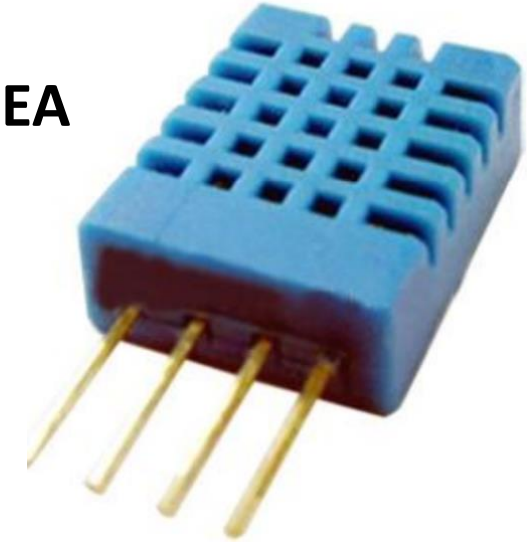
MICROS 32 BITS
STM – I2C

ROBINSON JIMENEZ MORENO

INTRODUCCION: COMUNICACION SERIAL A 1 LINEA



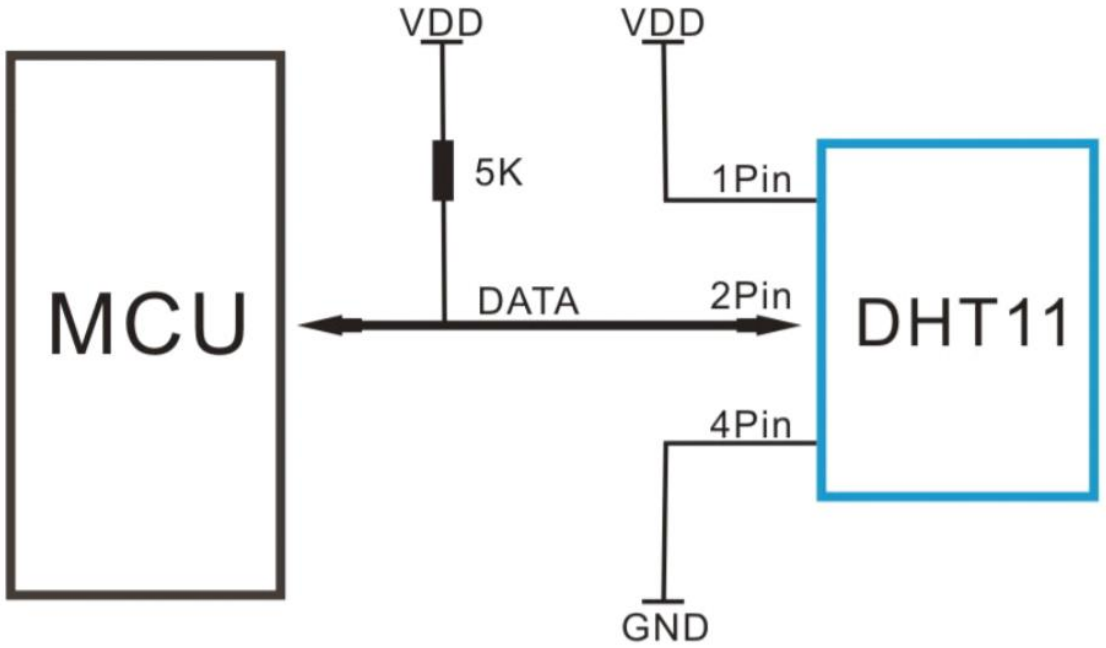
DHT11 Humidity & Temperature Sensor

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.

https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

2. Technical Specifications:

Overview:



Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

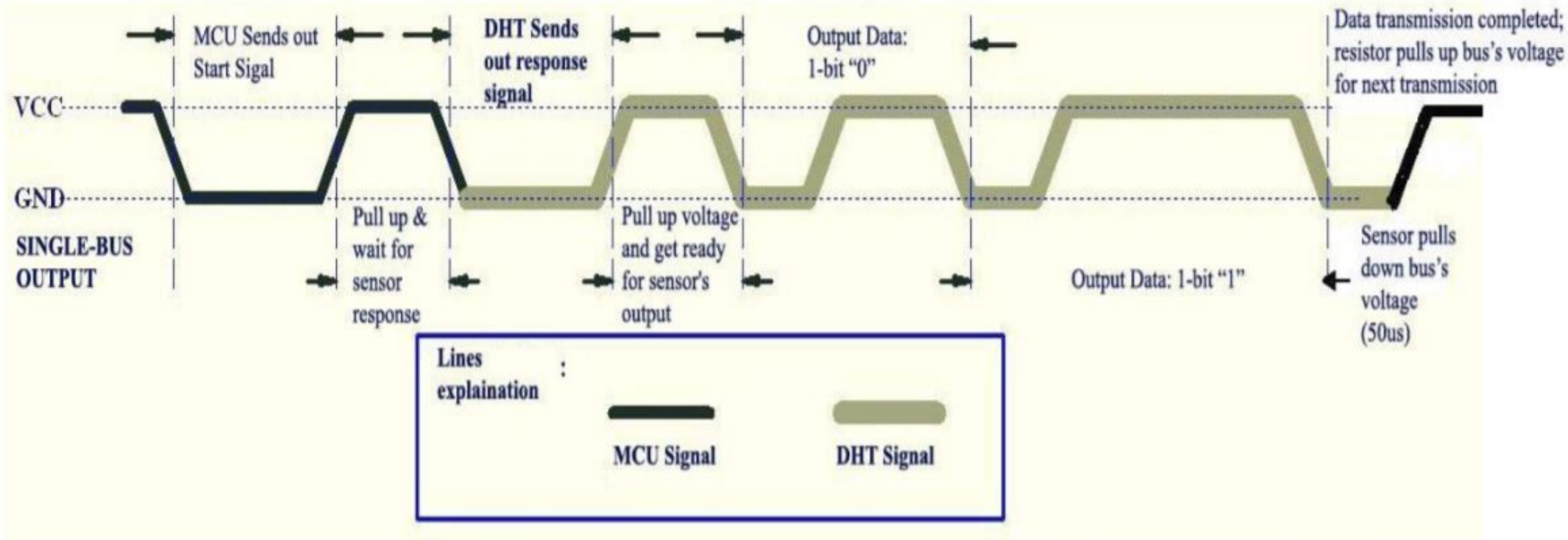
Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

5.1 Overall Communication Process (Figure 2, below)

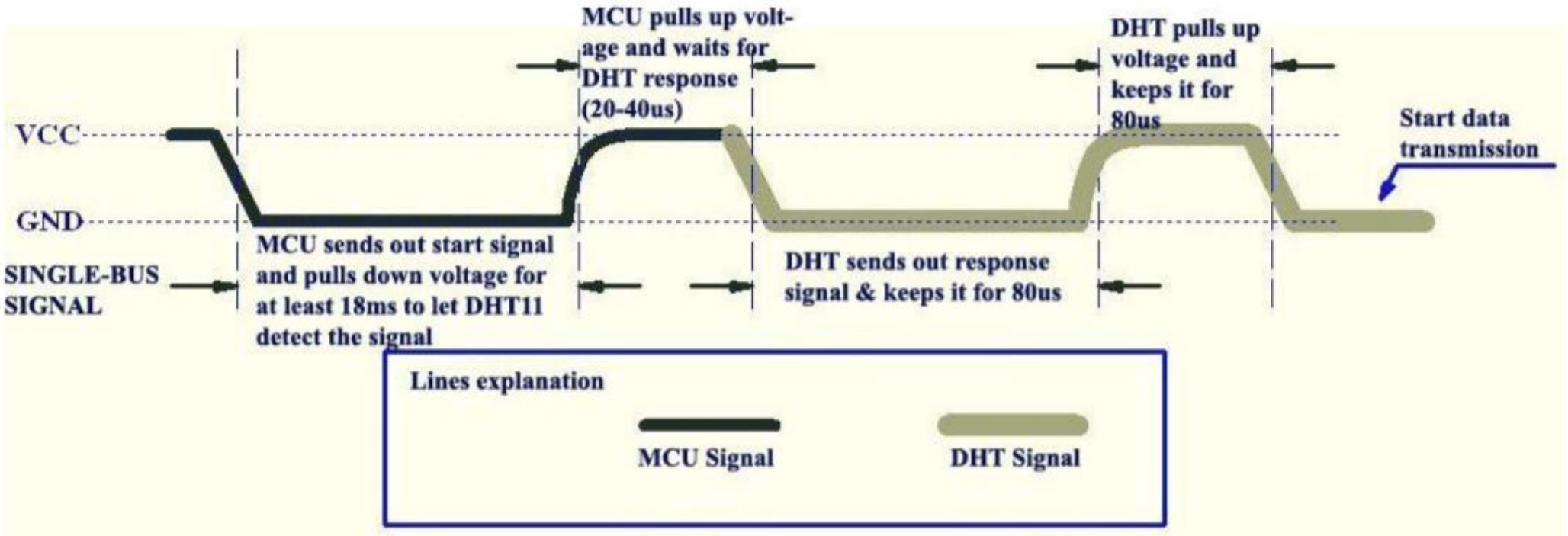
When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.



https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

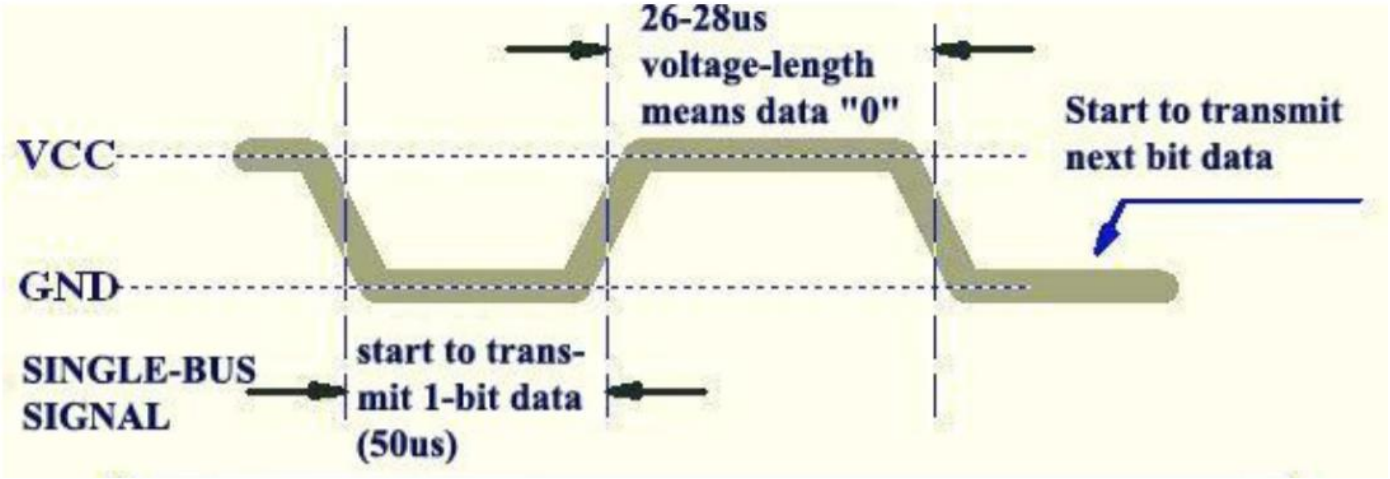
5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

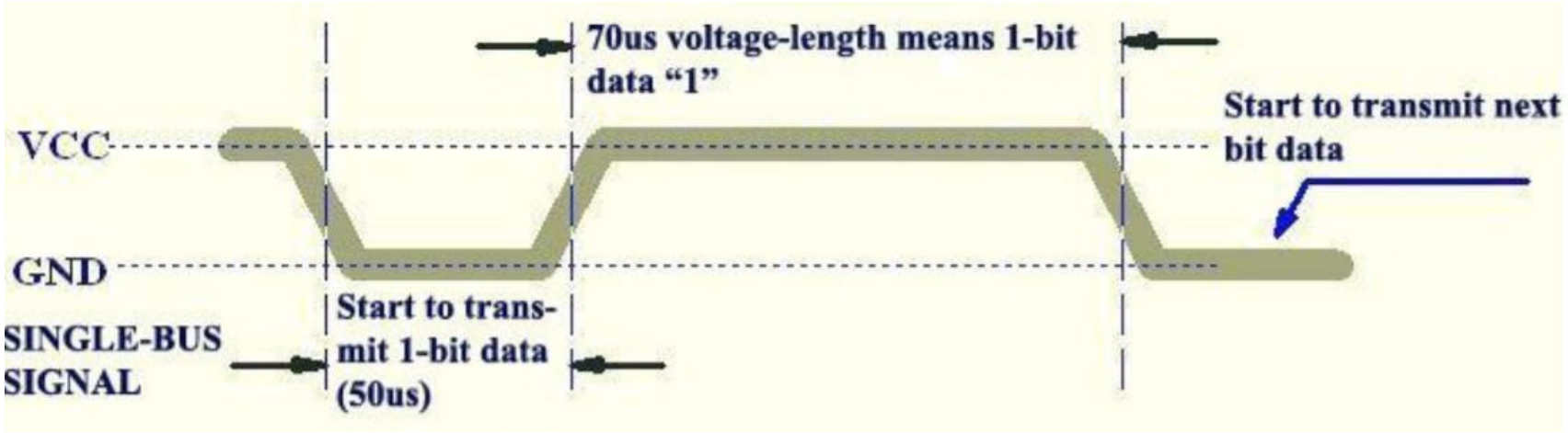
Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.



https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).





https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf


```
//////////////// Temper humedad///////////////////
#include "LibrLCD.h"
LibrLCD lib;
void LCD();
char dato[7]={'T','e','m','p',' ','=',' '};
char dato2[7]={'H','u','m','e',' ','=',' '};
char clear=0x01;
char LCD1L=0x80;
char LCD2L=0xC0;
int t,h,h1,h2,datos,ck=0;
```

```
int main(void){
  lib.cfglcd();
  lib.send_comando(clear);
  for (int i=0;i<50000;i++);
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);
  RCC -> APB1ENR = 0X1;
  TIM2 -> CR1 = 0X1;
  TIM2 -> ARR = 30000;
  TIM2 -> PSC = 16;
  GPIOC->MODER =1;
  GPIOC->OSPEEDR =3;
  GPIOC->OTYPER =1;
while(true){}
}
void LCD(){
  h1=(h&0xf0)>>4;h2=h&0x0f;
lib.send_comando(LCD1L+4);
for (int i=0;i<7;i++){
  lib.send_dato(dato[i]); }
  lib.send_dato((t/10)+0x30);
  lib.send_dato((t%10)+0x30);
  lib.send_comando(LCD2L+4);
for (int i=0;i<7;i++){
  lib.send_dato(dato2[i]);}
  lib.send_dato(h2+0x30);
  lib.send_dato(h1+0x30);
}
```

