MICROS 32 BITS
STM – I2C

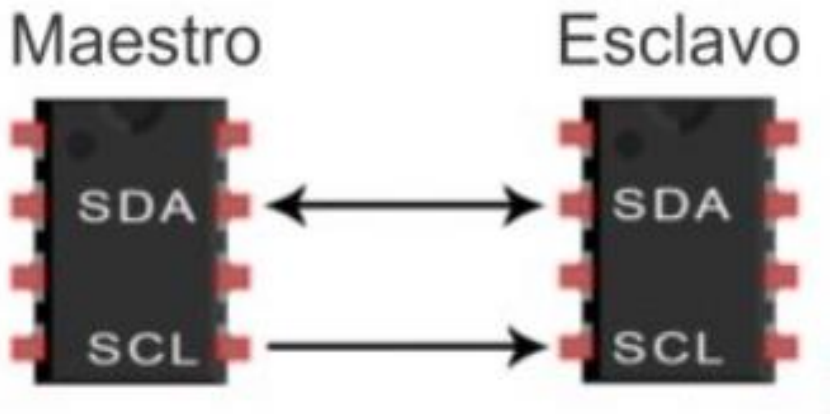
ROBINSON JIMENEZ MORENO

UNIVERSIDAD MILITAR
NUEVA GRANADA

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.



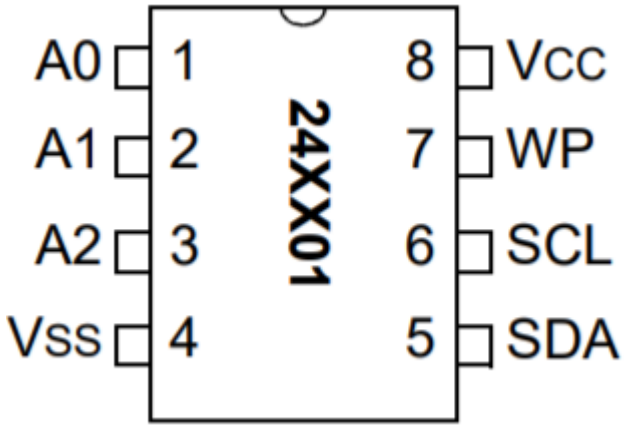
SDA – Serial Data. Es la vía de comunicación entre el maestro y el esclavo para enviarse información.
SCL – Serial Clock. Es la vía por donde viaja la señal de reloj.

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.

