Diseño Basado en Microprocesadores

Tema 2. Microcontroladores

- 2.1. Introducción a los microcontroladores
- 2.8. Entradas/Salidas Digitales
- 2.3. Temporizadores
- 2.4. Excepciones
- 2.5. Conversión Analógica/Digital
- 2.6. Comunicación serie RS232C
- 2.7. Teclado, conversión D/A y sonido
- 2.8. Interfaz I2C

2.7. Interfaz I^2C

UM10562 Cap. 22

- 2.8.1. Interfaz I^2C.
- 2.8.2. Características
- 2.8.3. Terminología
- 2.8.4. Señales
- 2.8.5. Arbitraje
- 2.8.6. Transferencias
- 2.8.7. Cálculo de resistencias de pull-up
- 2.8.8. I^2C en el LPC40xx
- 2.8.9. Arquitectura del I^2C
- 2.8.10. Registros
- 2.8.11. Pasos para la programación
- 2.8.12. Ejemplo programación

2.8.1. Interfaz I^2C

- I2C (también I2C) significa “Inter-Integrated Circuit”
- Bus de comunicación serie para interconectar varios circuitos integrados dentro de una misma PCB
- Desarrollado por Philips (ahora NXP) a comienzos de la década de 1980
- Hoy en día lo usan multitud de fabricantes y existe una gran variedad de dispositivos compatibles: EEPROM, RTC, LCD, ADC, DAC, etc.
- Algunos fabricantes lo nombran Two Wire Interface, (TWI)

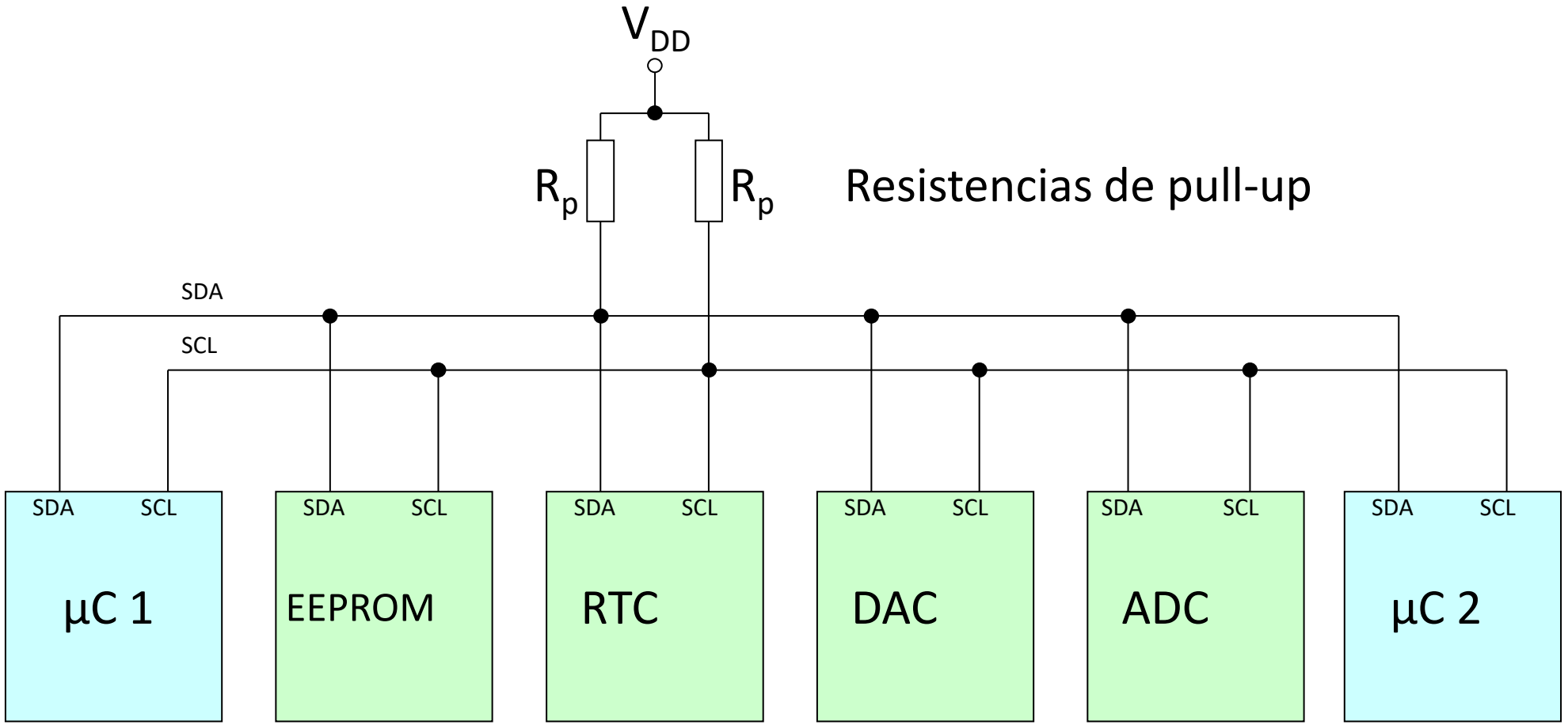
2.8.2. Características I2C (1)