Example 1: 40 data is received:

0011 0101	0000 0000	0001 1000	0000 0000	0100 1101
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate：

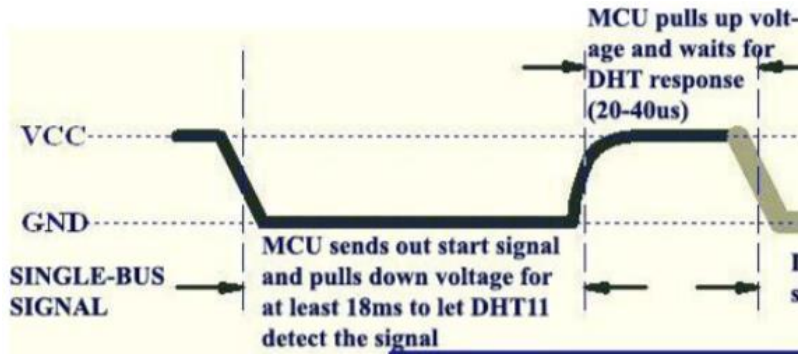
0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101

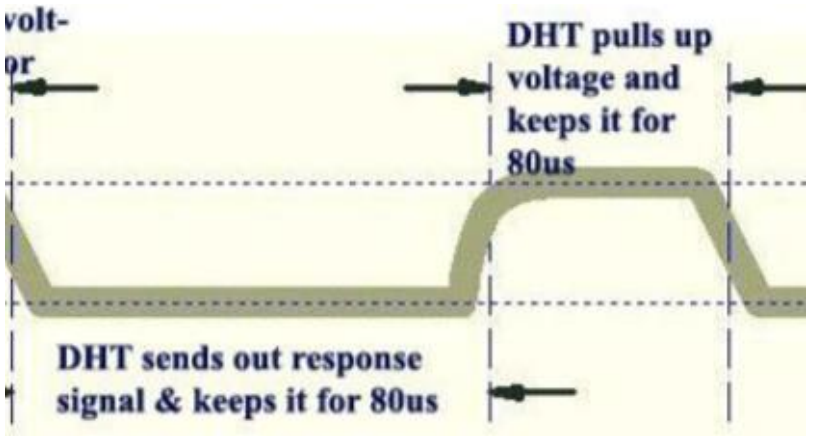
Received data is correct：

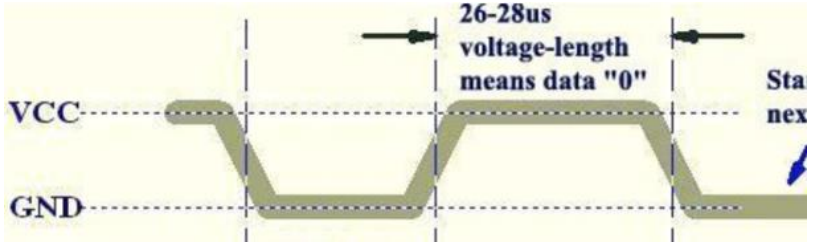
Humidity：0011 0101=35H=53%RH

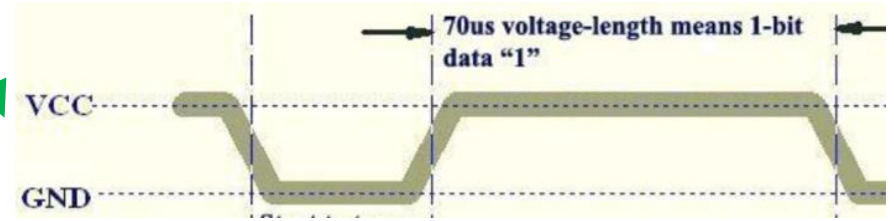
Temperature：0001 1000=18H=24℃

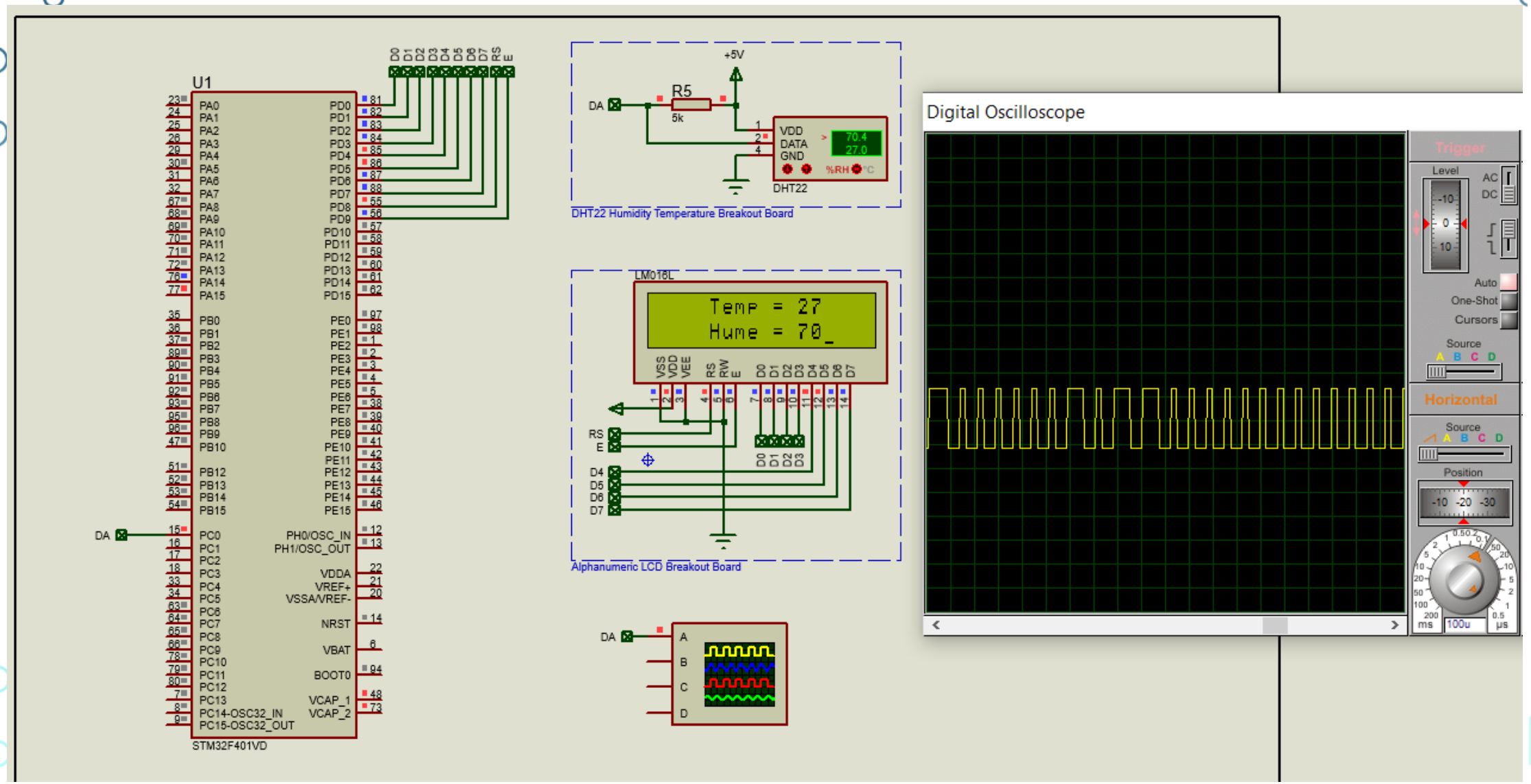
```
void SysTick_Handler(void)
{ datos=0;ck=0;
  GPIOC->ODR=0;
  TIM2 -> CNT=0;
  while(TIM2 -> CNT < 20000);
  GPIOC->ODR=1;
  TIM2 -> CNT=0;
  while(TIM2 -> CNT < 30);
  GPIOC->ODR=0;
  GPIOC->MODER =0;
  TIM2 -> CNT=0;
  while (TIM2 -> CNT <170 );
    for(int i=32;i>0;i--){
  while((GPIOC->IDR & 0x01)==0);
  TIM2 -> CNT=0;
  while((GPIOC->IDR & 0x01)==1);
  if(TIM2 -> CNT < 30){
  datos=datos | (0UL<<(i-1)); }
  if(TIM2 -> CNT > 50){
  datos=datos | (1UL<<(i-1));}
    }
  for(int i=8;i>0;i--){
  while((GPIOC->IDR & 0x01)==0);
  TIM2 -> CNT=0;
  while((GPIOC->IDR & 0x01)==1);
  if(TIM2 -> CNT < 30){
  ck=datos | (0UL<<(i-1));  }
  if(TIM2 -> CNT > 40){
  ck=ck | (1UL<<(i-1));}
    }
  TIM2 -> CNT=0;
  while(TIM2 -> CNT < 51);
  GPIOC->MODER =1; GPIOC->ODR=1;
  h=(datos&0xff000000)>>24;
  t=(datos&0x0000ff00)>>8;
LCD();
  }
}
```

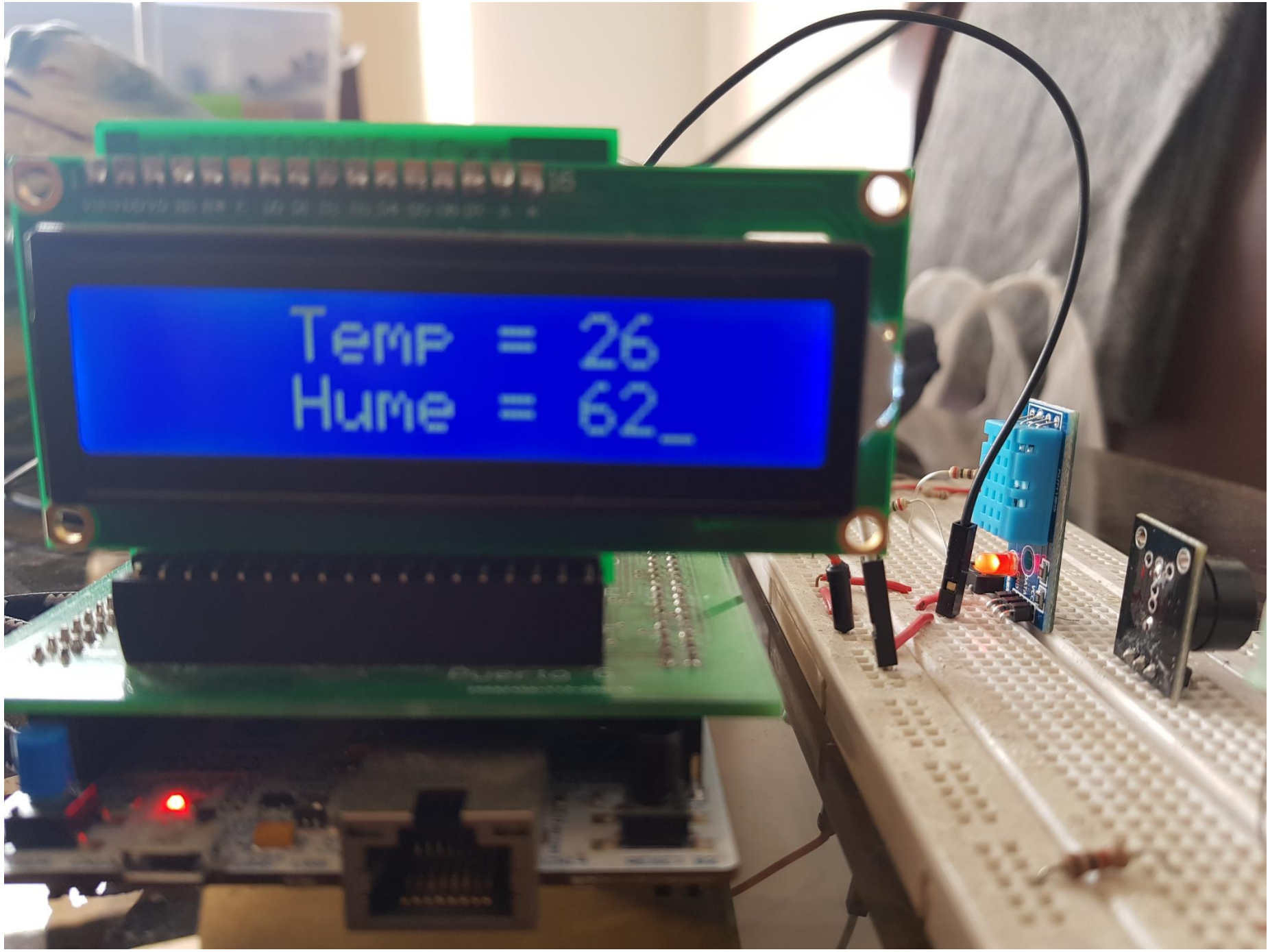









U1
STM32F401VD
DA
D0 D1 D2 D3 D4 D5 D6 D7 RS E
+5V
R5
5k
VDD
DATA
GND
70.4
27.0
%RH °C
DHT22
DHT22 Humidity Temperature Breakout Board
LM016L
Temp = 27
Hume = 70_
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
RS
E
D4
D5
D6
D7
Alphanumeric LCD Breakout Board
A
B
C
D
Digital Oscilloscope
Trigger
Level
AC
DC
Auto
One-Shot
Cursors
Source
Horizontal
Position
ms
100u
µs



INGENIERÍA MECATRÓNICA UMNG

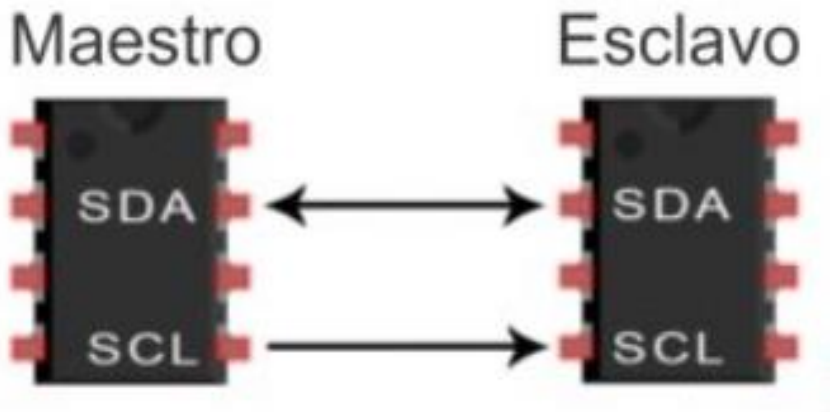
UNIVERSIDAD MILITAR NUEVA GRANADA

COMUNICACION A 2 LINES I2C

I2C o Circuito Interintegrado (Inter-Integrated Circuit) es un protocolo de comunicación serial desarrollado por Phillips Semiconductors en la década de los 80s. Se creó para poder comunicar varios chips al mismo tiempo dentro de los televisores.