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

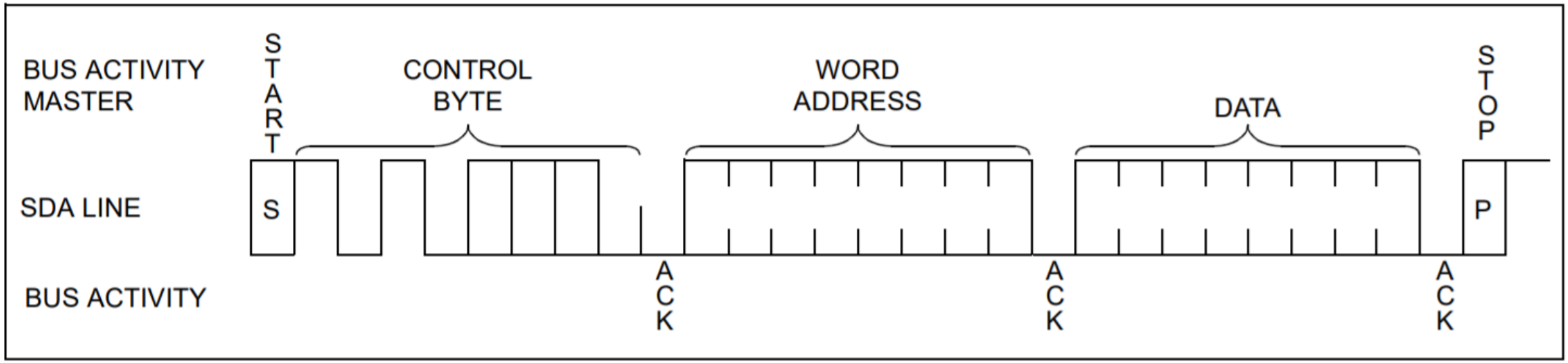
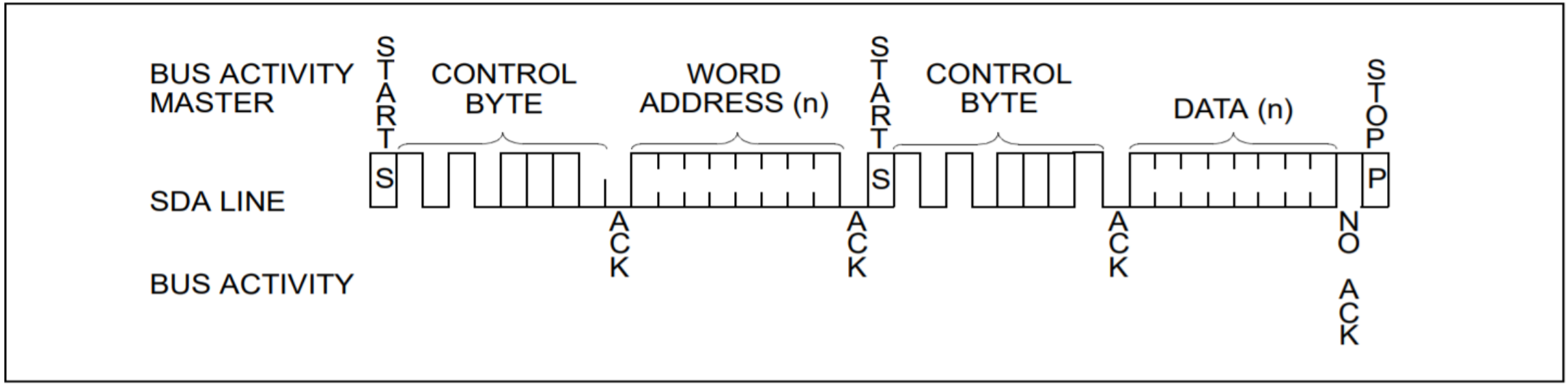
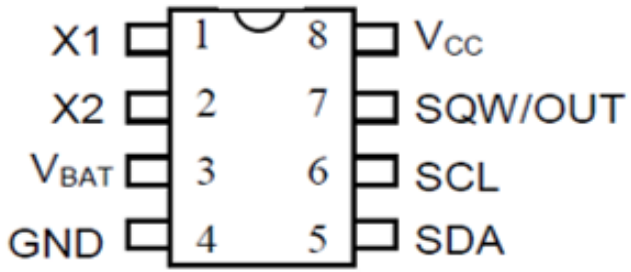


FIGURE 7-2: RANDOM READ



Reloj En Tiempo Real (RTC)

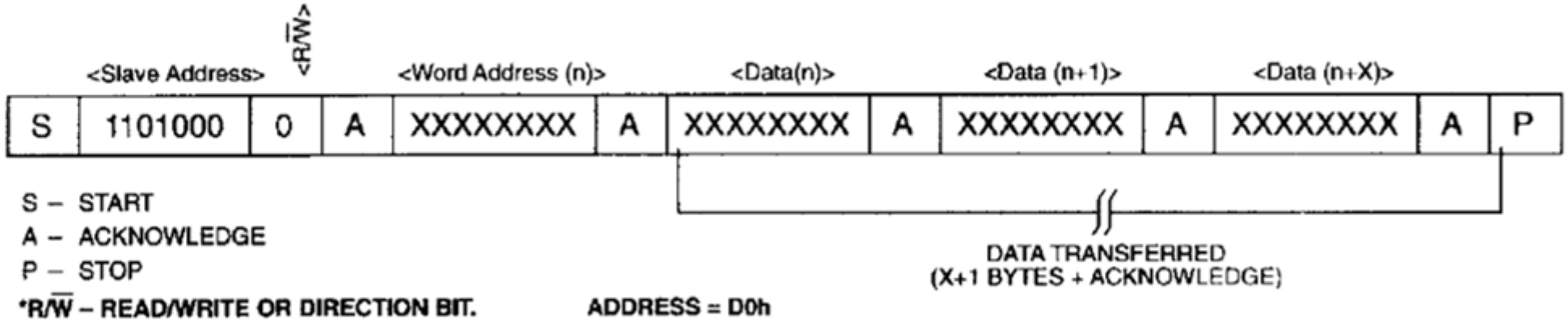
Otro dispositivo común para uso del protocolo I2C es el reloj en tiempo real de referencia DS107, que requiere un oscilador externo en los pines X1 y X2 para manejar la temporización, como se aprecia en la siguiete figura.