- Sólo son necesarias dos líneas de bus (además de GND):
  - Serial Data (SDA)
  - Serial Clock (SCL)
- **Cada dispositivo** conectado al bus es direccionable mediante una **dirección propia**
- Durante la comunicación existe siempre una relación maestro-esclavo
- Los maestros pueden operar como maestros transmisores o maestros receptores

2.8.2. Características I2C (2)

- Es un bus con posibilidad multimaestro que incluye detección de colisiones y arbitraje para impedir la corrupción de datos si dos o más maestros inician la comunicación simultáneamente
- Velocidad de comunicación:
  - Standard-mode: hasta 100 kbit/s
  - Fast-mode: hasta 400 kbit/s
  - Fast-mode Plus: hasta 1 Mbit/s
  - High-speed mode: hasta 3.4 Mbit/s
  - Ultra-Fast mode: 5 Mbit/s
- El número de dispositivos conectados al bus está limitado fundamentalmente por la máxima capacidad en las líneas del bus (400 pF)

2.8.2. Ejemplo de configuración



2.8.2. Ventajas e inconvenientes

Ventajas

- Permite que los circuitos integrados tengan menos pines y encapsulados más pequeños
- Reduce la complejidad de la interconexión entre circuitos integrados
- El trazado de pistas de la PCB resulta más sencillo
- La capacidad de direccionar cada dispositivo elimina la necesidad de líneas de selección individuales

Inconvenientes

- Baja tasa máxima de comunicación (menor que SPI)

2.8.3. Terminología

Transmisor
- Dispositivo que envía datos a través del bus

Receptor
- Dispositivo que recibe datos a través del bus

Maestro
- Dispositivo que inicia la transferencia, genera la señal de reloj y termina la transferencia

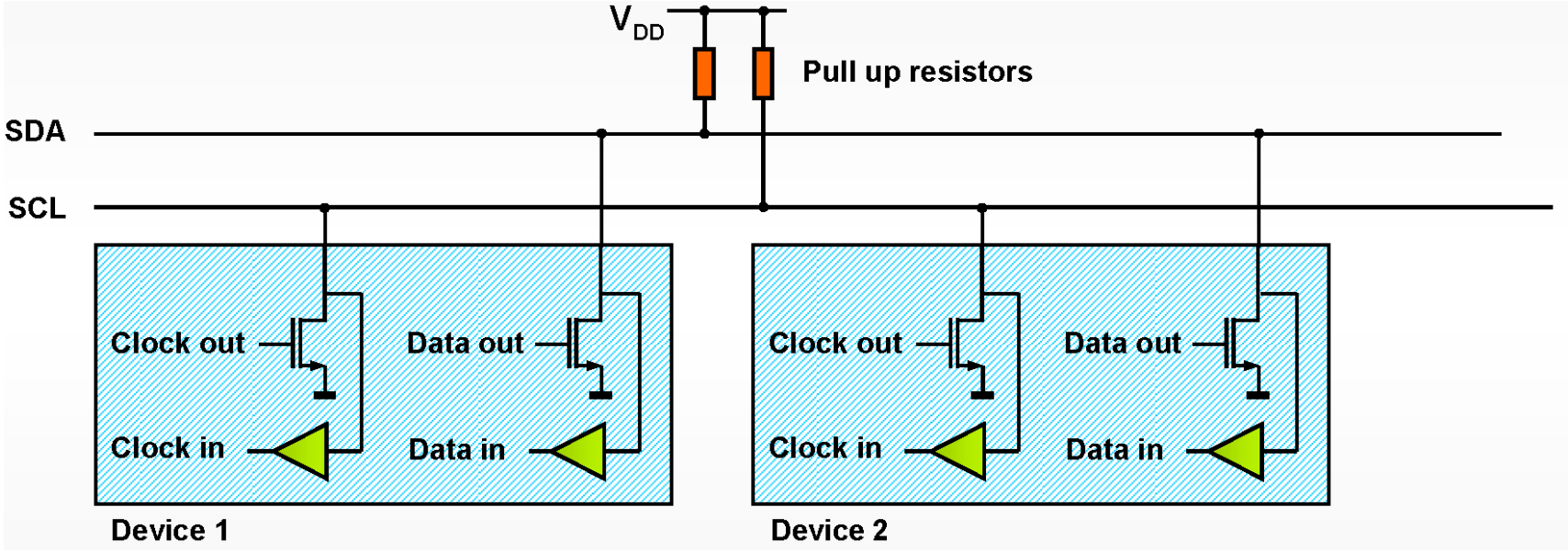
Esclavo
- Dispositivo direccionado por el maestro

