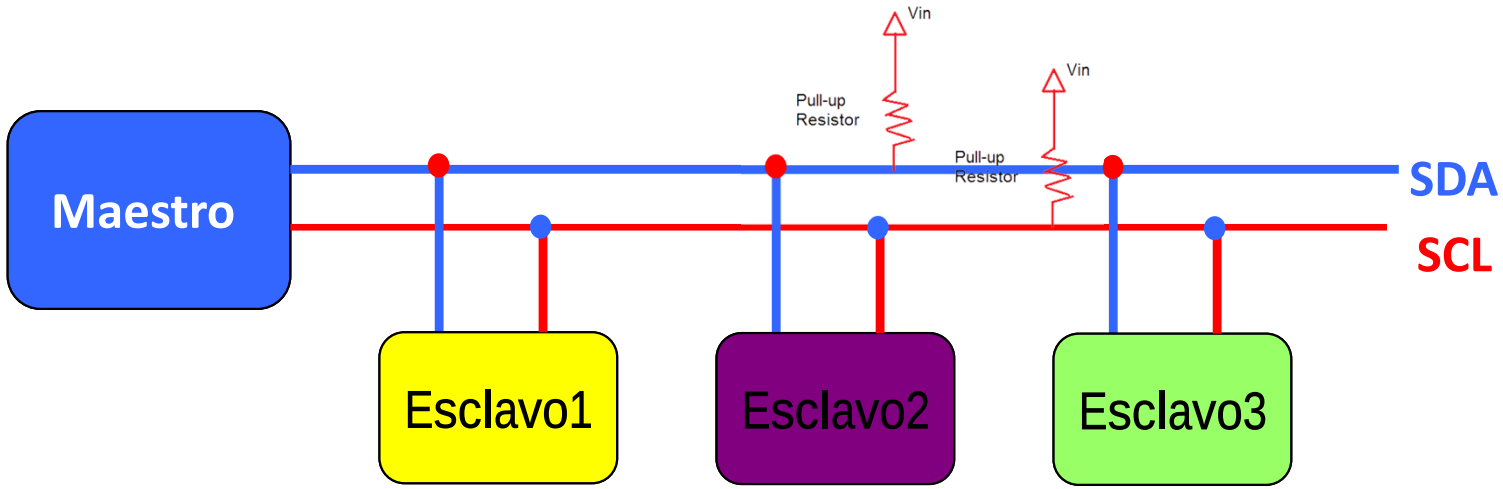
Comunicación serie, utilizando un conductor para manejar el timming (SCL) (pulsos de reloj) y otro para intercambiar datos (SDA), que transportan información entre los dispositivos conectados al bus.

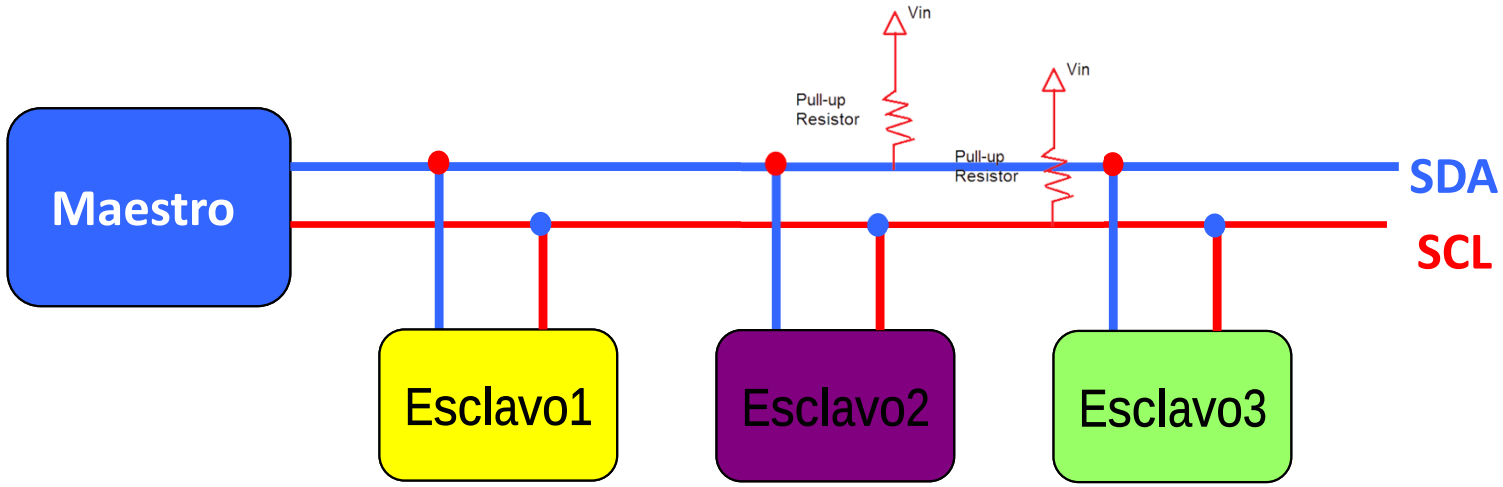
Las líneas SDA (Serial Data) y SCL (Serial Clock) están conectadas a la fuente de alimentación a través de las resistencias de pull-up. Cuando el bus está libre, ambas líneas están en nivel alto.

Los dispositivo puede ser considerado como Maestro (Master) o esclavo (Slave).

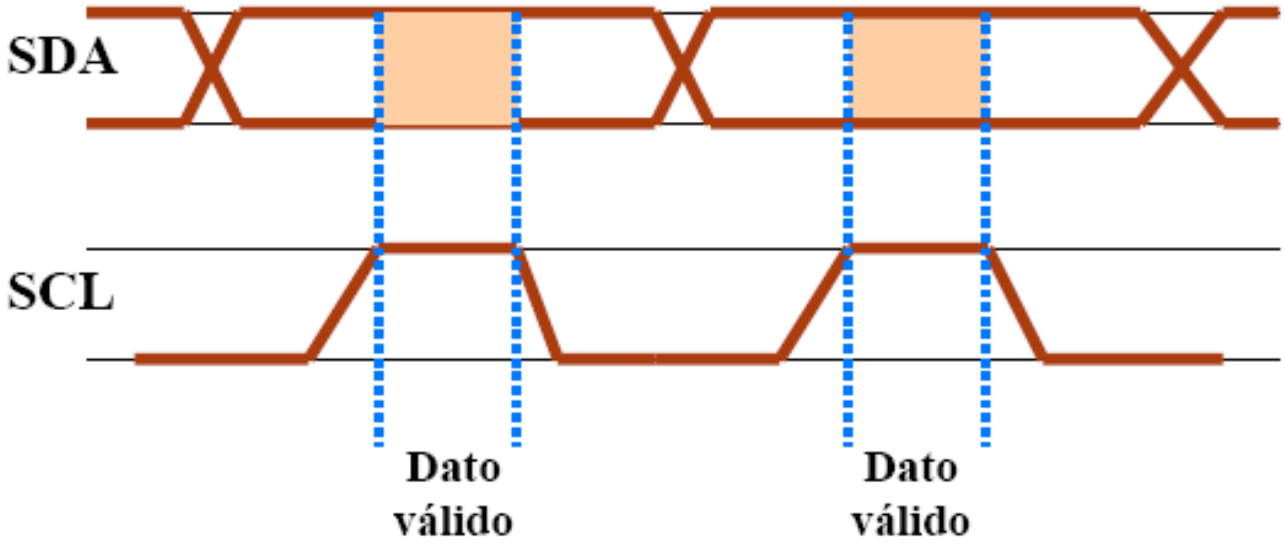
El Maestro es el dispositivo que inicia la transferencia en el bus y genera la señal de Clock.