Con el protocolo I2C es posible tener a varios maestros controlando uno o múltiples esclavos. Esto puede ser de gran ayuda cuando se van a utilizar varios microcontroladores para almacenar un registro de datos hacia una sola memoria o cuando se va a mostrar información en una sola pantalla.

El protocolo I2C utiliza sólo dos vías o cables de comunicación, así como también lo hace el protocolo UART.



SDA – Serial Data. Es la vía de comunicación entre el maestro y el esclavo para enviarse información.
SCL – Serial Clock. Es la vía por donde viaja la señal de reloj.

I2C es un protocolo de comunicación serial: envía información a través de una sola vía de comunicación. La información es enviada bit por bit de forma coordinada. Trabaja de forma síncrona,el envío de bits por la vía de comunicación SDA está sincronizado por una señal de reloj que comparten tanto el maestro como el esclavo a través de la vía SCL.

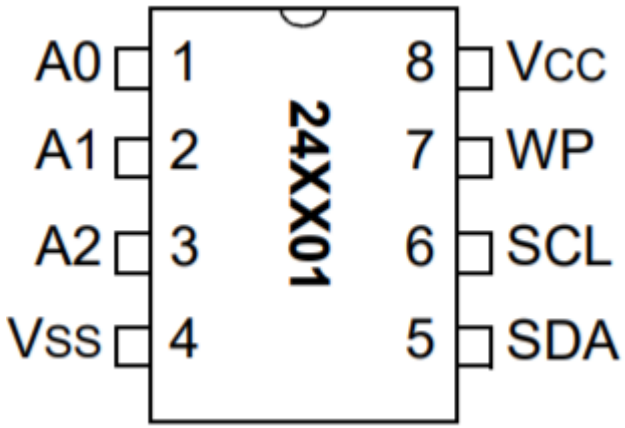
Velocidad máxima	Modo estándar (Sm) = 100kbps
	Modo rápido (Fm) = 400kbps
	Modo High Speed (Fm+) = 3.4Mbps
	Modo Ultra Fast (Hs-mode) = 5Mbps

La información viaja en mensajes divididos en tramas de datos. Cada mensaje lleva un trama con una dirección la cuál transporta la dirección binaria del esclavo al que va dirigido el mensaje, y una o más tramas que llevan la información del mensaje. También el mensaje contiene condiciones de inicio y paro, lectura y escritura de bits, y los bits ACK y NACK.

Mensaje

Start	7 o 10 Bits	Bit para Leer/ Escribir	Bit para reconocer ACK/ NACK	8 Bits	Bit para reconocer ACK/ NACK	8 Bits	Bit para reconocer ACK/ NACK	Stop
Condición de inicio	Sección destinada para la dirección			Sección 1 para transportar información		Sección 2 para transportar información		Condición de paro

Memoria I2C



A control byte is the first byte received following the Start condition from the master device. The control byte consists of a four-bit control code. For the 24XX01, this is set as '1010' binary for read and write operations. The next three bits of the control byte are 'don't care's' for the 24XX01.

The last bit of the control byte defines the operation to be performed. When set to '1', a read operation is selected. When set to '0', a write operation is selected. Following the Start condition, the 24XX01 monitors the SDA bus checking the device type identifier being transmitted. Upon receiving a '1010' code, the slave device outputs an Acknowledge signal on the SDA line. Depending on the state of the R/$\overline{W}$ bit, the 24XX01 will select a read or write operation.

Operation	Control Code	Block Select	R/$\overline{W}$
Read	1010	Block Address	1
Write	1010	Block Address	0

FIGURE 4-1: BYTE WRITE

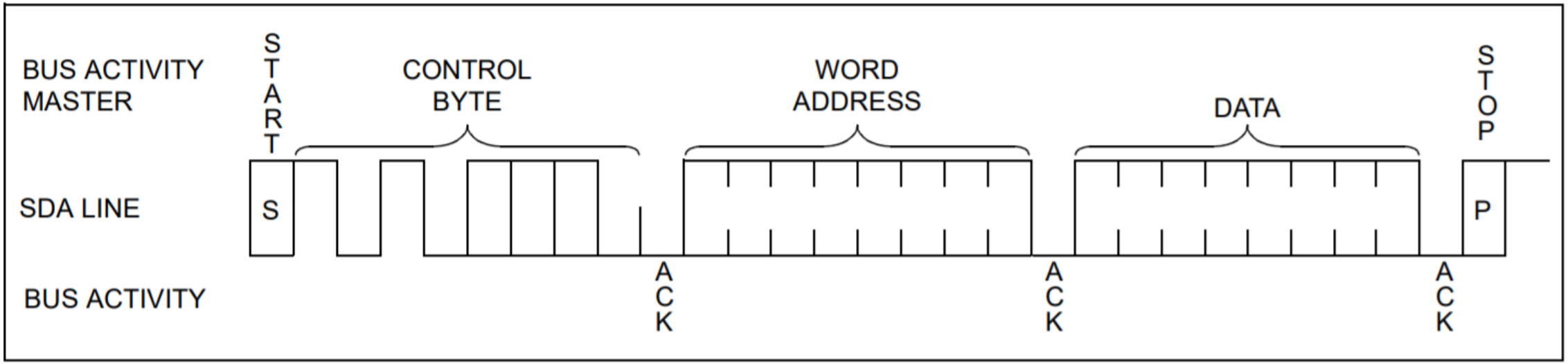
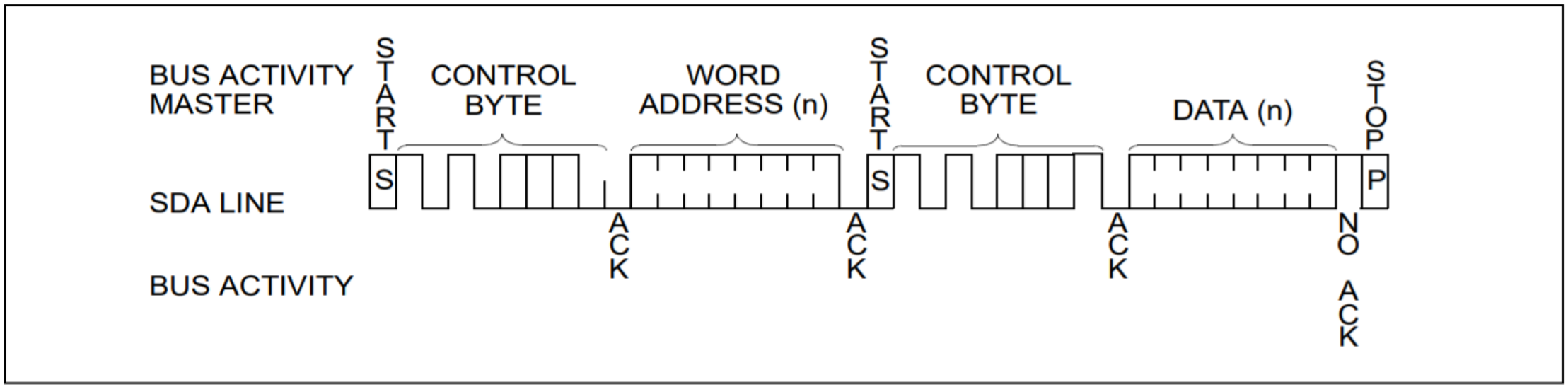
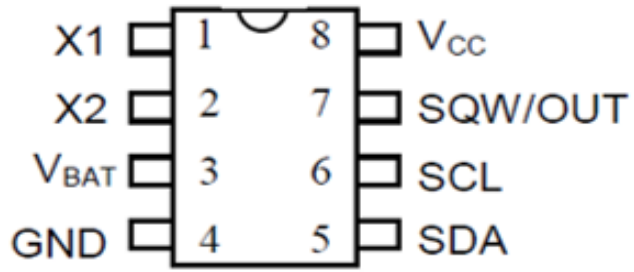


FIGURE 7-2: RANDOM READ



Reloj En Tiempo Real (RTC)

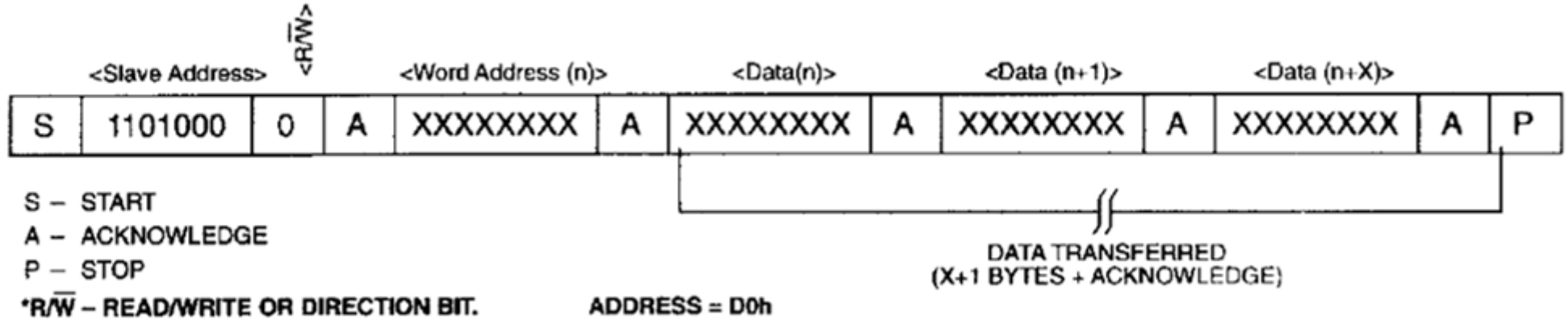
Otro dispositivo común para uso del protocolo I2C es el reloj en tiempo real de referencia DS107, que requiere un oscilador externo en los pines X1 y X2 para manejar la temporización, como se aprecia en la siguiete figura.



Su configuración require del uso de la siguiente tabla para programar y leer la hora y calendario.

PosMem\bit	7	6	5	4	3	2	1	0
00H	0	Decenas de Segundos			Unidades de Segundos			
01H	0	Decenas de Minutos			Unidades de Minutos			
02H	0	12/24	am/pm	Dec.H.	Unidades de Horas			
03H	0	0	0	0	0	Dia de la semana		
04H	0	0	Decenas Día		Unidades de Día			
05H	0	0	0	Dec. M.	Unidades de Mes			
06H	Decenas de Año				Unidades de Año			

DATA WRITE – SLAVE RECEIVER MODE Figure 6

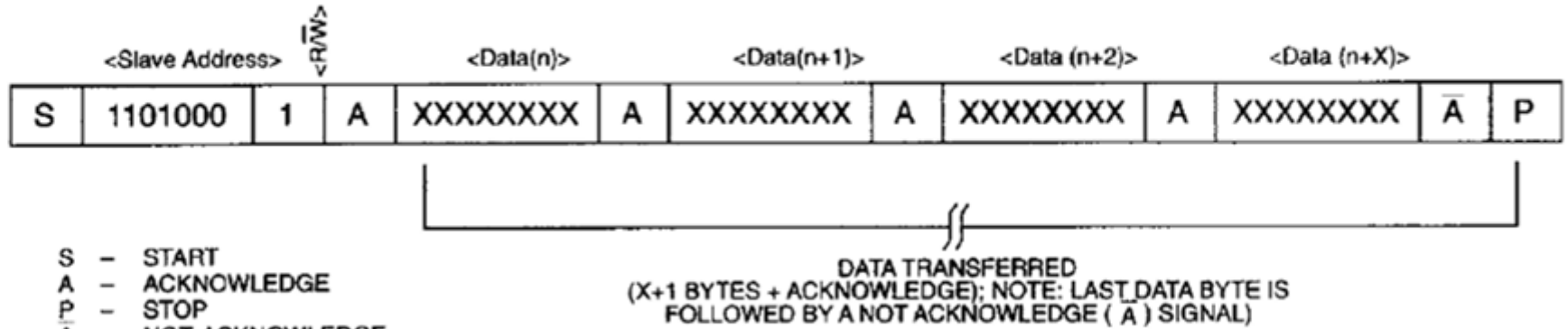


S – START
A – ACKNOWLEDGE
P – STOP

*R/$\overline{W}$ – READ/WRITE OR DIRECTION BIT.

ADDRESS = D0h

DATA READ – SLAVE TRANSMITTER MODE Figure 7



S – START
A – ACKNOWLEDGE
P – STOP
Ā – NOT ACKNOWLEDGE

*R/$\overline{W}$ – READ/WRITE OR DIRECTION BIT

ADDRESS = D1h

I2C main features

- I^2C bus specification rev03 compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode
  - Multiple 7-bit slave addresses (2 addresses, 1 with configurable mask)
  - All 7-bit addresses acknowledge mode
  - General call
  - Programmable setup and hold times
  - Easy to use event management
  - Optional clock stretching
  - Software reset

I2C features[1]	I2C1	I2C2	I2C3	I2C4
7-bit addressing mode	X	X	X	X
10-bit addressing mode	X	X	X	X
Standard-mode (up to 100 kbit/s)	X	X	X	X
Fast-mode (up to 400 kbit/s)	X	X	X	X
Fast-mode Plus with 20mA output drive I/Os (up to 1 Mbit/s)	X	X	X	X
Independent clock	X	X	X	X

The I2C is clocked by an independent clock source which allows to the I2C to operate independently from the PCLK frequency.

This independent clock source can be selected from the following three clock sources:

- PCLK1: APB1 clock (default value)
- HSI: high speed internal oscillator
- SYSCLK: system clock

Mode selection

The interface can operate in one of the four following modes:

- Slave transmitter
- Slave receiver
- Master transmitter
- Master receiver

By default, it operates in slave mode. The interface automatically switches from slave to master when it generates a START condition, and from master to slave if an arbitration loss or a STOP generation occurs, allowing multimaster capability.