Multimaestro
- Varios maestros pueden intentar controlar el bus al mismo tiempo. En estos casos entra en acción un mecanismo de arbitraje

Arbitraje
- Procedimiento que asegura que en caso de que varios maestros intenten controlar el bus simultáneamente sólo uno lo consiga sin que el mensaje que éste intentaba enviar se corrompa

Sincronización
- Procedimiento para sincronizar la señal de reloj de dos o más dispositivos

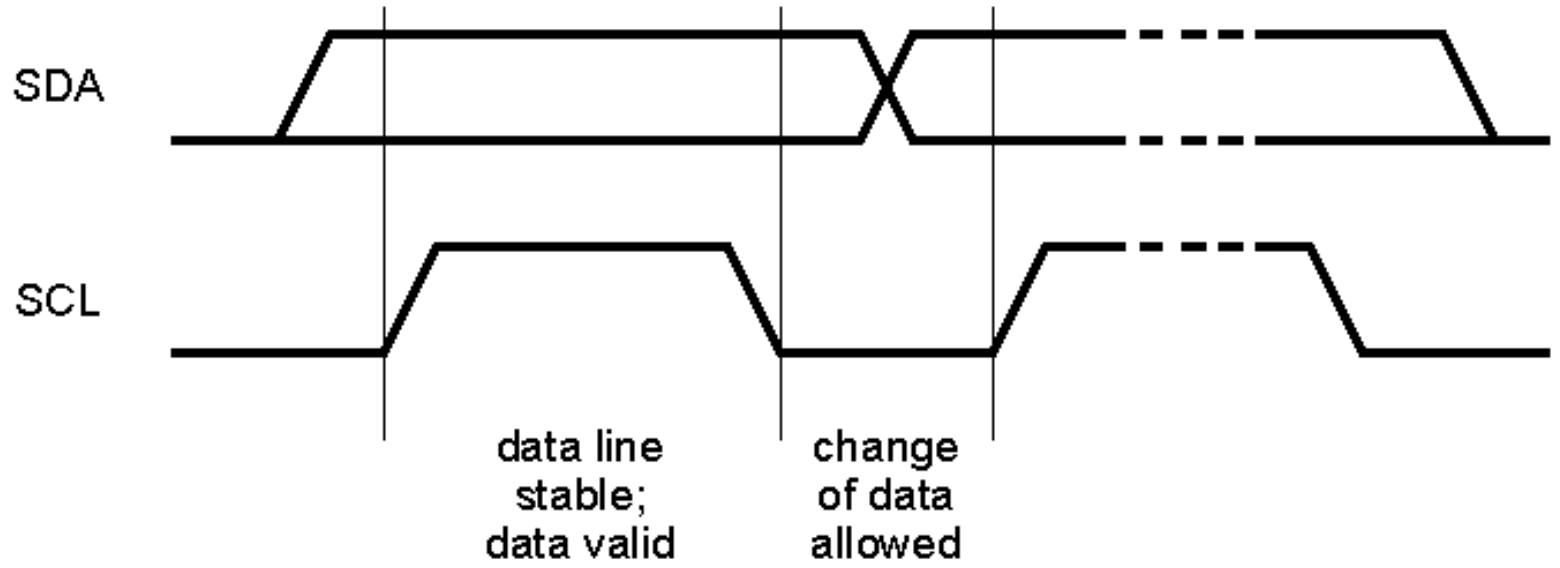
2.8.3. Circuitos internos para SDA y SCL en los dispositivos



- Las salidas en colector/drenador abierto producen una función “AND cableada”
  - SDA/SCL se pone a cero con tal de que uno de los dispositivos conectados al bus las fuerce a cero
  - SDA/SCL se ponen a uno sólo si ninguno de los dispositivos la fuerza a nivel bajo
  - Es la base de los mecanismos de arbitraje y sincronización
- Cuando el bus está inactivo (libre) tanto SDA como SCL están a nivel alto