El Slave (esclavo) es el dispositivo direccionado.

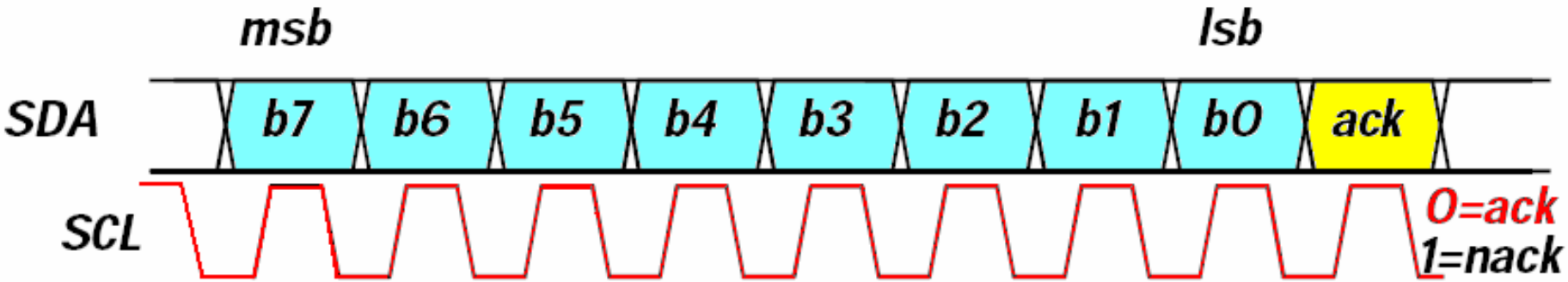




– Los bits de datos van por SDA
– Por cada bit de información es necesario un pulso de SCL
– Los datos sólo pueden cambiar cuando SCL está a nivel bajo



- Los datos transitan durante el nivel bajo del reloj
- El dato es recibido en el flanco de bajada del reloj
- El bit más significativo se envía primero
- El nodo que recibe debe generar un acknowledge (nivel bajo en SDA) después de completado el byte
- El nodo maestro siempre genera el reloj



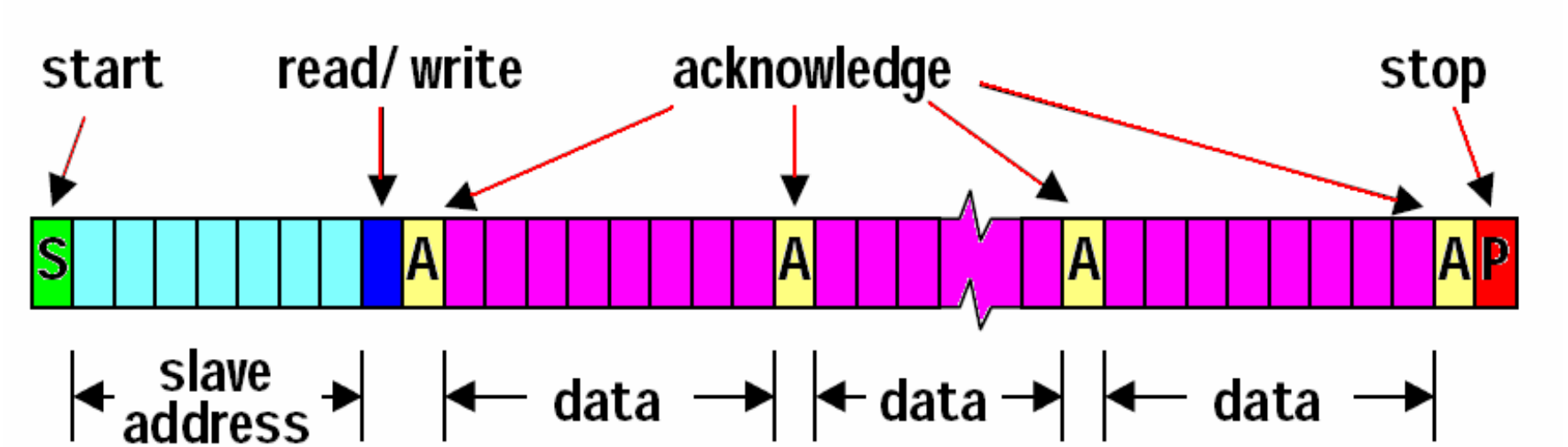
Trasferencia de datos:

➢ **Los datos y direcciones que se transmiten por SDA son de 8 bits.**

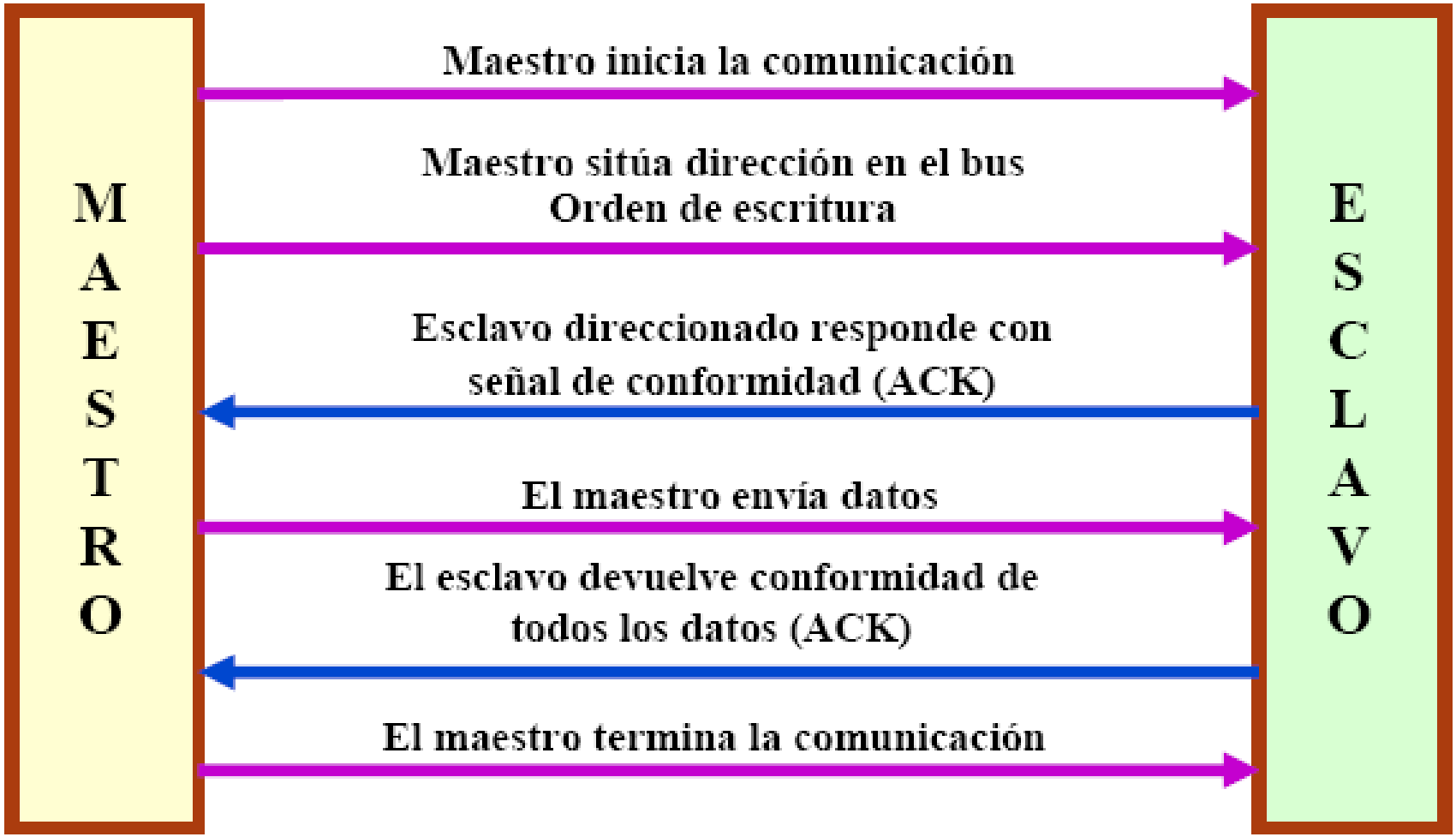
➢Tras cada bloque debe recibirse una señal de reconocimiento

Formato del Mensaje

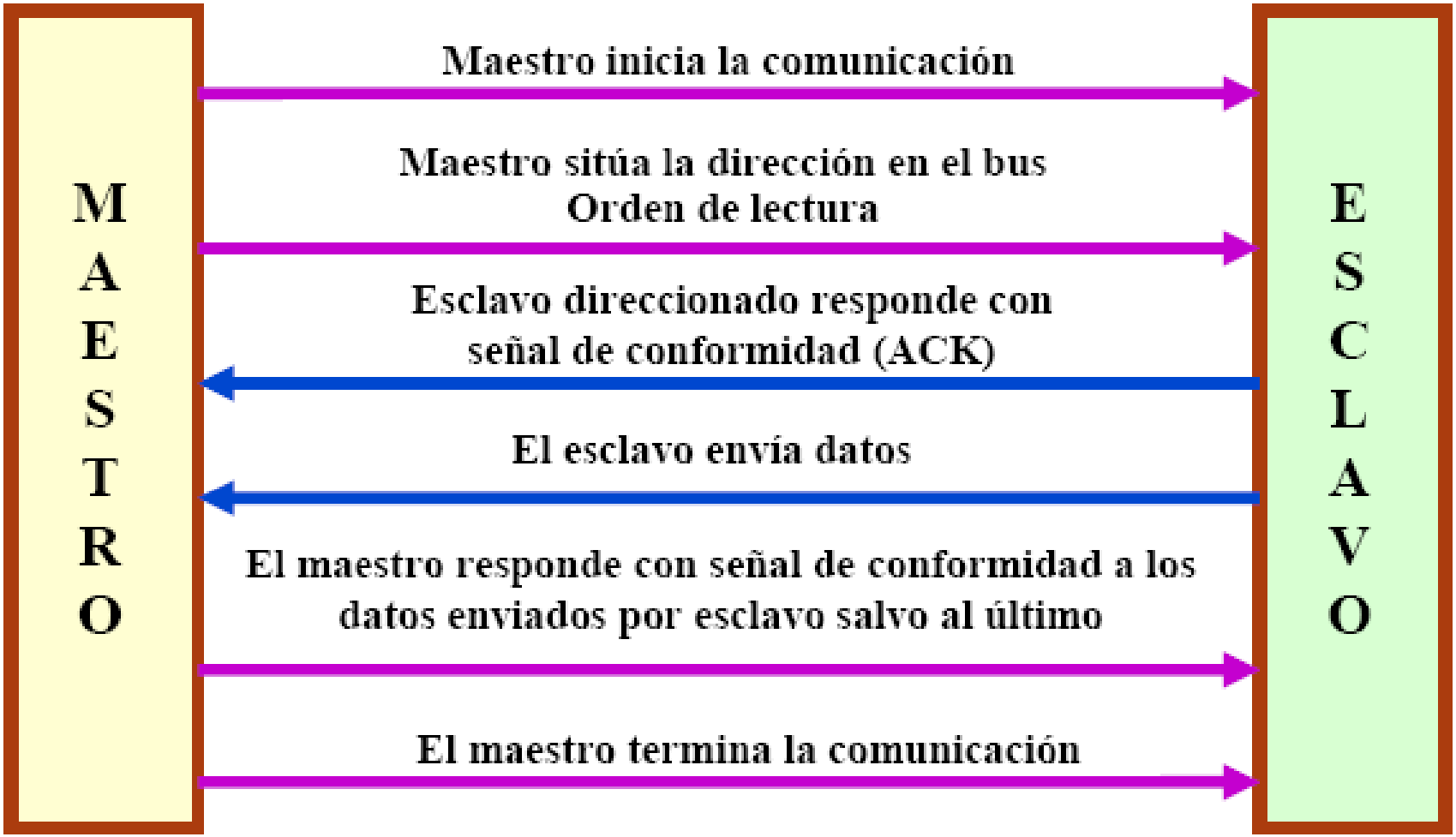
- Un protocolo orientado a BIT
- Handshaking
- Bidireccional