I^2C master mode

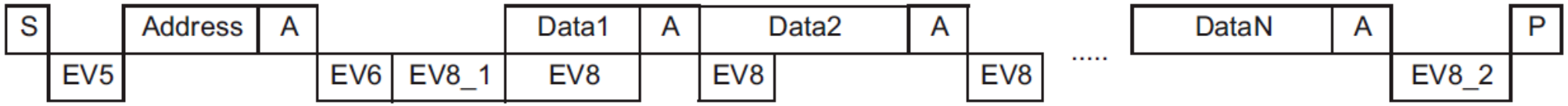
In Master mode, the I^2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a Start condition and ends with a Stop condition. Master mode is selected as soon as the Start condition is generated on the bus with a START bit.

The following is the required sequence in master mode.

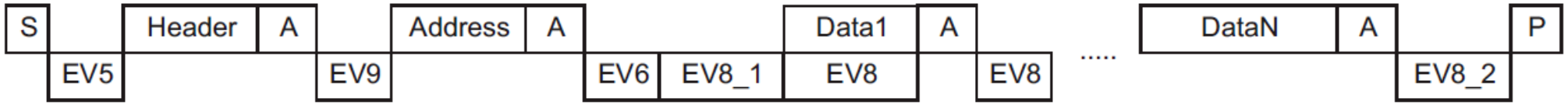
- Program the peripheral input clock in I2C_CR2 Register in order to generate correct timings
- Configure the clock control registers
- Configure the rise time register
- Program the I2C_CR1 register to enable the peripheral
- Set the START bit in the I2C_CR1 register to generate a Start condition

Figure 164. Transfer sequence diagram for master transmitter

7-bit master transmitter



10-bit master transmitter



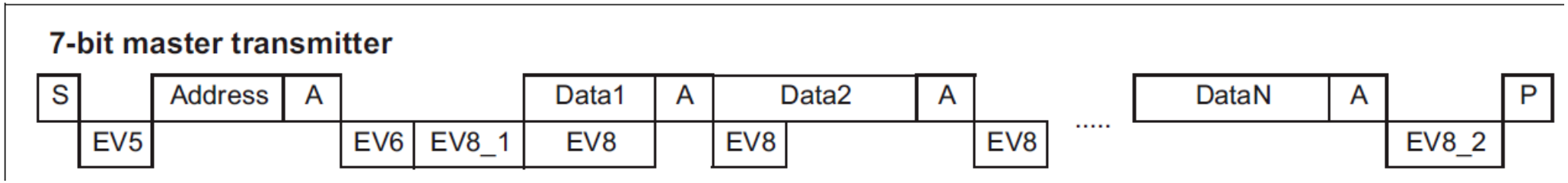
Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge,
EVx= Event (with interrupt if ITEVFEN = 1)

EV5: SB=1, cleared by reading SR1 register followed by writing DR register with Address.
EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2.
EV8_1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV8: TxE=1, shift register not empty,.data register empty, cleared by writing DR register
EV8_2: TxE=1, BTF = 1, Program Stop request. TxE and BTF are cleared by hardware by the Stop condition
EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

ai18210

1. The EV5, EV6, EV9, EV8_1 and EV8_2 events stretch SCL low until the end of the corresponding software sequence.
2. The EV8 event stretches SCL low if the software sequence is not complete before the end of the next byte transmission.

Figure 164. Transfer sequence diagram for master transmitter



I²C Status register 1 (I2C_SR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

```
I2C1->CR1 |= (1<<8);
while (!(I2C1->SR1 & 0x01));

while (!(I2C1->SR1 & (1<<1)));
while (!(I2C1->SR1 & (1<<7)));

I2C1->CR1  |= (1<<9);
while (I2C1->SR2 & 0x0002);
```

EV5: SB=1, cleared by reading SR1 register followed by writing DR register with Address.

EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2.

EV8_1: TxE=1, shift register empty, data register empty, write Data1 in DR.

EV8: TxE=1, shift register not empty,.data register empty, cleared by writing DR register

EV8_2: TxE=1, BTF = 1, Program Stop request. TxE and BTF are cleared by hardware by the Stop condition

EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

6.3.11 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PWR EN	Reserved				I2C3 EN	I2C2 EN	I2C1 EN	Reserved			USART2 EN	Reser-ved
			rw					rw	rw	rw				rw	

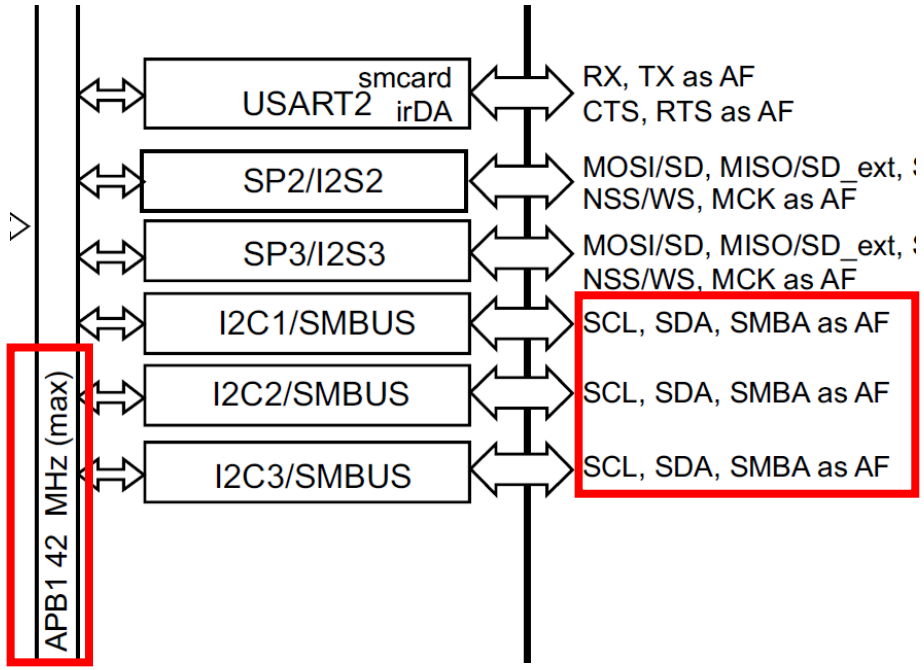


Table 9. Alternate function mapping

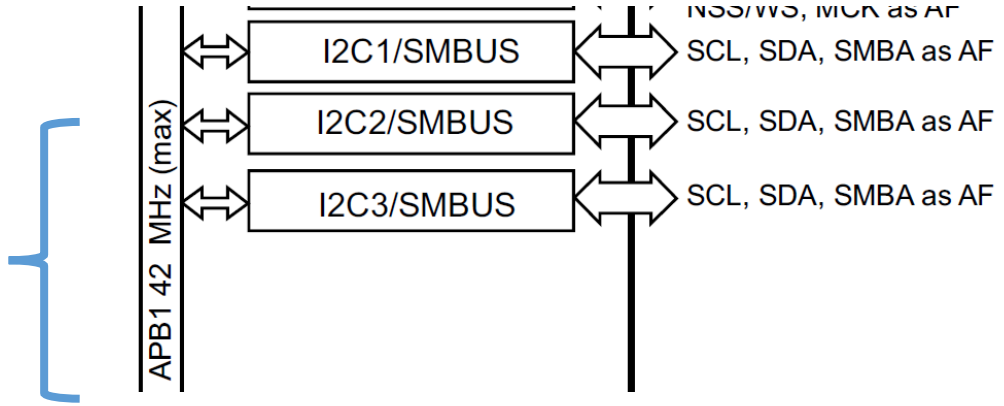
Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	SPI2/I2S2/ SPI3/ I2S3	SPI3/I2S3 USART1 USART2
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	-	-	-	-	-
	PB1	-	TIM1_CH3N	TIM3_CH4	-	-	-	-	-
	PB2	-	-	-	-	-	-	-	-
	PB3	JTDO-SWO	TIM2_CH2	-	-	-	SPI1_SCK	SPI3_SCK/ I2S3_CK	-
	PB4	JTRST	-	TIM3_CH1	-	-	SPI1_ MISO	SPI3_MISO	I2S3ext_SD
	PB5	-	-	TIM3_CH2	-	I2C1_ SMBA	SPI1 _MOSI	SPI3_MOSI/ I2S3_SD	-
	PB6	-	-	TIM4_CH1	-	I2C1_SCL	-	-	USART1_TX
	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	-	USART1_RX
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-	-
	PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS/I 2S2_WS	-	-
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK/I 2S2_CK	-	-
	PB12	-	TIM1_BKIN	-	-	I2C2_ SMBA	SPI2_NSS/I 2S2_WS	-	-

18.6.2 I²C Control register 2 (I2C_CR2)

Address offset: 0x04
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]					
			rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

5	4	3	2	1	0
FREQ[5:0]					
rw	rw	rw	rw	rw	rw




```
I2C1->CR2 |= (42<< 0);
```

18.6.8 I²C Clock control register (I2C_CCR)

Address offset: 0x1C
Reset value: 0x0000

Note: *f_{PCLK1} must be at least 2 MHz to achieve Sm mode I²C frequencies. It must be at least 4 MHz to achieve Fm mode I²C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I²C Fm mode clock.*

The CCR register must be config[illegible]

$$CCR = \frac{T_{r(SCL)} + T_{w(SCLH)}}{T_{PCLK1}}$$

15	14	13	12	11	10	9	
F/S	DUTY	Reserved					
rw	rw			rw	rw	rw	r

Bits 11:0 **CCR[11:0]:** Clock control register in Fm/Sm mode (Ma

Controls the SCL clock in master mode.

Sm mode or SMBus:

$T_{high} = CCR * T_{PCLK1}$
$T_{low} = CCR * T_{PCLK1}$

Fm mode:

If DUTY = 0:
$T_{high} = CCR * T_{PCLK1}$
$T_{low} = 2 * CCR * T_{PCLK1}$

If DUTY = 1: (to reach 400 kHz)
$T_{high} = 9 * CCR * T_{PCLK1}$
$T_{low} = 16 * CCR * T_{PCLK1}$

For instance: in Sm mode, to generate a 100 kHz S

If FREQR = 08, T_{PCLK1} = 125 ns so CCR must be p
(0x28 <=> 40d x 125 ns = 5000 ns.)

Table 61. I²C characteristics

Symbol	Parameter	Standard mode I²C(1)(2)		Fast mode I²C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{w(SCLL)}$	SCL clock low time	4.7	-	1.3	-	µs
$t_{w(SCLH)}$	SCL clock high time	4.0	-	0.6	-	
$t_{su(SDA)}$	SDA setup time	250	-	100	-	ns
$t_{h(SDA)}$	SDA data hold time	-	3450(3)	-	900(4)	
$t_{v(SDA, ACK)}$	Data, ACK valid time	-	3.45	-	0.9	
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time	-	1000	-	300	
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time	-	300	-	300	
$t_{h(STA)}$	Start condition hold time	4.0	-	0.6	-	µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	0.6	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	0.6	-	µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7	-	1.3	-	µs
t_{SP}	Pulse width of the spikes that are suppressed by the analog filter for standard and fast mode	-	-	0.05	0.09(5)	µs

18.6.8 I²C Clock control register (I2C_CCR)

$$CCR = \frac{T_{r(SCL)} + T_{w(SCLH)}}{T_{PCLK1}}$$

T_{PCLK1} =1/42MHz

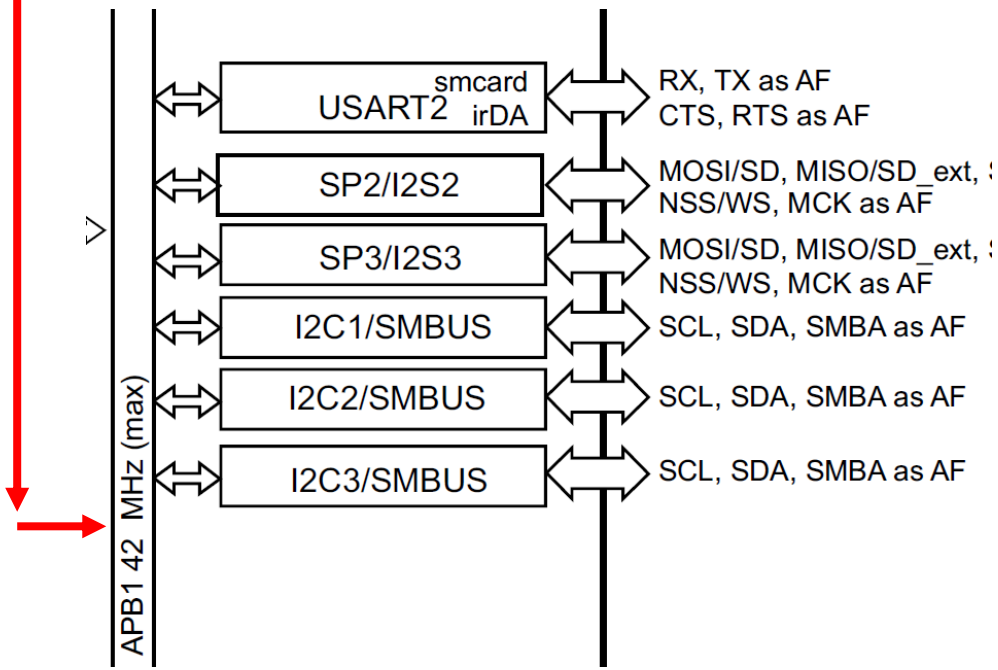


Table 61. I²C characteristics

Symbol	Parameter	Standard mode I²C(1)(2)		Fast mode I²C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{w(SCLL)}$	SCL clock low time	4.7	-	1.3	-	µs
$t_{w(SCLH)}$	SCL clock high time	4.0	-	0.6	-	
$t_{su(SDA)}$	SDA setup time	250	-	100	-	ns
$t_{h(SDA)}$	SDA data hold time	-	3450(3)	-	900(4)	
$t_{v(SDA, ACK)}$	Data, ACK valid time	-	3.45	-	0.9	
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time	-	1000	-	300	
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time	-	300	-	300	
$t_{h(STA)}$	Start condition hold time	4.0	-	0.6	-	µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	0.6	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	0.6	-	µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7	-	1.3	-	µs
t_{SP}	Pulse width of the spikes that are suppressed by the analog filter for standard and fast mode	-	-	0.05	0.09(5)	µs

24.6.9 I²C TRISE register (I2C_TRISE)

$$TRISE = \frac{T_{r(SCL)}}{T_{PCLK1}} + 1$$

Address offset: 0x20
Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TRISE[5:0]					
										rw	rw	rw	rw	rw	rw