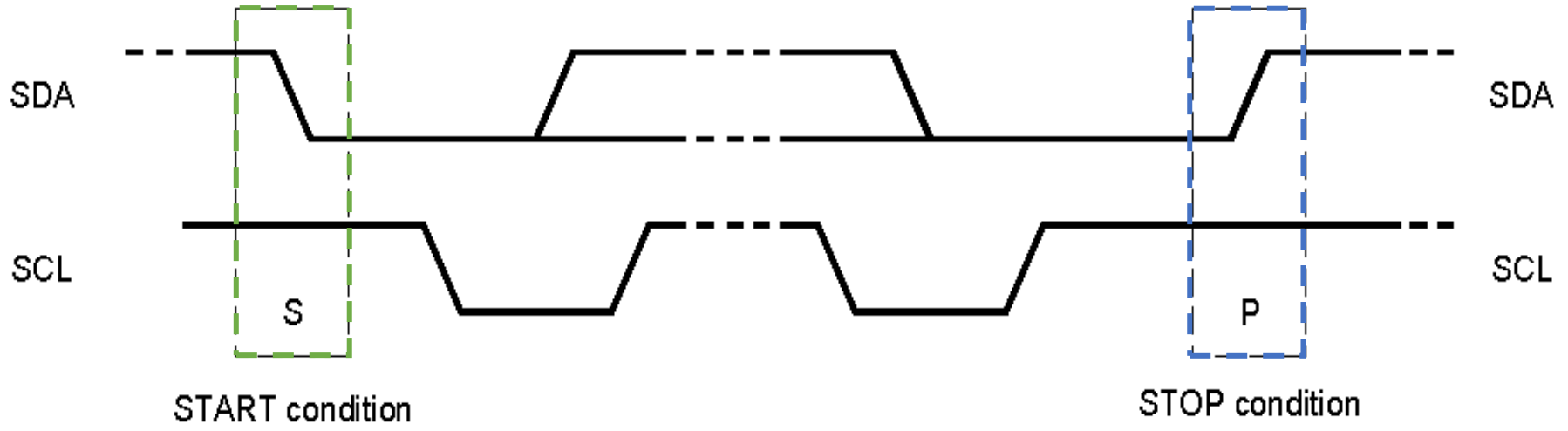
2.8.4. Señales. Transferencias de bits

Durante las transferencias de bits, la línea SDA sólo puede cambiar cuando la señal SCL está a nivel bajo, si SCL está a nivel alto SDA permanece estable



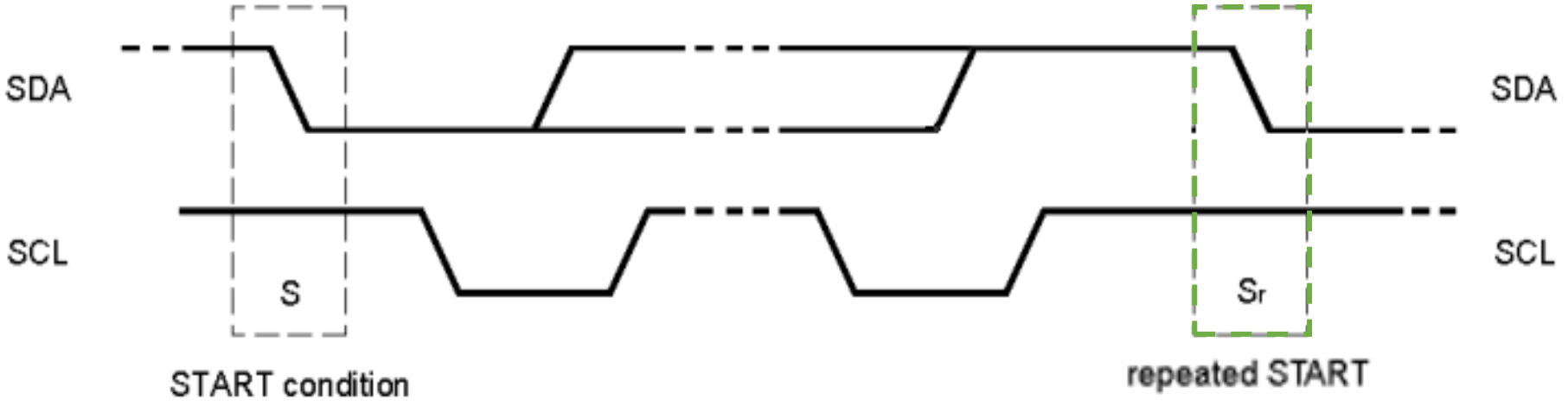
2.8.4. Señales. Condiciones de Start y Stop

- Una transacción se produce entre una condición de START y una de STOP. Estas se generan mediante cambios en la línea SDA mientras la línea SCL está a nivel alto
- Las condiciones de START y STOP son generadas por el maestro del bus
- El bus se considera ocupado después de que se produzca una condición de START y hasta que se genere la condición de STOP