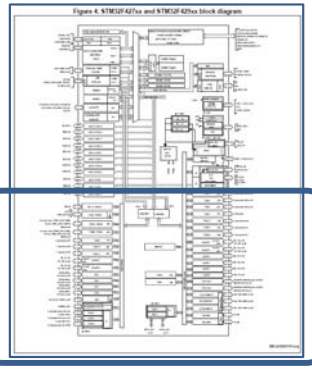


Maestro envía datos a un esclavo



Maestro lee datos de un esclavo



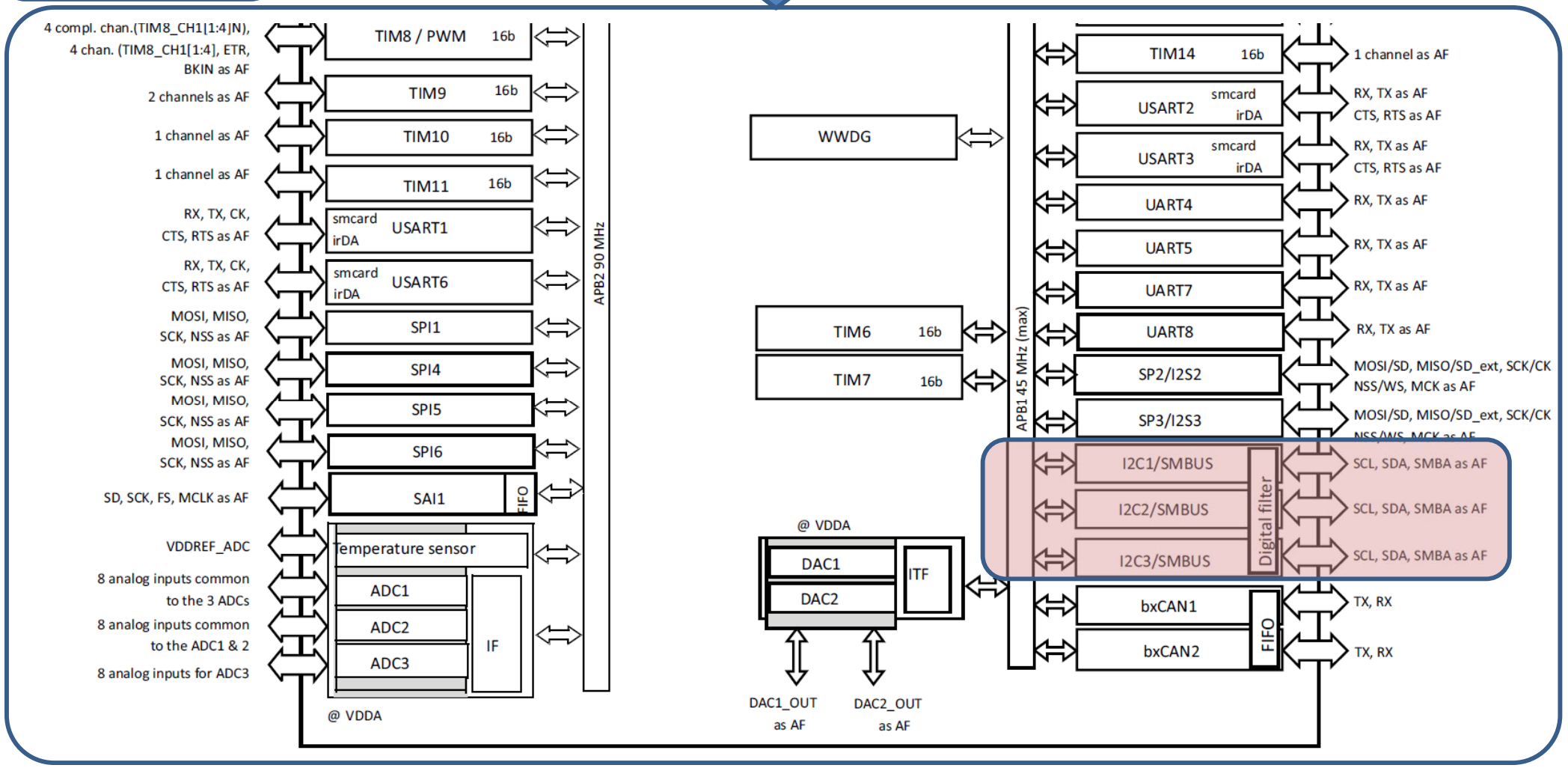


➢ Manejo de I2C desde CMSIS Driver

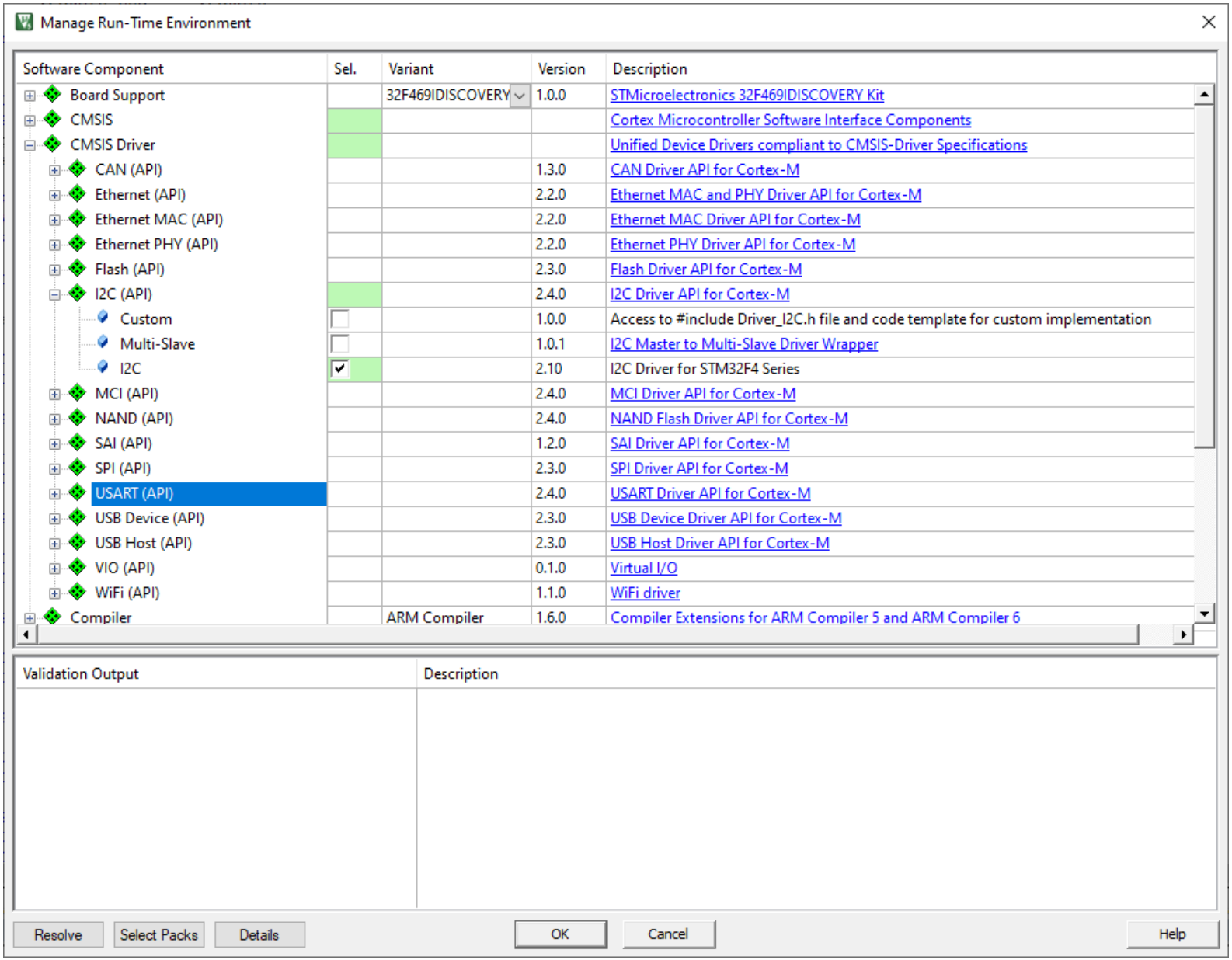
https://www.keil.com/pack/doc/CMSIS/Driver/html/group__i2c__interface__gr.html