Bits 15:6 Reserved, must be kept at reset value

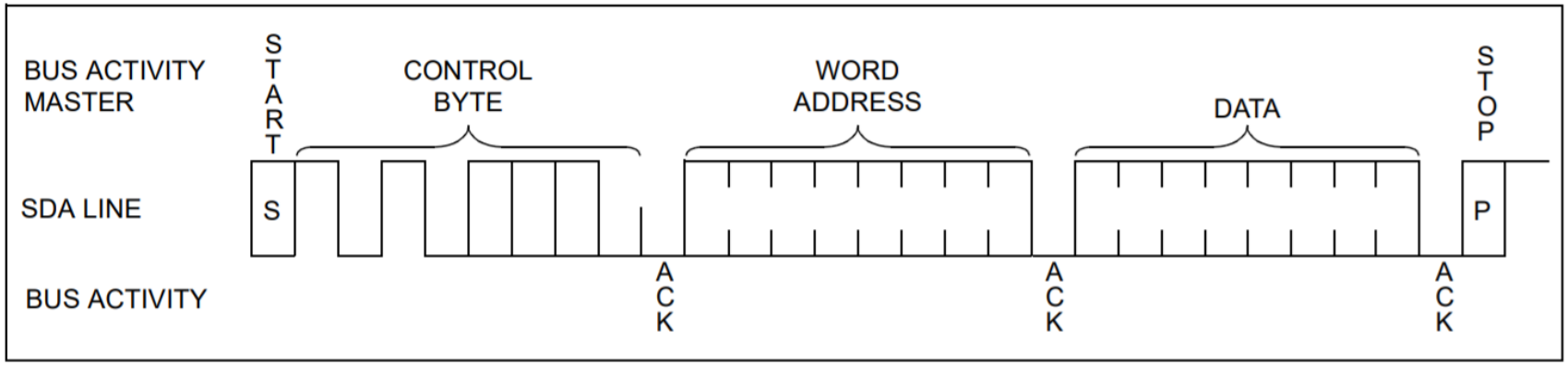
Bits 5:0 **TRISE[5:0]**: Maximum rise time in Fm/Sm mode (Master mode)

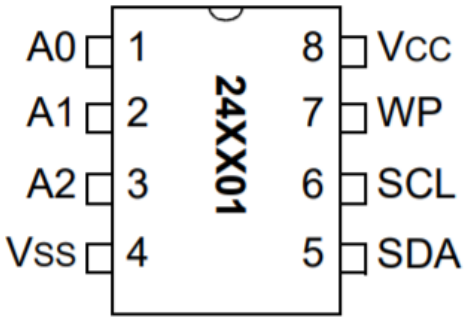
These bits should provide the maximum duration of the SCL feedback loop in master mode.
The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration.
These bits must be programmed with the maximum SCL rise time given in the I²C bus specification, incremented by 1.
For instance: in Sm mode, the maximum allowed SCL rise time is 1000 ns.
If, in the I2C_CR2 register, the value of FREQ[5:0] bits is equal to 0x08 and T_{PCLK1} = 125 ns therefore the TRISE[5:0] bits must be programmed with 09h.
(1000 ns / 125 ns = 8 + 1)
The filter value can also be added to TRISE[5:0].
If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note: TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

Memoria I2C

FIGURE 4-1: BYTE WRITE






```
void escr (){
    I2C1->CR1 |= (1<<8);
    while (!(I2C1->SR1 & 0x01));
    while (!(I2C1->SR1 & (1<<7)));
    I2C1->DR = 0xA0; //direccion
    while (!(I2C1->SR1 & (1<<1)));
    while (!(I2C1->SR1 & (1<<7)));
    I2C1->DR = 0x05;
    while (!(I2C1->SR1 & (1<<7)));
    I2C1->DR = 0x68;
    while (!(I2C1->SR1 & (1<<2)));
    I2C1->CR1  |= (1<<9);
    while (I2C1->SR2 & 0x0002);
WORK;
}
```

Communication flow

In Master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a START condition and ends with a STOP condition. Both START and STOP conditions are generated in master mode by software.

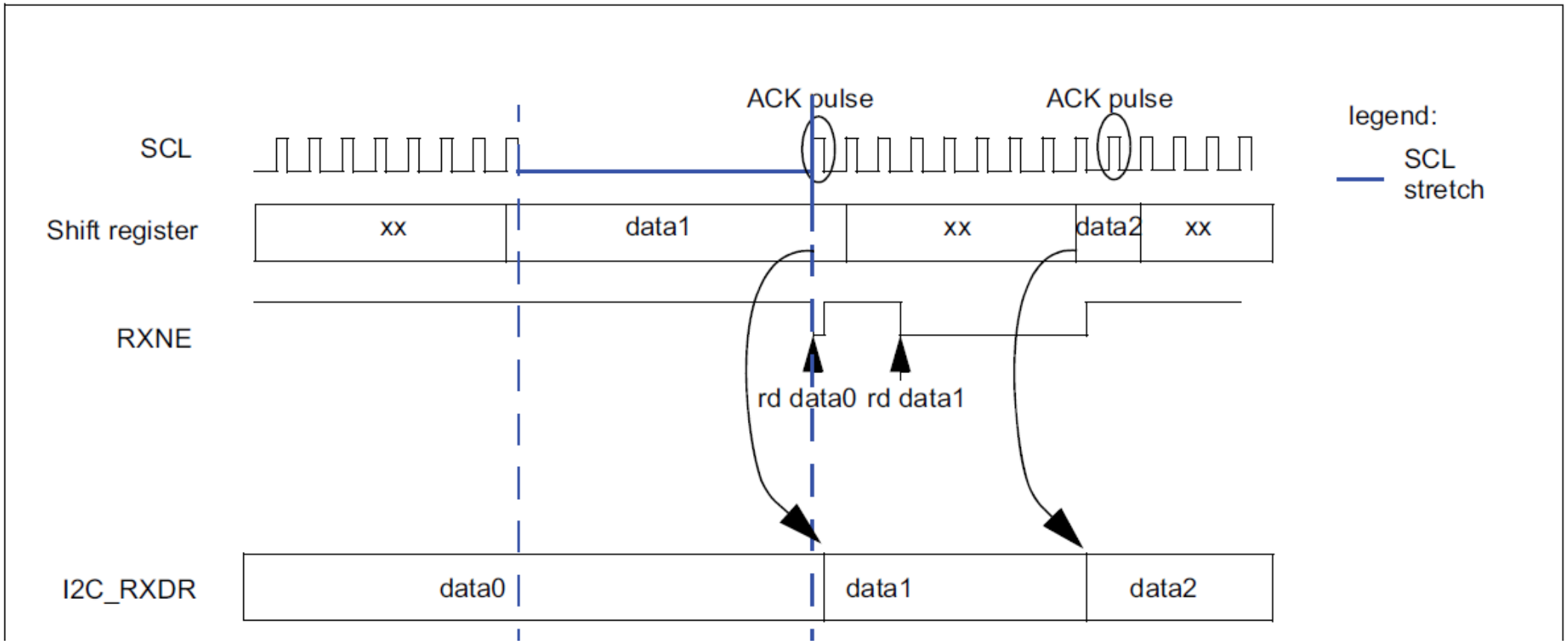
In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection can be enabled or disabled by software. The reserved SMBus addresses can also be enabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the START condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

Reception

The SDA input fills the shift register. After the 8th SCL pulse (when the complete data byte is received), the shift register is copied into I2C_RXDR register if it is empty (RXNE=0). If RXNE=1, meaning that the previous received data byte has not yet been read, the SCL line is stretched low until I2C_RXDR is read. The stretch is inserted between the 8th and 9th SCL pulse (before the Acknowledge pulse).

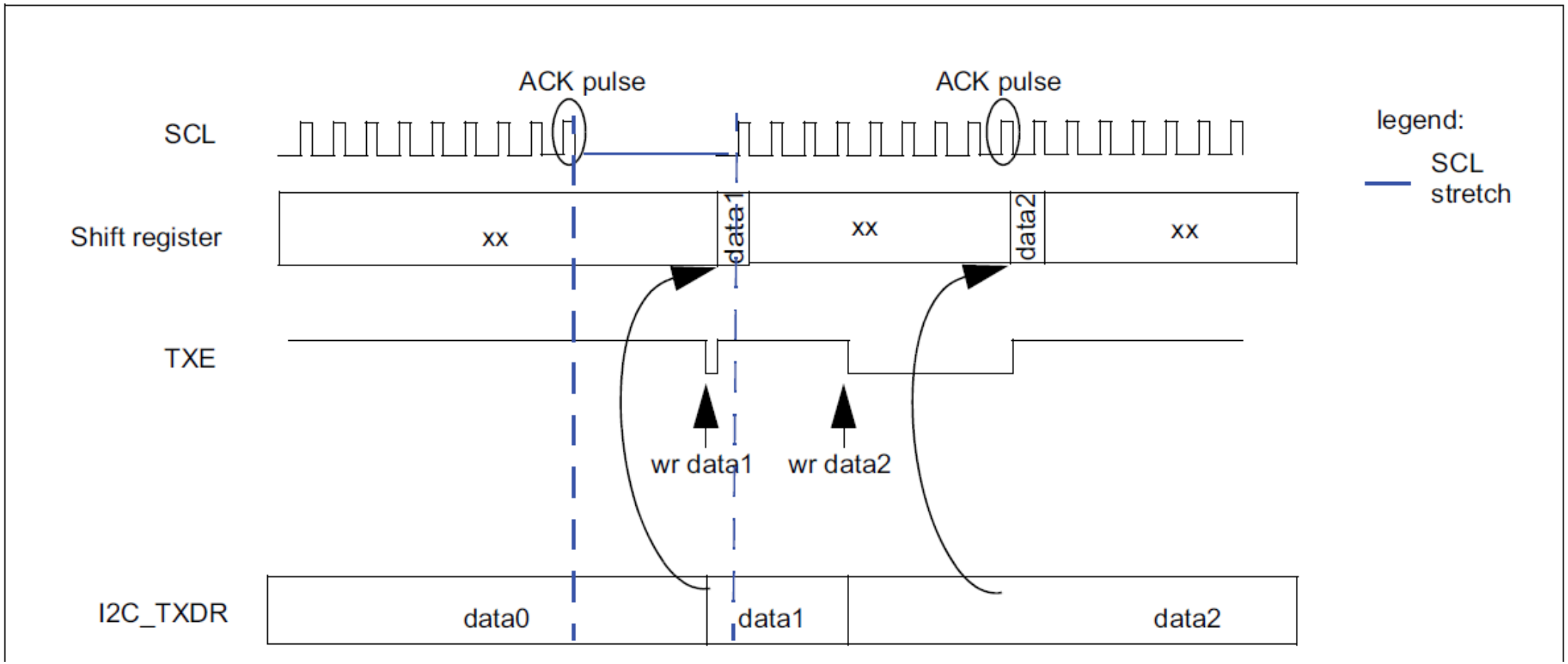
Figure 307. Data reception



Transmission

If the I2C_TXDR register is not empty (TXE=0), its content is copied into the shift register after the 9th SCL pulse (the Acknowledge pulse). Then the shift register content is shifted out on SDA line. If TXE=1, meaning that no data is written yet in I2C_TXDR, SCL line is stretched low until I2C_TXDR is written. The stretch is done after the 9th SCL pulse.

Figure 308. Data transmission



Ejemplo stm32f7:

Diseñar un programa que permita enviar y recibir un dato a través del modulo I2C.

Puertos y Pines:

- PF1-SCL (Reloj).
- PF0-SDA (Datos).
- Modulo I2C

RCC_AHB1ENR peripheral clock register.

```
RCC->AHB1ENR  |=  0x00000020; // Encender reloj puertoF
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPIEN	OTGHS EN	ETHM ACPTP EN	ETHM ACRX EN	ETHM ACTX EN	ETHMA CEN	Res.	DMA2D EN	DMA2 EN	DMA1 EN	DTCMRA MEN	Res.	BKPSR AMEN	Res.	Res.
	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
Res.	Res.	Res.	CRC EN	Res.	GPIOK EN	GPIOJ EN	GPIOI EN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
			rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Asignación de Puertos PF0 y PF1.

```
GPIOF->AFR[0] |=  0x00000044; // seleccion de la funcion alterna 4 del puerto F(I2C) para  PF1-SCL,PF0-SDA -> I2C2
```

Table 12. STM32F745xx and STM32F746x

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6
		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
Port E	PE14	-	TIM1_CH4	-	-	-	SPI4_MOSI	-
	PE15	-	TIM1_BKIN	-	-	-	-	-
	PF0	-	-	-	-	I2C2_SDA	-	-
	PF1	-	-	-	-	I2C2_SCL	-	-
	PF2	-	-	-	-	I2C2_SMBA	-	-
	PF3	-	-	-	-	-	-	-

GPIOx_Moder port moder register

```
GPIOF->MODER  |=  0x0000000A; // PF0,PF1 => en modo alterno
```

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+1:2y **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O mode.

00: Input mode (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

GPIOx_OTYPER port output type register.

```
GPIOF->OTYPER |=   0x0003;      // Open drain
```

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
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

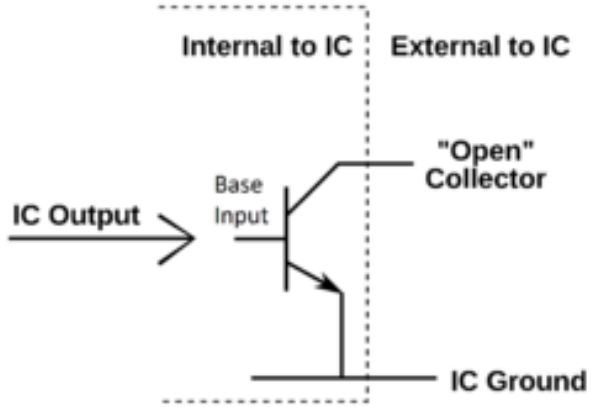
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output type.

0: Output push-pull (reset state)

1: Output open-drain



GPIOx_PUPDR port pull-up/pull-down register.

```
GPIOF->PUPDR |=  0x5;
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[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
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y+1:2y **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

GPIOx_OSPEEDR port output speed register.

```
GPIOF->OSPEEDR |= 0xC;
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [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
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y+1:2y **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

00: Low speed

01: Medium speed

10: High speed

11: Very high speed

RCC_APB1ENR peripheral clock enable register.

```
RCC->APB1ENR  |= 0x00400000;          // Enable clock for I2C2 bit 22
```

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 22 **I2C2EN:** I2C2 clock enable

This bit is set and cleared by software.

0: I2C2 clock disabled

1: I2C2 clock enabled

DCKCFGR2 dedicated clocks configuration register.