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

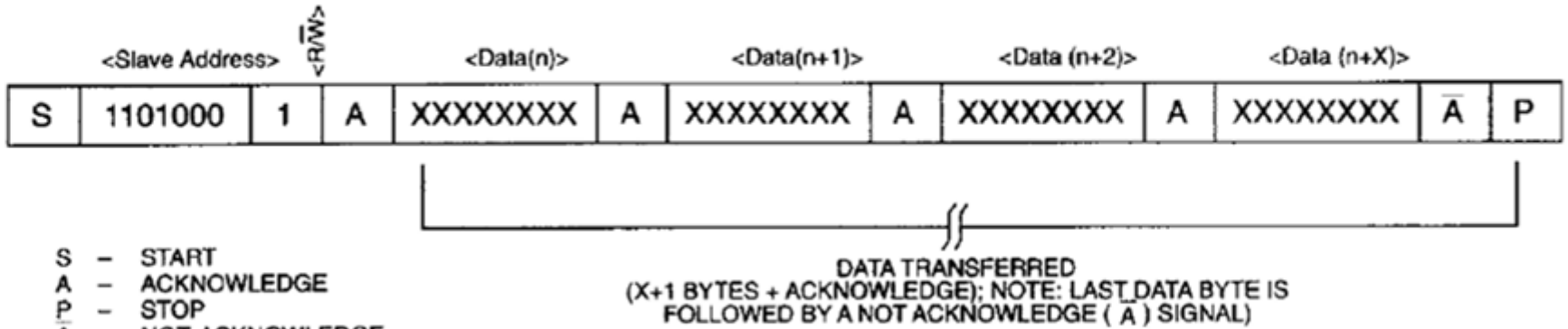


S – START
A – ACKNOWLEDGE
P – STOP

*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.

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.