Uso de 2 buses I2C:

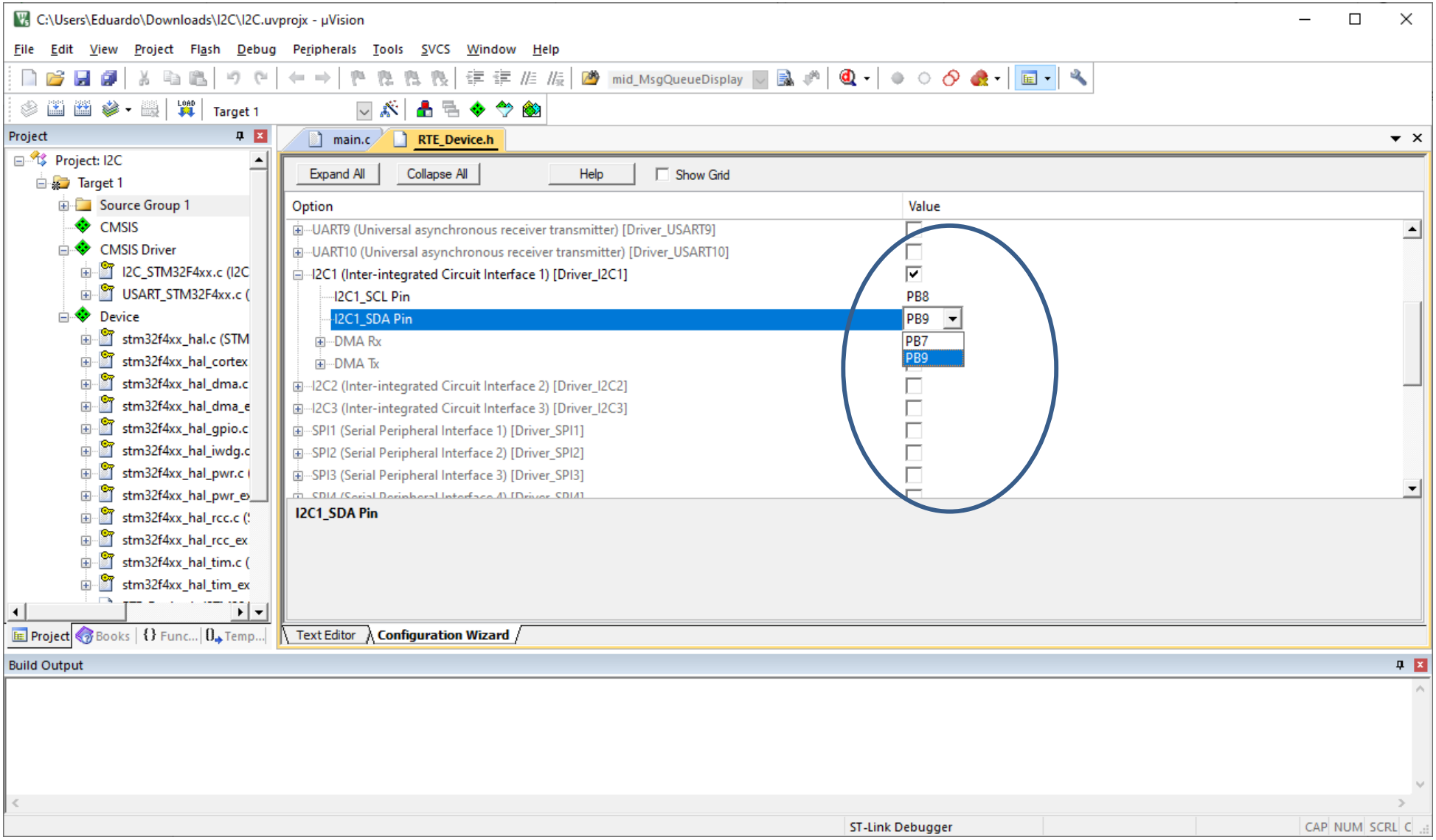
I2C1→Sensor Temp.
I2C2→Sintonizador FM



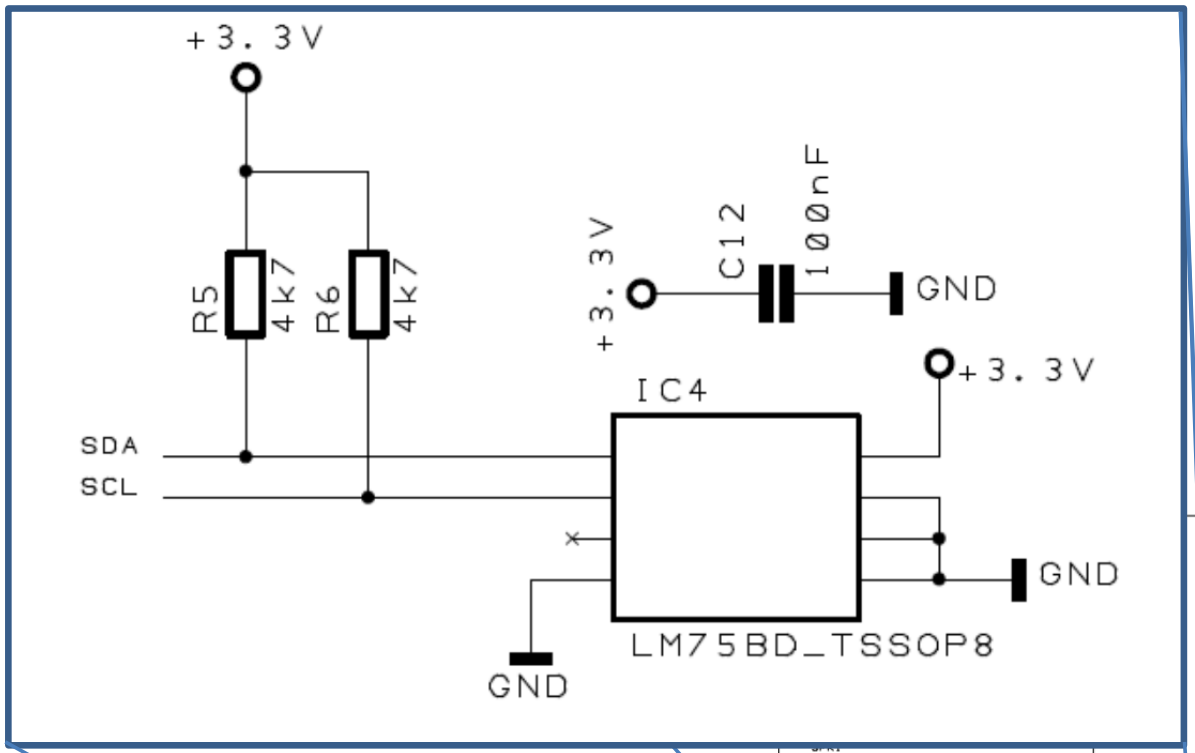
Configuración de I2C en Keil. Run-Time Environment



Configuración de I2C en Keil. RTE_Device.h



Sensor de temperatura LM75B



https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

LM75B

Digital temperature sensor and thermal watchdog

Rev. 6.1 — 6 February 2015 Product data sheet

7.3 Slave address

The LM75B slave address on the I^2C-bus is partially defined by the logic applied to the device address pins A2, A1 and A0. Each of them is typically connected either to GND for logic 0, or to V_{CC} for logic 1. These pins represent the three LSB bits of the device 7-bit address. The other four MSB bits of the address data are preset to '1001' by hard wiring inside the LM75B. Table 4 shows the device's complete address and indicates that up to 8 devices can be connected to the same bus without address conflict. Because the input pins, SCL, SDA and A2 to A0, are not internally biased, it is important that they should not be left floating in any application.

Table 4. Address table

1 = HIGH; 0 = LOW.

MSB						LSB
1	0	0	1	A2	A1	A0

LM75B

Digital temperature sensor and thermal watchdog

Rev. 6.1 — 6 February 2015 Product data sheet

7.4 Register list

The LM75B contains four data registers beside the pointer register as listed in Table 5. The pointer value, read/write capability and default content at power-up of the registers are also shown in Table 5.

Table 5. Register table

Register name	Pointer value	R/W	POR state	Description
Conf	01h	R/W	00h	Configuration register: contains a single 8-bit data byte; to set the device operating condition; default = 0.