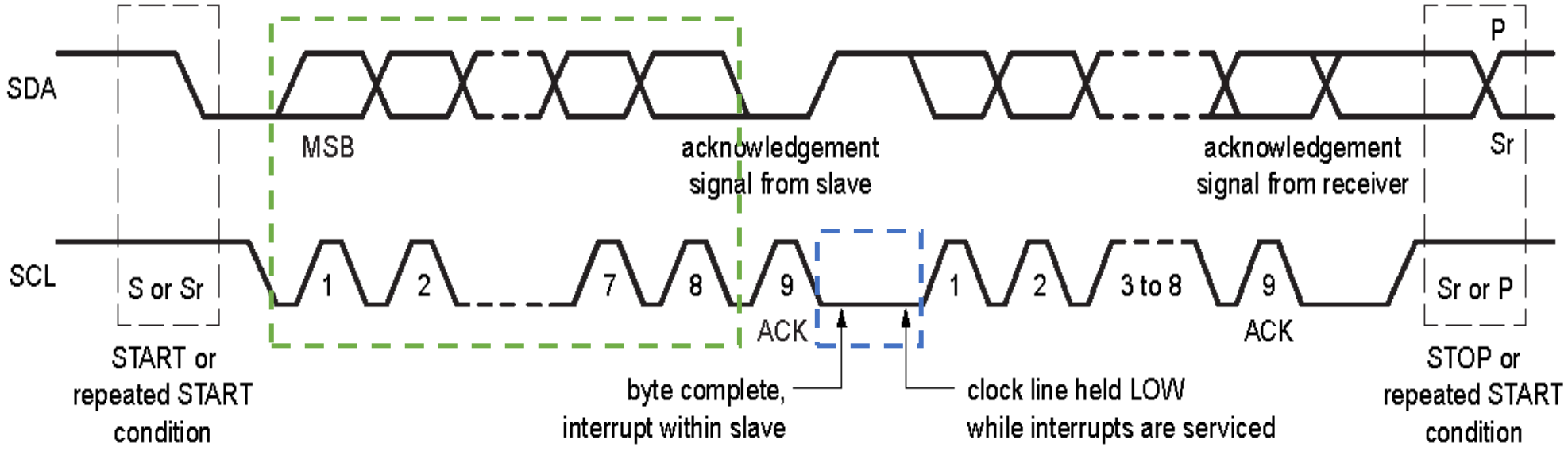
2.8.4. Señales. Condición de Repeated Start

- Un maestro puede concatenar transferencias seguidas generando una condición de START repetida o REPEATED START. (Esto es más rápido que un STOP y un START)



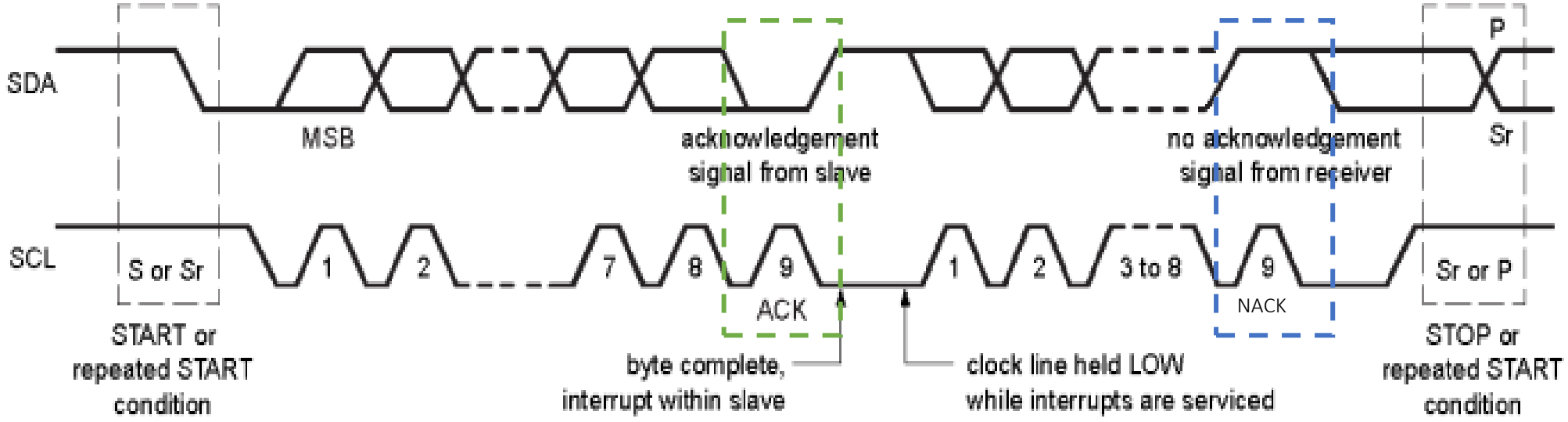
2.8.4. Señales. Formato de los bytes

- En la transmisión los datos se intercambian siempre en bytes (8 bits).
- Se envía primero el bit más significativo (MSB) de cada byte.
- No hay límite al número de bytes que pueden enviarse en cada transferencia.
- Después de cada byte, el receptor (esclavo/maestro) puede hacer que el transmisor (maestro) espere forzando la línea SCL a nivel bajo.