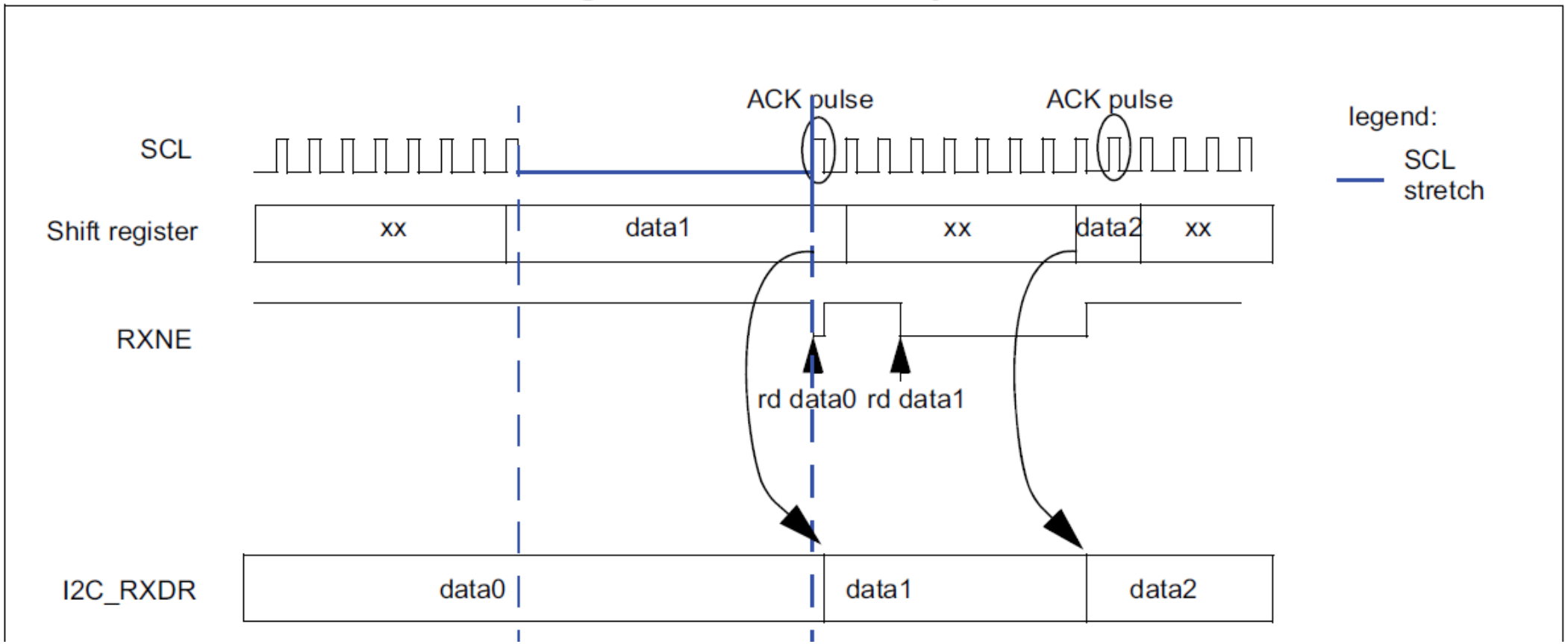
In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the General Call address. The General Call address detection can be enabled or disabled by software. The reserved SMBus addresses can also be enabled by software.

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).