Temp	00h	read only	n/a	Temperature register: contains two 8-bit data bytes; to store the measured Temp data.
Tos	03h	R/W	5000h	Overtemperature shutdown threshold register: contains two 8-bit data bytes; to store the overtemperature shutdown $T_{th(ots)}$ limit; default = 80 °C.
Thyst	02h	R/W	4B00h	Hysteresis register: contains two 8-bit data bytes; to store the hysteresis T_{hys} limit; default = 75 °C.

7.4.3 Temperature register

The Temperature register (Temp) holds the digital result of temperature measurement or monitor at the end of each analog-to-digital conversion. This register is read-only and contains two 8-bit data bytes consisting of one Most Significant Byte (MSByte) and one Least Significant Byte (LSByte). However, only 11 bits of those two bytes are used to store the Temp data in two's complement format with the resolution of 0.125 °C. Table 9 shows the bit arrangement of the Temp data in the data bytes.

Table 9. Temp register

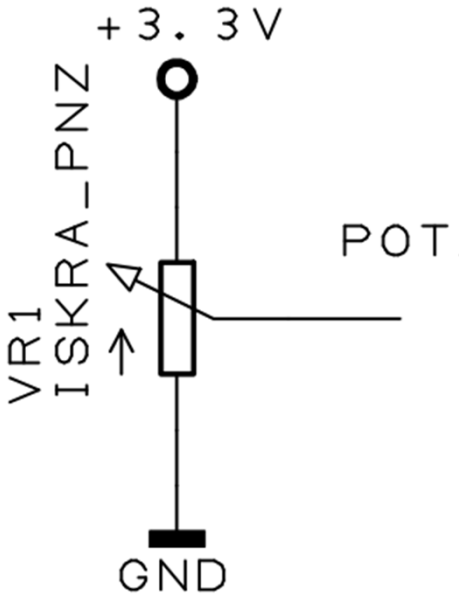
MSByte								LSByte							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X

When reading register Temp, all 16 bits of the two data bytes (MSByte and LSByte) are provided to the bus and must be all collected by the controller to complete the bus operation. However, only the 11 most significant bits should be used, and the 5 least significant bits of the LSByte are zero and should be ignored. One of the ways to calculate the Temp value in °C from the 11-bit Temp data is:

1. If the Temp data MSByte bit D10 = 0, then the temperature is positive and Temp value (°C) = +(Temp data) × 0.125 °C.
2. If the Temp data MSByte bit D10 = 1, then the temperature is negative and Temp value (°C) = –(two's complement of Temp data) × 0.125 °C.