```
RCC->DCKCFGR2   |= 0x80000;                    //Reloj de frecuencia del I2C2
```

This register allows to select the source clock for the 48MHz, SDMMC, HDMI-CEC, LPTIM1, UARTs, USARTs and I2Cs clocks.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	SDMMC1SEL	CK48MSEL	CECSEL	LPTIM1SEL		I2C4SEL		I2C3SEL		I2C2SEL		I2C1SEL	
			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
UART8SEL		UART7SEL		USART6SEL		UART5SEL		UART4SEL		UART3SEL		UART2SEL		UART1SEL	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 19:18 **I2C2SEL**: I2C2 clock source selection

Set and reset by software.

00: APB1 clock (PCLK1) is selected as I2C2 clock

01: System clock is selected as I2C2 clock

10: HSI clock is selected as I2C2 clock

11: reserved

Control Register (I2C_CR1)

```
// Disable the I2Cx peripheral
I2C2->CR1 &= ~I2C_CR1_PE; //deshabilita SCL y SDA para el periferico
while (I2C2->CR1 & I2C_CR1_PE);
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	Res.	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw		rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw				rw	rw	rw	rw	rw	rw	rw	rw

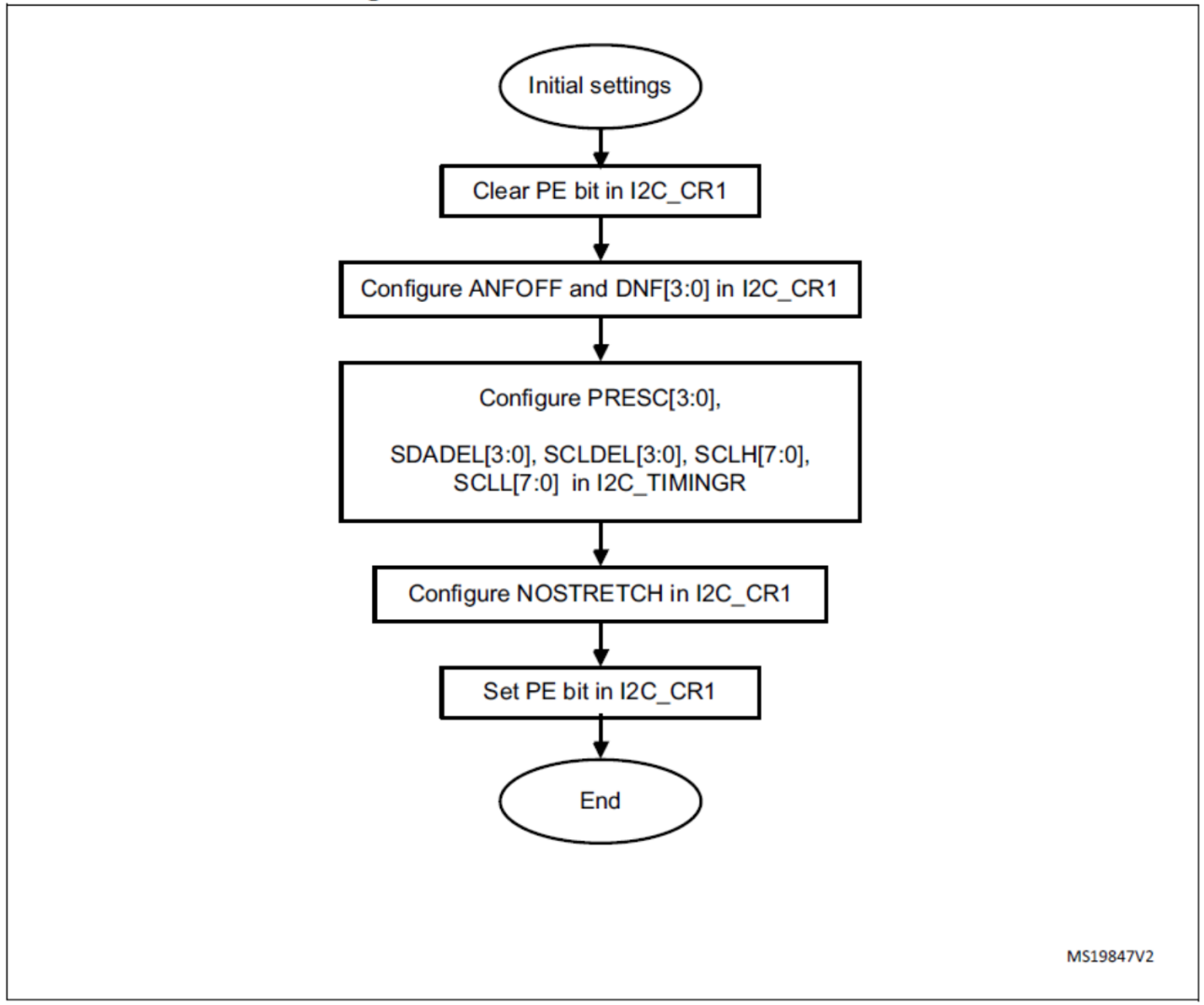
Bit 0 **PE**: Peripheral enable

0: Peripheral disable

1: Peripheral enable

Los tiempos deben ser configurados de tal forma que permitan la transferencia y configuración de información usada en los módulos maestros y esclavos. Esto se logra mediante la configuración del PRESC, SCLDEL, SDADEL, en el registro I2C_TIMINGR.

Figure 288. I2C initialization flowchart

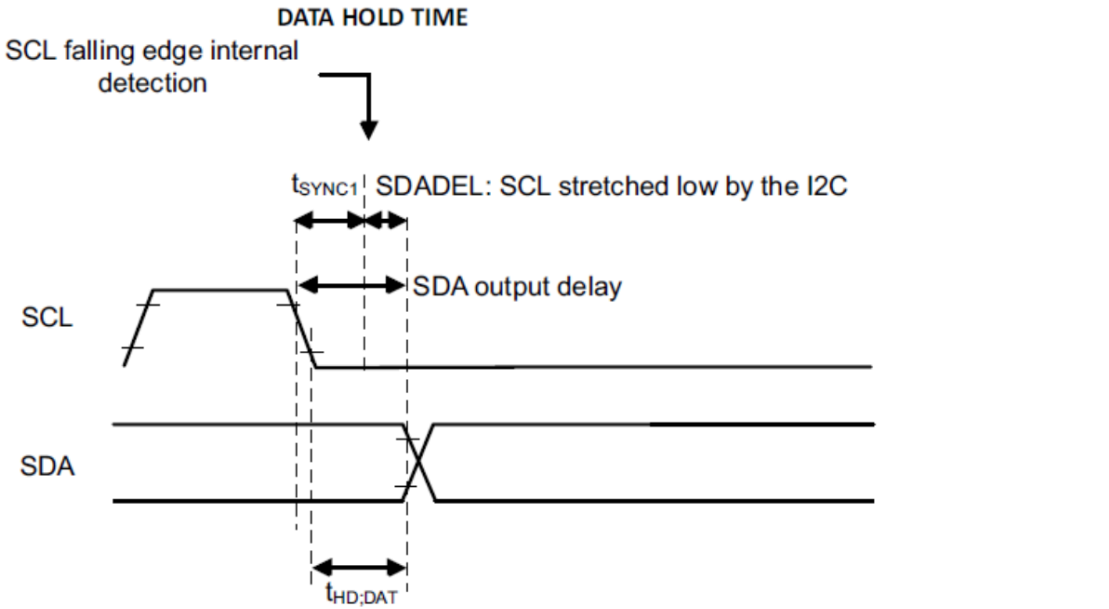


Los tiempos deben ser configurados de tal forma que permitan la transferencia y configuración de información usada en los módulos maestros y esclavos. Esto se logra mediante la configuración del PRESC, SCLDEL, SDADEL, en el registro I2C_TIMINGR.

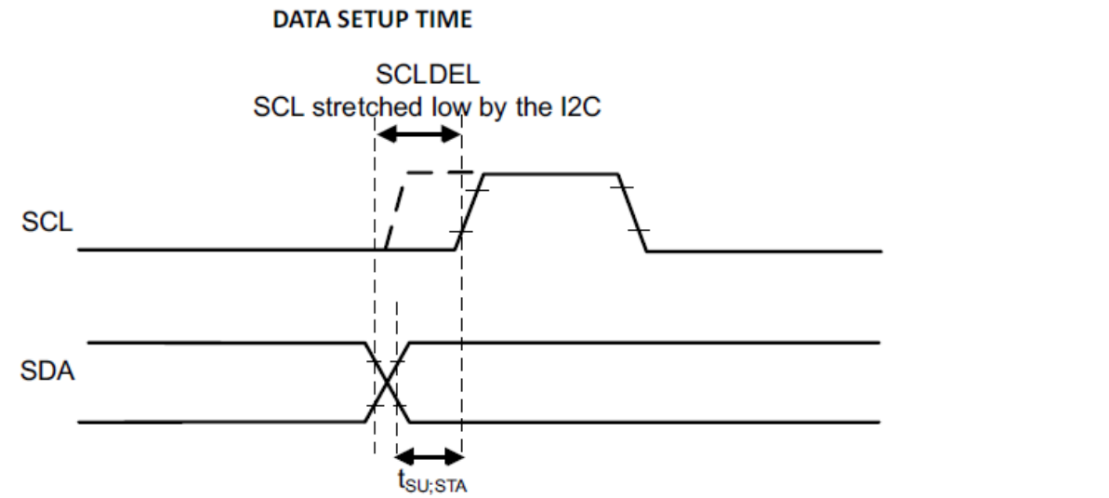
Table 161. Examples of timings settings for f_{I2CCLK} = 16 MHz

Parameter	Standard-mode (Sm)		Fast-mode (Fm)	Fast-mode Plus (Fm+)
	10 kHz	100 kHz	400 kHz	1000 kHz
PRESC	3	3	1	0
SCLL	0xC7	0x13	0x9	0x4
t_{SCLL}	200 x 250 ns = 50 µs	20 x 250 ns = 5.0 µs	10 x 125 ns = 1250 ns	5 x 62.5 ns = 312.5 ns
SCLH	0xC3	0xF	0x3	0x2
t_{SCLH}	196 x 250 ns = 49 µs	16 x 250 ns = 4.0 µs	4 x 125ns = 500 ns	3 x 62.5 ns = 187.5 ns
t_{SCL} (1)	~100 µs(2)	~10 µs(2)	~2500 ns(3)	~1000 ns(4)
SDADEL	0x2	0x2	0x2	0x0
t_{SDADEL}	2 x 250 ns = 500 ns	2 x 250 ns = 500 ns	2 x 125 ns = 250 ns	0 ns
SCLDEL	0x4	0x4	0x3	0x2
t_{SCLDEL}	5 x 250 ns = 1250 ns	5 x 250 ns = 1250 ns	4 x 125 ns = 500 ns	3 x 62.5 ns = 187.5 ns

Figure 287. Setup and hold timings



Data hold time: in case of transmission, the data is sent on SDA output after the SDADEL delay, if it is already available in I2C_TXDR.



Data setup time: in case of transmission, the SCLDEL counter starts when the data is sent on SDA output.

MSv40108V1

30.7.5 Timing register (I2C_TIMINGR)

Address offset: 0x10

Reset value: 0x0000 0000

Access: No wait states

PRESC = 3
SCLDEL = 4
SDADEL = 2
SCLH = 15
SCLL=19

11 0000 0100 0010

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]			
rw								rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]								SCLL[7:0]							
rw								rw							

00001111 00010011

```
I2C2->TIMINGR |= 0x30420F13;          //I2C2 Timing config
```

11000001000010000011110010011

Bits 31:28 **PRESC[3:0]**: Timing prescaler

This field is used to prescale I2CCLK in order to generate the clock period t_{PRESC} used for data setup and hold counters (refer to *I2C timings on page 923*) and for SCL high and low level counters (refer to *I2C master initialization on page 938*).

t_{PRESC} = (PRESC+1) x t_{I2CCLK}

$$t_{PRESC} = (PRESC + 1) * t_{I2CCLK}$$
$$t_{PRESC} = (3 + 1) * 62{,}5ns$$
$$t_{PRESC} = 250ns$$

Bits 23:20 **SCLDEL[3:0]**: Data setup time

This field is used to generate a delay t_{SCLDEL} between SDA edge and SCL rising edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during t_{SCLDEL}.

t_{SCLDEL} = (SCLDEL+1) x t_{PRESC}

Note: t_{SCLDEL} is used to generate $t_{SU:DAT}$ timing.

$$t_{SCLDEL} = (SCLDEL + 1) * t_{PRESC}$$
$$t_{SCLDEL} = (4 + 1) * 250ns$$
$$t_{SCLDEL} = 1{,}25us$$

Bits 19:16 **SDADEL[3:0]**: Data hold time

This field is used to generate the delay t_{SDADEL} between SCL falling edge and SDA edge. In master mode and in slave mode with NOSTRETCH = 0, the SCL line is stretched low during t_{SDADEL}.

t_{SDADEL}= SDADEL x t_{PRESC}

Note: SDADEL is used to generate $t_{HD:DAT}$ timing.

$$t_{SDADEL} = (SDADEL) * t_{PRESC}$$
$$t_{SDADEL} = (2) * 250ns$$
$$t_{SDADEL} = 500ns$$

Bits 15:8 **SCLH[7:0]**: SCL high period (master mode)

This field is used to generate the SCL high period in master mode.

t_{SCLH} = (SCLH+1) x t_{PRESC}

Note: SCLH is also used to generate $t_{SU:STO}$ and $t_{HD:STA}$ timing.

$$t_{SCLH} = (SCLH + 1) * t_{PRESC}$$
$$t_{SCLH} = (15 + 1) * 250ns$$
$$t_{SCLH} = 4us$$

Bits 7:0 **SCLL[7:0]**: SCL low period (master mode)

This field is used to generate the SCL low period in master mode.

t_{SCLL} = (SCLL+1) x t_{PRESC}

Note: SCLL is also used to generate t_{BUF} and $t_{SU:STA}$ timings.

$$t_{SCLL} = (SCLH + 1) * t_{PRESC}$$
$$t_{SCLL} = (19 + 1) * 250ns$$
$$t_{SCLL} = 5us$$

30.7.2 Control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

NBYTES = 2

SADD = 0X7E -> Solo es valido para este ejemplo

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

0 1 0 00000010

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTO END	RE LOAD	NBYTES[7:0]							
					rs	rw	rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD 10R	ADD10	RD_ WRN	SADD[9:0]									
rs	rs	rs	rw	rw	rw	rw									

0 0 1 0 0 0 0001111110

```
I2C2->CR2 |= 0x0202207E;
```

10000000100010000001111110

Bit 13 **START**: Start generation

This bit is set by software, and cleared by hardware after the Start followed by the address sequence is sent, by an arbitration loss, by a timeout error detection, or when PE = 0. It can also be cleared by software by writing '1' to the ADDRCF bit in the I2C_ICR register.