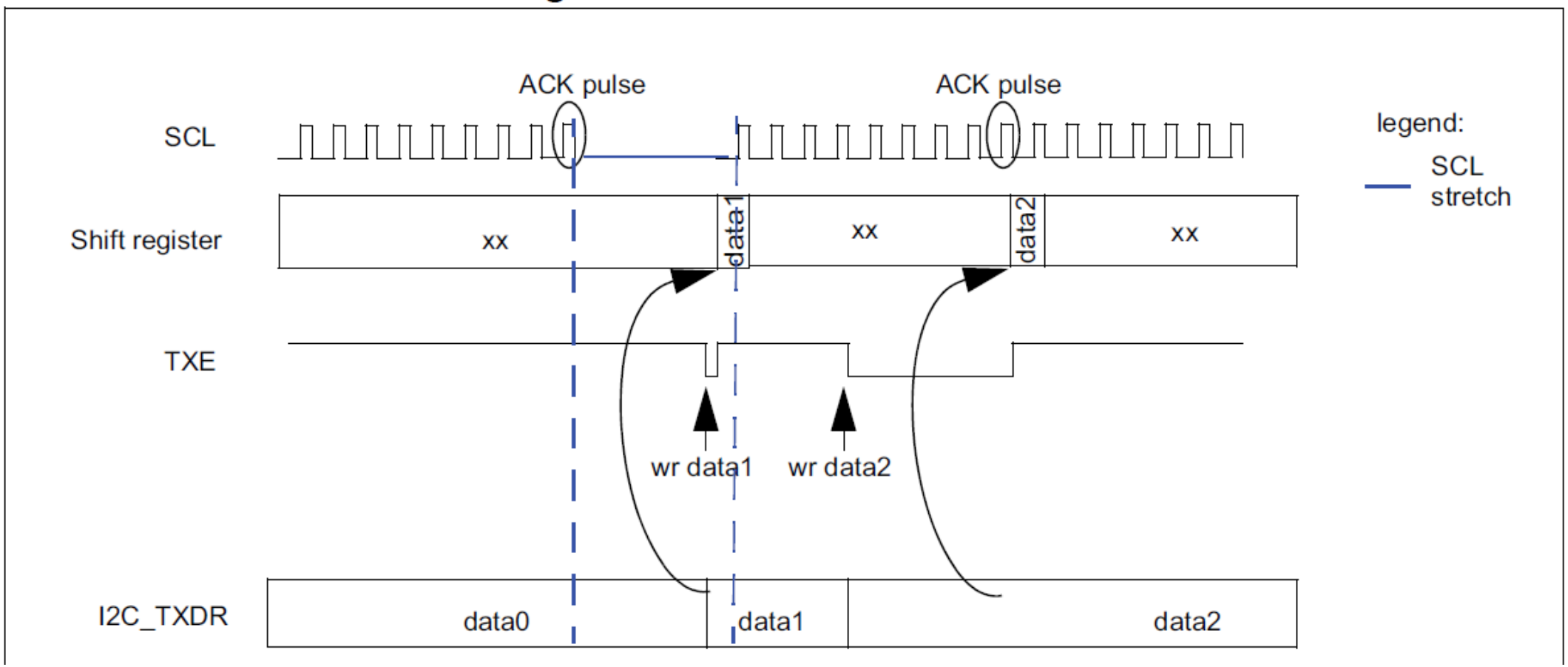
Figure 307. Data reception



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



```c
#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];
 }
```

```c
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
  }
}
```

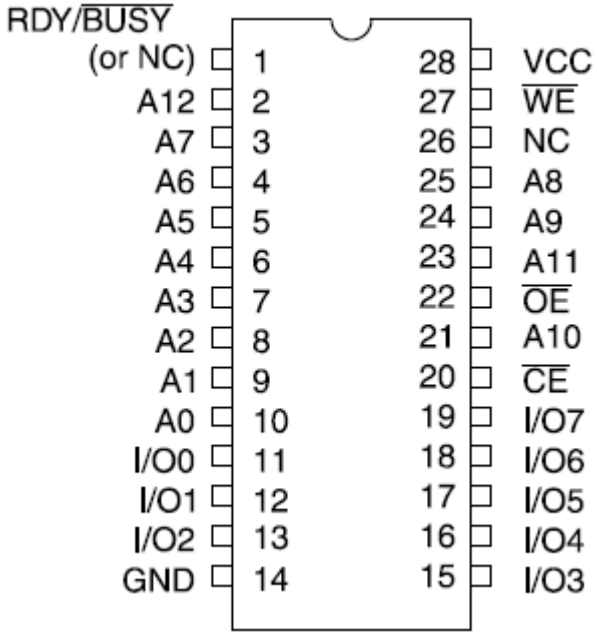
```
static void SystemClock_Config(void)
{
  RCC->CR |= ((uint32_t)RCC_CR_HSION);                     /* Enable HSI */
  while ((RCC->CR & RCC_CR_HSIRDY) == 0);                  /* Wait for HSI Ready */

  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++);
}
```

TALLER PRE EXAMEN

Programar un microcontrolador para almacenar los 100 últimos datos de temperatura de tres sensores lm35 en una memoria AT28c64

Operating Modes

Mode	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	I/O
Read	V_{IL}	V_{IL}	V_{IH}	D_{OUT}
Write (2)	V_{IL}	V_{IH}	V_{IL}	D_{IN}
Standby/Write Inhibit	V_{IH}	X (1)	X	High Z
Write Inhibit	X	X	V_{IH}	
Write Inhibit	X	V_{IL}	X	
Output Disable	X	V_{IH}	X	High Z
Chip Erase	V_{IL}	V_H (3)	V_{IL}	High Z



Un sistema de control de nivel de un tanque emplea una electroválvula comandada por servomotor y un sensor de nivel por ultrasonido HCSR04. Se recibe el nivel deseado por puerto serial, de forma que la electroválvula debe actuar por una señal PWM cuyo ciclo útil se calcula mediante la ecuación:

$$u(k) = u(k-1) + Ae(k) + Be(k-1) + Ce(k-2)$$

Donde A vale 1,8 B vale 1,2 y c vale 0,54. e(k) corresponde a la diferencia o error entre el nivel actual y el recibido. e(k-1) al error una muestra antes y e(k-2) al error dos muestras antes.