2.8.4. Señales. Reconocimiento y no reconocimiento

- Tras los 8 bits del byte el maestro genera un noveno pulso de reloj
- El receptor genera un reconocimiento (ACK) del byte forzando SDA a nivel bajo durante el noveno pulso de SCL
- El receptor genera un "no reconocimiento" (NACK) del byte dejando SDA a nivel alto durante el noveno pulso de SCL



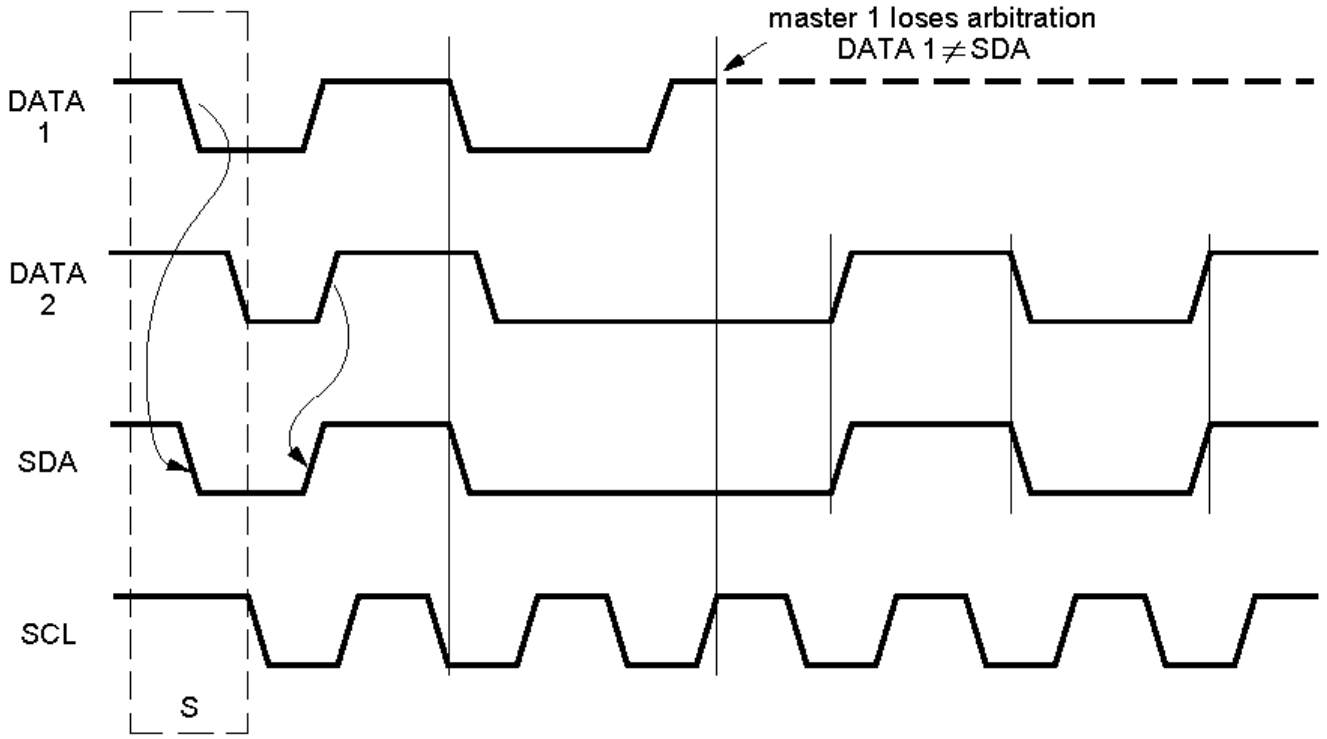
2.8.4. Señales. Posibles causas de no reconocimiento (NACK)

- En el bus no hay ningún esclavo con la dirección especificada por el maestro
- Un esclavo no puede recibir o transmitir debido a que está realizando una operación interna y no está preparado para comunicarse con el maestro
- Un receptor recibe datos que no puede entender
- Un receptor no puede aceptar en este momento más bytes de datos
- Un maestro receptor indica a un esclavo transmisor que debe dejar de enviar datos

2.8.5. Arbitraje (1)

- Permite que si dos maestros inician la transmisión al mismo tiempo uno de ellos gane el acceso al bus sin pérdida del mensaje que éste intentaba enviar
- Se basa en la función “AND cableada” en SDA
- Bit a bit, cada maestro examina la línea SDA comprobando que su estado coincide con el que él pretende imponer
- Si un maestro pretende poner SDA a nivel alto y la encuentra a nivel bajo, el maestro en cuestión pierde el arbitraje y desactiva su driver de SDA
- Una vez que el bus queda libre, un maestro que ha perdido el arbitraje puede reintentar la comunicación