***float* variable**

Sistemas Basados en Microprocesador

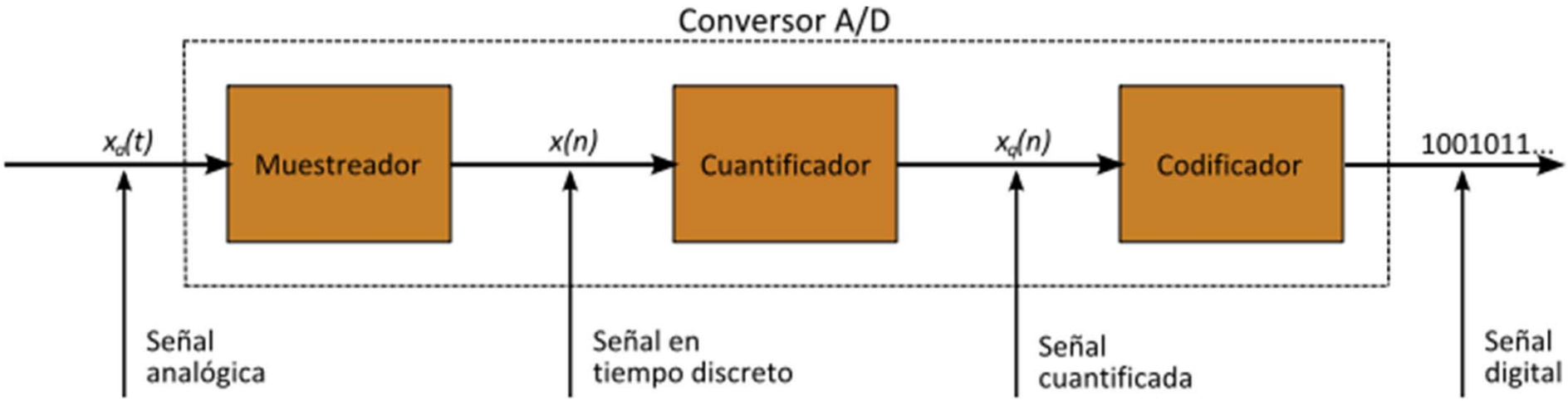


Bloque 3. DISEÑO

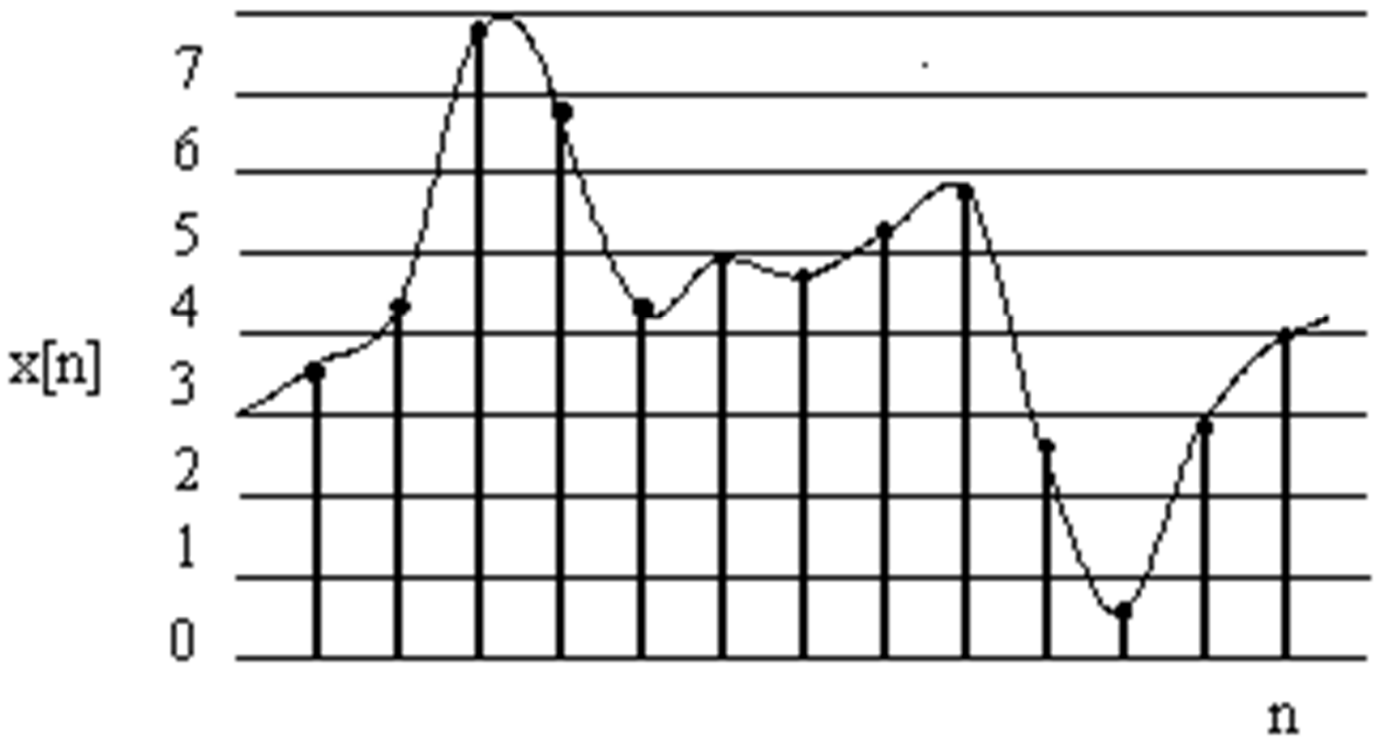
ADC

La conversion de una señal analógica en una digital implica cuatro pasos:

- **Muestreo**: toma (habitualmente periódica) de muestras de la señal analógica
- **Retención**: las muestras tomadas han de ser retenidas (retención) por un circuito de retención (hold) el tiempo suficiente para permitir evaluar su nivel (cuantificación)
- **Cuantificación**: se mide el nivel de voltaje de cada una de las muestras. Consiste en asignar un margen de valor de una señal analizada a un único nivel de salida
- **Codificación**: traducción de los valores obtenidos durante la cuantificación a código binario