0: No Start generation.

1: Restart/Start generation:

– If the I2C is already in master mode with AUTOEND = 0, setting this bit generates a Repeated Start condition when RELOAD=0, after the end of the NBYTES transfer.

– Otherwise setting this bit will generate a START condition once the bus is free.

Note: Writing '0' to this bit has no effect.

The START bit can be set even if the bus is BUSY or I2C is in slave mode.

This bit has no effect when RELOAD is set. In 10-bit addressing mode, if a NACK is received on the first part of the address, the START bit is not cleared by hardware and the master will resend the address sequence, unless the START bit is cleared by software

Bit 25 **AUTOEND**: Automatic end mode (master mode)

This bit is set and cleared by software.

0: software end mode: TC flag is set when NBYTES data are transferred, stretching SCL low.

1: Automatic end mode: a STOP condition is automatically sent when NBYTES data are transferred.

Note: This bit has no effect in slave mode or when the RELOAD bit is set.

Bits 23:16 **NBYTES[7:0]**: Number of bytes

The number of bytes to be transmitted/received is programmed there. This field is don't care in slave mode with SBC=0.

Note: Changing these bits when the START bit is set is not allowed.

30.7.2 Control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	PEC BYTE	AUTO END	RE LOAD	NBYTES[7:0]							
					rs	rw	rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	STOP	START	HEAD 10R	ADD10	RD_ WRN	SADD[9:0]									
rs	rs	rs	rw	rw	rw	rw									

Bit 11 **ADD10**: 10-bit addressing mode (master mode)

0: The master operates in 7-bit addressing mode,

1: The master operates in 10-bit addressing mode

Note: Changing this bit when the START bit is set is not allowed.

```
// Use 7-bit addresses
I2C2->CR2 &=~ I2C_CR2_ADD10; //Modo maestro receptor, solo recibe 7 de 10 bits de comunicación
// Enable auto-end mode
I2C2->CR2 |= I2C_CR2_AUTOEND; //autofinalización activada
```

30.7.1 Control register 1 (I2C_CR1)

```
// Disable the analog filter
I2C2->CR1 |= I2C_CR1_ANFOFF; //filtro de ruido desactivado
// Disable NOSTRETCH
I2C2->CR1 |= I2C_CR1_NOSTRETCH; //nostretch desactivado
```

Address offset: 0x00

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	Res.	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw		rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw				rw	rw	rw	rw	rw	rw	rw	rw

Bit 17 **NOSTRETCH**: Clock stretching disable

This bit is used to disable clock stretching in slave mode. It must be kept cleared in master mode.

0: Clock stretching enabled

1: Clock stretching disabled

Note: This bit can only be programmed when the I2C is disabled (PE = 0).

Bit 12 **ANFOFF:** Analog noise filter OFF

0: Analog noise filter enabled

1: Analog noise filter disabled

Note: This bit can only be programmed when the I2C is disabled (PE = 0).

```
// Enable the I2Cx peripheral
I2C2->CR1 |= I2C_CR1_PE; //habilita SCL y SDA para el periferico
```

INGENIERÍA MECATRÓNICA UMNG

30.7.1 Control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

Access: No wait states, except if a write access occurs while a write access to this register is ongoing. In this case, wait states are inserted in the second write access until the previous one is completed. The latency of the second write access can be up to 2 x PCLK1 + 6 x I2CCLK.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERT EN	SMBD EN	SMBH EN	GCEN	Res.	NOSTR ETCH	SBC
								rw	rw	rw	rw	rw		rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXDMA EN	TXDMA EN	Res.	ANF OFF	DNF				ERRIE	TCIE	STOP IE	NACK IE	ADDR IE	RXIE	TXIE	PE
rw	rw		rw	rw				rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 **PE**: Peripheral enable

0: Peripheral disable

1: Peripheral enable

Note: When PE=0, the I2C SCL and SDA lines are released. Internal state machines and status bits are put back to their reset value. When cleared, PE must be kept low for at least 3 APB clock cycles.

UNIVERSIDAD MILITAR NUEVA GRANADA

30.7.7 Interrupt and status register (I2C_ISR)

Address offset: 0x18

Reset value: 0x0000 0001

Access: No wait states

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADDCODE[6:0]							DIR
								r							r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY	Res.	ALERT	TIME OUT	PEC ERR	OVR	ARLO	BERR	TCR	TC	STOPF	NACKF	ADDR	RXNE	TXIS	TXE
r		r	r	r	r	r	r	r	r	r	r	r	r	rs	rs

Bit 1 **TXIS**: Transmit interrupt status (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty and the data to be transmitted must be written in the I2C_TXDR register. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to '1' by software when NOSTRETCH=1 only, in order to generate a TXIS event (interrupt if TXIE=1 or DMA request if TXDMAEN=1).

Note: This bit is cleared by hardware when PE=0.

Bit 0 **TXE**: Transmit data register empty (transmitters)

This bit is set by hardware when the I2C_TXDR register is empty. It is cleared when the next data to be sent is written in the I2C_TXDR register.

This bit can be written to '1' by software in order to flush the transmit data register I2C_TXDR.

Note: This bit is set by hardware when PE=0.

Bit 5 **STOPF**: Stop detection flag

This flag is set by hardware when a Stop condition is detected on the bus and the peripheral is involved in this transfer:

– either as a master, provided that the STOP condition is generated by the peripheral.

– or as a slave, provided that the peripheral has been addressed previously during this transfer.

It is cleared by software by setting the STOPCF bit.

Note: This bit is cleared by hardware when PE=0.

```
void I2C2_Write (char Dat){
//I2C2 Initialization
    I2C2->CR2 |= (1UL << 13);  //bit de START  bit 13
    while (!(I2C2->ISR & I2C_ISR_TXIS));
    I2C2->TXDR = LCD_I2C_Address;
    while (!(I2C2->ISR & I2C_ISR_TXIS));
    I2C2->TXDR = Dat;
    while (!(I2C2->ISR & I2C_ISR_TXE));
    while (!(I2C2->ISR & I2C_ISR_STOPF));
    I2C2->ISR &= ~I2C_ISR_STOPF;
}
```

Función para recibir un dato por el modulo de I2C:

```
//FUNCION PARA RECIBIR
char I2C2_Read (char Adr){

  I2C2->CR2 |= (1UL<<13);    //bit de Start
    while (!(I2C2->ISR & I2C_ISR_TXIS));
    I2C2->TXDR = Adr;
    while (!(I2C2->ISR & I2C_ISR_TXE));
    while (!(I2C2->ISR & I2C_ISR_STOPF));
    I2C2->ISR &= ~I2C_ISR_STOPF;
    I2C2->CR2=0;

    return (Dato);
}
```

EJEMPLO

```
#include "stm32f7xx.h"
#include "math.h"
 //A MANERA GENERAL, LA TARJETA ESTÁ TRABAJANDO COMO MAESTRO EN LA COMUNICACIÓN
 //PF1->SCL
 //PF0->SDA
int sa[]={0x01,0x2,0x4,0x8,0x10,0x20,0x40,0x80};                    //2
int co[]={0x40,0x100,0x200,0x400,0x800,0x1000,0x2000,0x8000};      //puerto comun
char dat[]={0,0,0};
char da1;
int x1=0,y1=0;
int t1,t2,t3,t4,t5,t6,cont=0,su=0;
int a=0,b=0,c=0,d=0,e=0,f=0,g=0,h=0;

static void SystemClock_Config(void);
void SystemIn(void);
//void ConfigSerial(void);
long double CTX, CTY, CTZ;
int Status2, Status4, Dato;
long i=0;
int Reg;
int8_t X1,X2,Y1,Y2,Z1,Z2;
uint8_t b2;
short TX,TY,TZ; //TOTAL X, TOTAL Y, TOTAL Z
uint16_t XS, YS, ZS;
char data;
```

```
void I2C2_Init (void)    {
    RCC->AHB1ENR  |=  0x00000020; // Encender reloj puertoF
    GPIOF->AFR[0] |=  0x00000044; // seleccion de la funcion alterna 4 del puerto F(I2C) para  PF1-SCL,PF0-SDA -> I2C2
    GPIOF->MODER  |=  0x0000000A; // PF0,PF1 => en modo alterno
    GPIOF->OTYPER |=  0x0003;     // Open drain
    GPIOF->PUPDR |=  0x5;
    GPIOF->OSPEEDR   |=  0xC;

    RCC->APB1ENR  |= 0x00400000;       // Enable clock for I2C2 bit 22
    RCC->DCKCFGR2  |= 0x80000;                 //Reloj de frecuencia del I2C2

    // Disable the I2Cx peripheral
    I2C2->CR1 &= ~I2C_CR1_PE; //deshabilita SCL y SDA para el periferico
    while (I2C2->CR1 & I2C_CR1_PE);
    // Set timings. Asuming I2CCLK is 50 MHz (APB1 clock source)
    //I2C2->TIMINGR = 0x40912732;        // Discovery BSP code from ST examples
    I2C2->TIMINGR |= 0x30420F13;         //I2C2 Timing config at t2clk=1/FHSI
    // Use 7-bit addresses
    I2C2->CR2 &=~ I2C_CR2_ADD10; //Modo maestro receptor, solo recibe 7 de 10 bits de comunicación
    // Enable auto-end mode
    I2C2->CR2 |= I2C_CR2_AUTOEND; //autofinalización activada
    // Disable the analog filter
    I2C2->CR1 |= I2C_CR1_ANFOFF; //filtro de ruido desactivado
    // Disable NOSTRETCH
    I2C2->CR1 |= I2C_CR1_NOSTRETCH; //nostretch desactivado
    // Enable the I2Cx peripheral
    I2C2->CR1 |= I2C_CR1_PE; //habilita SCL y SDA para el periferico
}
```

```
//FUNCION PARA ENVIAR

void I2C2_Write (char Adr, char Dat){
//I2C2 Initialization
    //ESTE ES EL ORIGINAL 0x0202203C Y SE CAMBIO POR 0x02022025 PERO PUEDE QUE SEA 0x020220D0
    I2C2->CR2 |= 0x020220D0;
    while (!(I2C2->ISR & I2C_ISR_TXIS));
    I2C2->TXDR = Adr;
    while (!(I2C2->ISR & I2C_ISR_TXIS));
    I2C2->TXDR = Dat;
    while (!(I2C2->ISR & I2C_ISR_TXE));
    while (!(I2C2->ISR & I2C_ISR_STOPF));
    I2C2->ISR &= ~I2C_ISR_STOPF;
}
```

```
void I2C2_Lectura(void){
        //REGISTROS EN CONFIGURACION PARA EL GIROSCOPIO
        for (i=0; i<10000; i++) {};   //OJO -> i=0; i<30000; i++
        X1 = I2C2_Read(0x3B);  //CAMBIAR 0X03, POR EL REGISTRO MSB EN X DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};
        X2 = I2C2_Read(0x3C);  //CAMBIAR 0X04, POR EL REGISTRO LSB EN X DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};
        Y1 = I2C2_Read(0x3D);  //CAMBIAR 0X07, POR EL REGISTRO MSB EN Y DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};
        Y2 = I2C2_Read(0x3E);  //CAMBIAR 0X08, POR EL REGISTRO LSB EN Y DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};
        Z1 = I2C2_Read(0x3F);  //CAMBIAR 0X05, POR EL REGISTRO MSB EN Z DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};
        Z2 = I2C2_Read(0x40);  //CAMBIAR 0X06, POR EL REGISTRO LSB EN Z DEL MPU6050 -> CAMBIAR POR ACELEROMETRO
        for (i=0; i<10000; i++) {};

          TX=X1;
          TY=Y1;
          TZ=Z1;
}

void Data_PromptX(void){

            XS = (uint16_t)TX;
    x1=((TX*8)/90)+2;
    GPIOD->ODR = 0XFFFF;
    GPIOD->ODR &=~ sa[x1];
}
```

```
int main ()  {

  RCC->AHB1ENR |= 0x1A;
  GPIOD->MODER |=0X5555;
  GPIOB->MODER |=0X45551000;

    SystemIn();
    SystemClock_Config();

    I2C2_Init();                        // initialize I2C2 block
    RCC->AHB1ENR  |= 0x3F;    // Enable clock for GPIOD
    GPIOD->MODER  |= 0x55555555;

    I2C2_Write(0x6B,0x00);     //LECTURAS
    for (i=0; i<10000; i++) {};
    I2C2_Write(0x1B,0x00);     //CONFIG_GYRO
    for (i=0; i<10000; i++) {};
    I2C2_Write(0x1C,0x00);     //CONFIG_ACELE
    for (i=0; i<10000; i++) {};

  while(1) {

        I2C2_Lectura();
        Data_PromptX();
        Data_PromptY();
      for (i=0; i<10000; i++) {};    // tiempo de mas o menos 100ms
  }
}
```

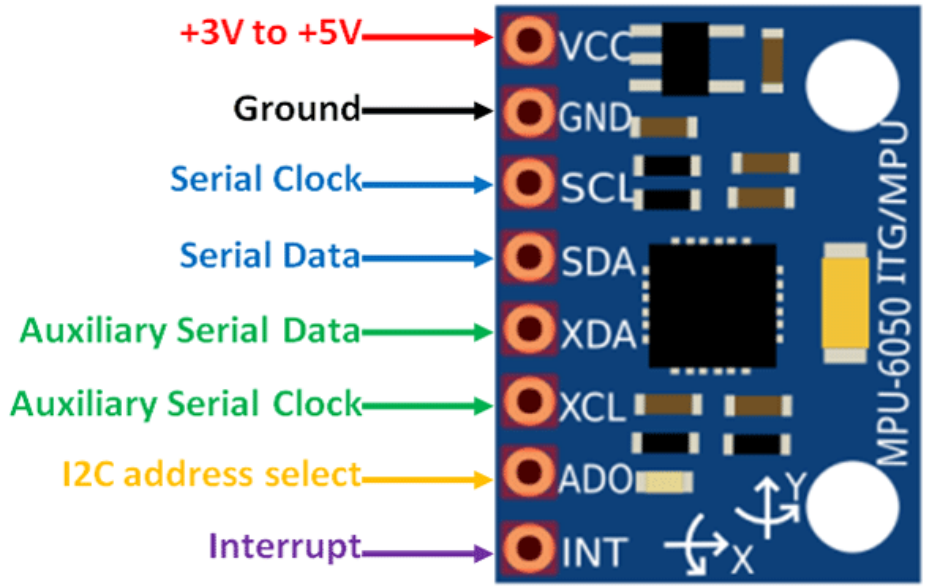
```
static void SystemClock_Config(void)
{
  RCC->CR |= ((uint32_t)RCC_CR_HSION);                      /* Enable HSI */
  while ((RCC->CR & RCC_CR_HSIRDY) == 0);                   /* Wait for HSI Ready */

  RCC->CFGR = RCC_CFGR_SW_HSI;                              /* HSI is system clock */
  while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_HSI);   /* Wait for HSI used as system clock */
  RCC->CFGR |= (RCC_CFGR_HPRE_DIV1  |                       /* HCLK = SYSCLK */
                RCC_CFGR_PPRE1_DIV1 |                       /* APB1 = HCLK/2 */
                RCC_CFGR_PPRE2_DIV1  );                     /* APB2 = HCLK/1 */
  RCC->CR &= ~RCC_CR_PLLON;                                 /* Disable PLL */
}

void SystemIn(void){
 int a=0;
 RCC->CR |= 0x10000;
 while((RCC->CR & 0x20000)==0);
 RCC->APB1ENR = 0x10080000;
 RCC->CFGR = 0x00009400;
 RCC->PLLCFGR = 0x07405408;
 RCC->CR |= 0x01000000;
 while((RCC->CR & 0x02000000)==0);
 FLASH->ACR = (0x00000605);
 RCC->CFGR |= 2;
 for (a=0;a<=500;a++);
}
```

SENSOR MPU6050

EL MPU6050 es una unidad de medición IMU de 6 grados de libertad pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, ya que con él es posible realizar distintas mediciones como goniometría, estabilización, temperatura. es un sensor de movimiento, que tiene un conversor ADC de 16 bits que convierte los datos a un valor digital, el módulo se comunica a través de la comunicación serie I2C a través del reloj serial (SCL) y datos (SDA).