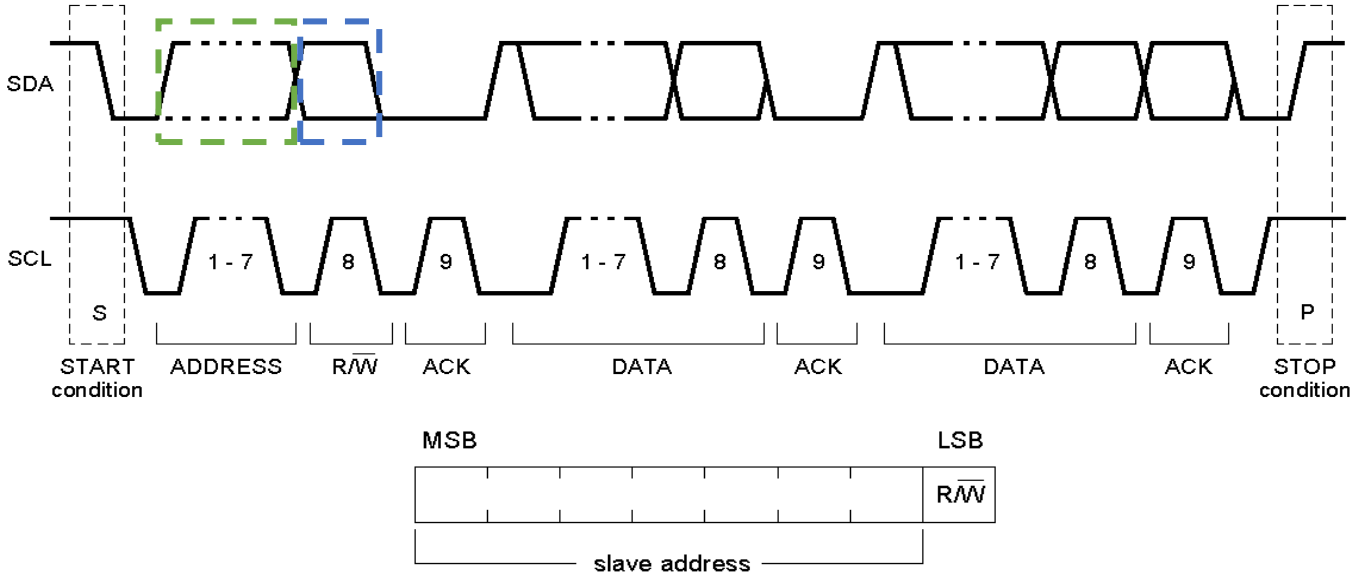
2.8.5. Arbitraje (2)



- Los maestros 1 y 2 compiten por el bus
- El maestro 1 pierde el arbitraje cuando pretende poner SDA a 1 mientras el maestro 2 la fuerza a 0

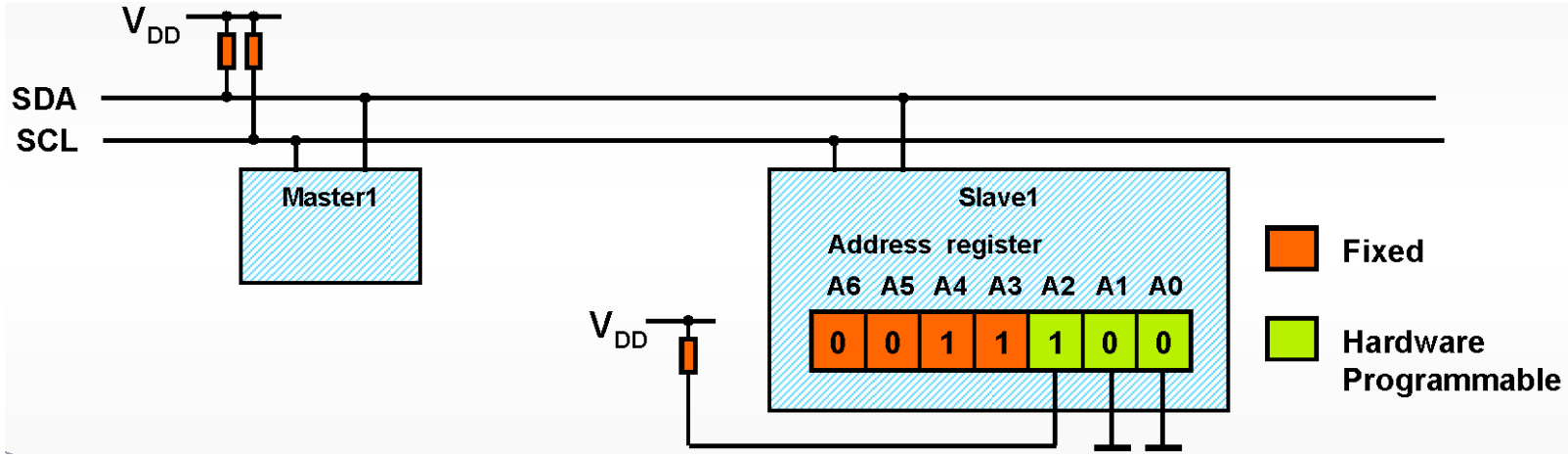
2.8.6. Transferencias. Dirección del esclavo y bit R/W