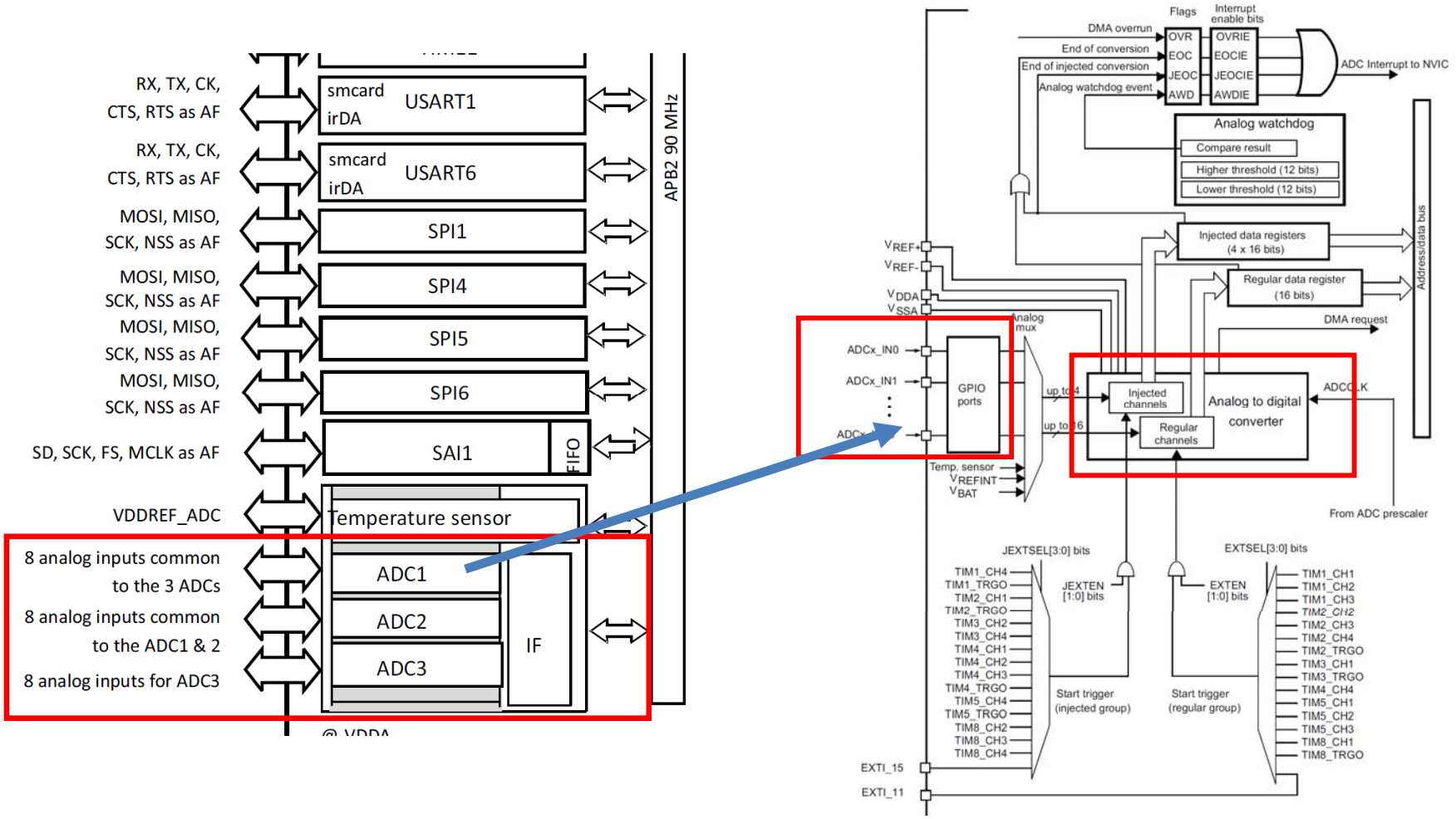


Convertidor Analógico / Digital. Ejemplo



Valores muestreados	3	4	7	6	4	4	4	5	5	2	0	2	3	
Códigos binarios	011	100	111	110	100	100	100	101	101	010	000	010	011	

ADCs in STM32F429



- Use of ADC1 analog to digital converter:
  - 12 bits of resolution
  - Input range (0 to 3.3v)
  - Resolution in volts = 3.3/4096 (0.8 mV)
  - Channel: ADC123_IN10, channel 10 in ADC1, ADC2 and ADC3
  - Single channel conversion mode managed by polling
    - We order the conversion start, and we wait until conversion is done
  - Use the code given: adc.h and adc.c

➢ **Manejo del ADC desde la capa HAL**

CN9	1	A0	ADC	PA3	ADC123_IN3	Arduino support
	3	A1	ADC	PC0	ADC123_IN10	
	5	A2	ADC	PC3	ADC123_IN13	
	7	A3	ADC	PF3	ADC3_IN9	
	9	A4	ADC	PF5 or PB9(1)	ADC3_IN15 (PF5) or I2C1_SDA (PB9)	
	11	A5	ADC	PF10 or PB8(1)	ADC3_IN8 (PF10) or I2C1_SCL (PB8)	
	13	D72	NC	-	-	-
	15	D71	I/O	PA7(2)	I/O	
	17	D70	I2C_B_SMBA	PF2	I2C_2	
	19	D69	I2C_B_SCL	PF1		

Code details

adc.h: three functions defined

```
#include "stm32f4xx_hal.h"
#ifndef __ADC_H
        void ADC1_pins_F429ZI_config(void); // pins configuration
        int ADC_Init_Single_Conversion(ADC_HandleTypeDef *, ADC_TypeDef  *); //adc configuration
        float ADC_getVoltage(ADC_HandleTypeDef * , uint32_t ); //read voltage converted to floating point
#endif
```

adc.c: configuring the pin for analog input

```
void ADC1_pins_F429ZI_config(){
      GPIO_InitTypeDef GPIO_InitStruct = {0};
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    /*PC0     ------> ADC1_IN10
    PC3     ------> ADC1_IN13
    */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}
```

adc.c: single conversion configuration

```
int ADC_Init_Single_Conversion(ADC_HandleTypeDef *hadc, ADC_TypeDef
*ADC_Instance)
{
  hadc->Instance = ADC_Instance;
  hadc->Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc->Init.Resolution = ADC_RESOLUTION_12B;
  hadc->Init.ScanConvMode = DISABLE;
  hadc->Init.ContinuousConvMode = DISABLE;
  hadc->Init.DiscontinuousConvMode = DISABLE;
  hadc->Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc->Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc->Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc->Init.NbrOfConversion = 1;
  hadc->Init.DMAContinuousRequests = DISABLE;
 hadc->Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(hadc) != HAL_OK)
  {
    return -1;
  }
 return 0;
}
```

adc.c: getting the voltage

```
float ADC_getVoltage(ADC_HandleTypeDef *hadc, uint32_t Channel)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	HAL_StatusTypeDef status;

	uint32_t raw = 0;
	float voltage = 0;
	sConfig.Channel = Channel;
	sConfig.Rank = 1;
	sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
	if (HAL_ADC_ConfigChannel(hadc, &sConfig) != HAL_OK)  {
		return -1;
	}
	HAL_ADC_Start(hadc);
	do (
		status = HAL_ADC_PollForConversion(hadc, 0); //This funtion uses the HAL_GetTick(), then it
                                                             only can be executed when the OS is running
	while (status != HAL_OK);
	raw = HAL_ADC_GetValue(hadc);
	voltage = raw*VREF/RESOLUTION_12B;
	return voltage;

}
```

Using the code given

```
void my_Thread (void *argument) {
 ADC_HandleTypeDef adchandle; //handler definition
 ADC1_pins_F429ZI_config(); //specific PINS configuration
 float value;
 ADC_Init_Single_Conversion(&adchandle , ADC1);
 while (1) {
 value=ADC_getVoltage(&adchandle , 10 );
 osDelay(1000);
 }
}
```

Sistemas Basados en Microprocesador

Bloque 3. DISEÑO

Especificaciones