Necesita 3.3V pero un regulador de voltaje en la tarjeta GY-521 le permite alimentarlo hasta 5V.

Como la comunicación del módulo es vía I2C, esto le permite trabajar con la mayoría de microcontroladores. En el módulo los pines SCL y SDA tienen una resistencia pull-up en la placa para una conexión directa al microcontrolador que estemos utilizando. Las características adicionales incluyen un oscilador en chip con una variación de ± 1% sobre el rango de temperatura de funcionamiento.



6.3 Electrical and Other Common Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	Units	Notes
TEMPERATURE SENSOR						
Range			-40 to +85		°C	
Sensitivity	Untrimmed		340		LSB/ºC	
Temperature Offset	35°C		-521		LSB	
Linearity	Best fit straight line (-40°C to +85°C)		±1		°C	
VDD POWER SUPPLY						
Operating Voltages		2.375		3.46	V	
Normal Operating Current	Gyroscope + Accelerometer + DMP		3.9		mA	
	Gyroscope + Accelerometer (DMP disabled)		3.8		mA	
	Gyroscope + DMP (Accelerometer disabled)		3.7		mA	
	Gyroscope only (DMP & Accelerometer disabled)		3.6		mA	
	Accelerometer only (DMP & Gyroscope disabled)		500		µA	
Accelerometer Low Power Mode Current	1.25 Hz update rate		10		µA	
	5 Hz update rate		20		µA	
	20 Hz update rate		70		µA	
	40 Hz update rate		140		µA	
Full-Chip Idle Mode Supply Current			5		µA	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			100	ms	
VLOGIC REFERENCE VOLTAGE	MPU-6050 only					
Voltage Range	VLOGIC must be ≤VDD at all times	1.71		VDD	V	
Power Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value			3	ms	
Normal Operating Current			100		µA	
TEMPERATURE RANGE						
Specified Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range	-40		+85	°C	

SENSOR MPU6050

Los datos dentro del conjunto de registros internos de los sensores siempre se actualizan a la frecuencia de muestreo. Mientras tanto, el conjunto de registros de lectura orientados al usuario duplica los valores de datos del conjunto de registros internos. Esto garantiza que una lectura de ráfaga de los registros del sensor leerá las mediciones al mismo instante de muestreo.

Cada medición es de 16 bits una escala definida.

Las mediciones de temperatura se escriben en estos registros a la frecuencia de muestreo como se define en el Registro 25.

Registro 25 - Divisor de frecuencia de muestreo SMPRT_DIV

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV[7:0]							

Este registro especifica el divisor de la velocidad de salida del sensor utilizado para generar la frecuencia de muestreo para el MPU-60X0.

La frecuencia de muestreo se genera dividiendo la frecuencia de salida del sensor por SMPLRT_DIV:

Frecuencia de muestreo = Velocidad de salida del sensor / (1 + SMPLRT_DIV)

SENSOR MPU6050

donde la velocidad de salida del sensor = 8kHz cuando el DLPF está deshabilitado (DLPF_CFG = 0 o 7), y 1kHz cuando el DLPF está habilitado

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Registros 65 y 66 - Medición de temperatura TEMP_OUT_H y TEMP_OUT_L

Tipo: solo lectura:

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

La temperatura en grados C para un valor de registro dado se puede calcular como:

Temperatura °C = (Valor de registro TEMP_OUT) / 340 + 36.53

"Tenga en cuenta que las matemáticas en la ecuación anterior están en decimal."

SENSOR MPU6050

Registros 104 – Signal Path Reset SIGNAL_PATH_RESET

Este registro se utiliza para restablecer las rutas de señal analógica y digital de los sensores de giroscopio, acelerómetro y temperatura.

El reinicio revertirá la ruta de señal de los convertidores y filtros analógicos a digitales a sus configuraciones de encendido.

"Este registro no borra los registros del sensor." Los bits 7 a 3 están reservados.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
68	104	-	-	-	-	-	GYRO _RESET	ACCEL _RESET	TEMP _RESET

GYRO_RESET: cuando se establece en 1, este bit restablece las rutas de señal digital y analógica del giroscopio.

ACCEL_RESET: cuando se establece en 1, este bit restablece las rutas de señal analógica y digital del acelerómetro.

T**EMP_RESET:** cuando se establece en 1, este bit restablece las rutas de señal analógica y digital del sensor de temperatura.

```c
#include "stm32f7xx.h"
#include "math.h"
#include "lcd.c"

 //A MANERA GENERAL, LA TARJETA ESTÁ TRABAJANDO COMO MAESTRO EN LA COMUNICACIÓN
 //PF1->SCL
 //PF0->SDA
char dat[]={0,0,0};
float temp1=0;
int temp2=0;
int unidades;
int miles;
int centenas;
int decenas;
int a=0,b=0,c=0;

static void SystemClock_Config(void);
void SystemIn(void);
//void ConfigSerial(void);
int Dato;
long i=0;
int8_t T1,T2;
//float T1;
short TT; //TOTAL Temperatura
int16_t TS;
char data;

int vec[5]={84,101,109,112,58};
void I2C2_Init (void)   {
    RCC->AHB1ENR  |=  0x00000020; // Encender reloj puertoF
    GPIOF->AFR[0] |=  0x00000044; // seleccion de la funcion alterna 4 del puerto F(I2C) para  PF1-SCL,PF0-SDA -> I2C2
    GPIOF->MODER  |=  0x0000000A; // PF0,PF1 => en modo alterno
    GPIOF->OTYPER |=  0x0003;     // Open drain
```

```
34      GPIOF->PUPDR |=  0x5;
35      GPIOF->OSPEEDR   |=  0xC;
36
37      RCC->APB1ENR  |= 0x00400000;        // Enable clock for I2C2 bit 22
38      RCC->DCKCFGR2  |= 0x80000;                  //Reloj de frecuencia del I2C2
39
40      // Disable the I2Cx peripheral
41      I2C2->CR1 &= ~I2C_CR1_PE; //deshabilita SCL y SDA para el periferico
42      while (I2C2->CR1 & I2C_CR1_PE);
43      // Set timings. Asuming I2CCLK is 50 MHz (APB1 clock source)
44      //I2C2->TIMINGR = 0x40912732;         // Discovery BSP code from ST examples
45      I2C2->TIMINGR |= 0x30420F13;          //I2C2 Timing config at t2clk=1/FHSI
46      // Use 7-bit addresses
47      I2C2->CR2 &=~ I2C_CR2_ADD10; //Modo maestro receptor, solo recibe 7 de 10 bits de comunicación
48      // Enable auto-end mode
49      I2C2->CR2 |= I2C_CR2_AUTOEND; //autofinalización activada
50      // Disable the analog filter
51      I2C2->CR1 |= I2C_CR1_ANFOFF; //filtro de ruido desactivado
52      // Disable NOSTRETCH
53      I2C2->CR1 |= I2C_CR1_NOSTRETCH; //nostretch desactivado
54      // Enable the I2Cx peripheral
55      I2C2->CR1 |= I2C_CR1_PE; //habilita SCL y SDA para el periferico
56  }
57
58  //FUNCION PARA ENVIAR
59
60  void I2C2_Write (char Adr, char Dat){
61  //I2C2 Initialization
62      //ESTE ES EL ORIGINAL 0x0202203C Y SE CAMBIO POR 0x02022025 PERO PUEDE QUE SEA 0x020220D0
63      I2C2->CR2 |= 0x020220D0;
64      while (!(I2C2->ISR & I2C_ISR_TXIS));
65      I2C2->TXDR = Adr;
66      while (!(I2C2->ISR & I2C_ISR_TXIS));
67      I2C2->TXDR = Dat;
```

```
67      I2C2->TXDR = Dat;
68      while (!(I2C2->ISR & I2C_ISR_TXE));
69      while (!(I2C2->ISR & I2C_ISR_STOPF));
70      I2C2->ISR &= ~I2C_ISR_STOPF;
71  }
72
73  //FUNCION PARA RECIBIR
74  int8_t I2C2_Read (char Adr){    // procedure: RM0090, pg. 584!
75      //ESTE ES EL ORIGINAL 0x0201203C; Y SE CAMBIO POR 0x02022025 PERO PUEDE QUE SEA 0x020220D0
76      I2C2->CR2 |= 0x020120D0;
77      while (!(I2C2->ISR & I2C_ISR_TXIS));
78      I2C2->TXDR = Adr;
79      while (!(I2C2->ISR & I2C_ISR_TXE));
80      while (!(I2C2->ISR & I2C_ISR_STOPF));
81      I2C2->ISR &= ~I2C_ISR_STOPF;
82      I2C2->CR2=0;
83
84      // Disable the I2Cx peripheral
85      I2C2->CR1 &= ~I2C_CR1_PE;
86      while (I2C2->CR1 & I2C_CR1_PE);
87      // Set timings. Asuming I2CCLK is 50 MHz (APB1 clock source)
88      //I2C2->TIMINGR = 0x40912732;         // Discovery BSP code from ST examples
89      I2C2->TIMINGR |= 0x30420F13;              //I2C2 Timing config at t2clk=1/FHSI
90      // Use 7-bit addresses
91      I2C2->CR2 &=~ I2C_CR2_ADD10;
92      // Enable auto-end mode
93      I2C2->CR2 |= I2C_CR2_AUTOEND;
94      // Disable the analog filter
95      I2C2->CR1 |= I2C_CR1_ANFOFF;
96      // Disable NOSTRETCH
97      I2C2->CR1 |= I2C_CR1_NOSTRETCH;
98      // Enable the I2Cx peripheral
99      I2C2->CR1 |= I2C_CR1_PE;
```

```
101     //ESTE ES EL ORIGINAL 0x0201243C; Y SE CAMBIO POR 0x02022025 PERO PUEDE QUE SEA 0x020220D0
102     I2C2->CR2 |= 0x020124D1;
103     while (!(I2C2->ISR & I2C_ISR_RXNE));
104     Dato = I2C2->RXDR;
105     while (!(I2C2->ISR & I2C_ISR_STOPF));
106     I2C2->ISR &= ~I2C_ISR_STOPF;
107     I2C2->CR2=0;
108
109     // Disable the I2Cx peripheral
110     I2C2->CR1 &= ~I2C_CR1_PE;
111     while (I2C2->CR1 & I2C_CR1_PE);
112     // Set timings. Asuming I2CCLK is 50 MHz (APB1 clock source)
113     //I2C2->TIMINGR = 0x40912732;        // Discovery BSP code from ST examples
114     I2C2->TIMINGR |= 0x30420F13;            //I2C2 Timing config at t2clk=1/FHSI
115     // Use 7-bit addresses
116     I2C2->CR2 &=~ I2C_CR2_ADD10;
117     // Enable auto-end mode
118     I2C2->CR2 |= I2C_CR2_AUTOEND;
119     // Disable the analog filter
120     I2C2->CR1 |= I2C_CR1_ANFOFF;
121     // Disable NOSTRETCH
122     I2C2->CR1 |= I2C_CR1_NOSTRETCH;
123     // Enable the I2Cx peripheral
124     I2C2->CR1 |= I2C_CR1_PE;
125
126     return (Dato);
127 }
128 void I2C2_Lectura(void){
129         T1 = I2C2_Read(0x41);
130         for (i=0; i<10000; i++) {};
131         T2 = I2C2_Read(0x42);
132         for (i=0; i<10000; i++) {};
```

```
135             TT=T1*340;
136             //TT=T1;
137   }
138
139   void Temp_Prompt(void){
140
141     TS=(int16_t)TT;
142     temp1=((TS/340)+36.53);
143
144     temp2=temp1*100;
145
146      miles=temp2/1000;
147      centenas=(temp2-(miles*1000))/100;
148      decenas=(temp2- (miles*1000 + centenas*100))/10;
149      unidades=temp2-(miles*1000 + centenas*100 + decenas*10 );
150
151
152   }
153 int main ()  {
154
155    RCC->AHB1ENR |= 0x1A;
156    GPIOB->MODER |=0X45551000;
157    lcdstart();
158      SystemIn();
159      SystemClock_Config();
160
161      I2C2_Init();                    // initialize I2C2 block
162      RCC->AHB1ENR  |= 0x3F;    // Enable clock for GPIOD
163
164      I2C2_Write(0x6B,0x00);
165      for (i=0; i<10000; i++) {};
166      I2C2_Write(0x68,0x01);
167      for (i=0; i<10000; i++) {};
```

```
164       I2C2_Write(0x6B,0x00);
165       for (i=0; i<10000; i++) {};
166       I2C2_Write(0x68,0x01);
167       for (i=0; i<10000; i++) {};
168
169       frase(vec,5);
170
171     while(1) {
172
173           I2C2_Lectura();
174           Temp_Prompt();
175
176         for (i=0; i<10000; i++) {};    // tiempo de mas o menos 100ms
177
178           posd(2);
179           numero(miles);
180           numero(centenas);
181           letra(46);
182           numero(decenas);
183           numero(unidades);
184           letra(67);
185
186     }
187 }
188
189
```