- Después de la condición de START el maestro envía la dirección del esclavo de 7 bits al que quiere acceder
- Tras la dirección del esclavo el maestro envía el bit R/W:
  - R/W = 0: el maestro quiere escribir en el esclavo (transmitir datos al esclavo)
  - R/W = 1: el maestro quiere leer el esclavo (recibir datos del esclavo)



2.8.6. Dirección parcialmente programable

- La mayoría de los dispositivos I2C tienen una parte de su dirección fija y el resto es programable mediante una serie de pines
- Permite la conexión de varios dispositivos idénticos al mismo bus. La parte programable se ajusta de forma diferente en cada uno para que posean direcciones I2C distintas



2.8.6. Direcciones reservadas

Las direcciones con el formato 0000xxx y 1111xxx están reservadas para propósitos especiales

Dirección	Bit R/W	Descripción
0000 000	0	Llamada general
0000 000	1	Byte START
0000 001	x	Dirección CBUS
0000 010	x	Reservada para un bus con distinto formato
0000 011	x	Reservada para uso futuro
0000 1xx	x	Maestro High-speed mode
1111 1xx	x	Reservadas para uso futuro
1111 0xx	x	Direccionamiento de 10 bits

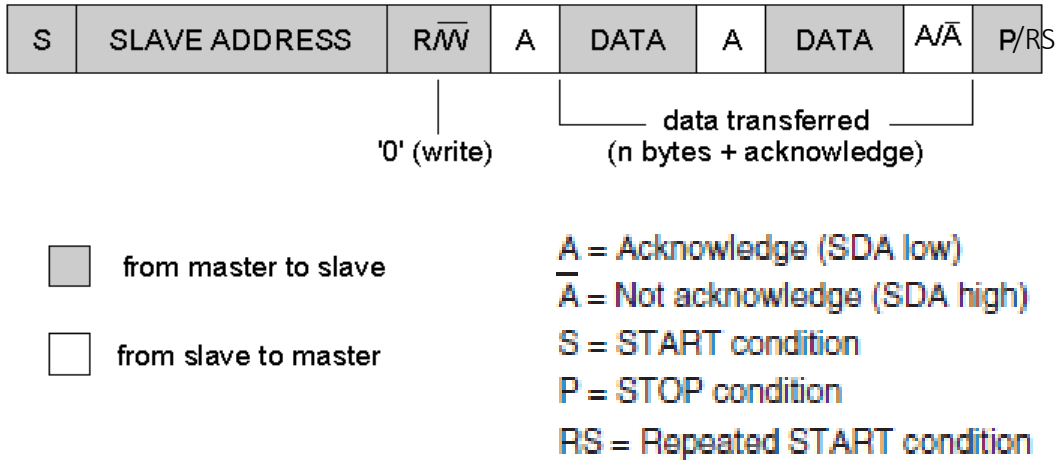
2.8.6. Direccionamiento de 10 bits

- Permite expandir el número de dispositivos conectados al bus
- Puede coexistir con el direccionamiento de 7 bits
- Emplea una primera dirección con la combinación especial 11110xx
- La dirección de 10 bits se forma concatenando los bits xx de la primera dirección y los 8 bits del segundo byte transmitido por el maestro

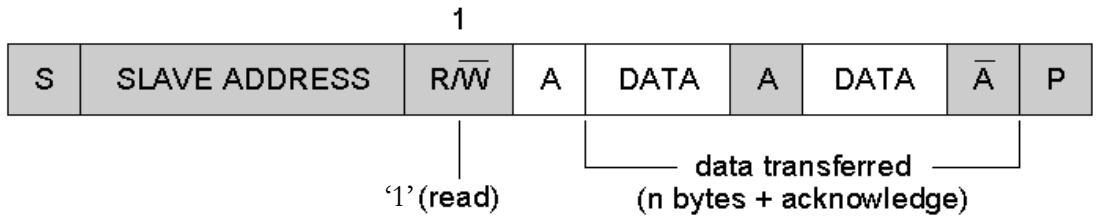
Ejemplo de maestro transmisor direccionando a un esclavo receptor con una dirección de 10 bits.

2.8.6. Ejemplos de transferencia (1)

- **Maestro transmisor** enviando una serie de bytes a un **esclavo receptor**



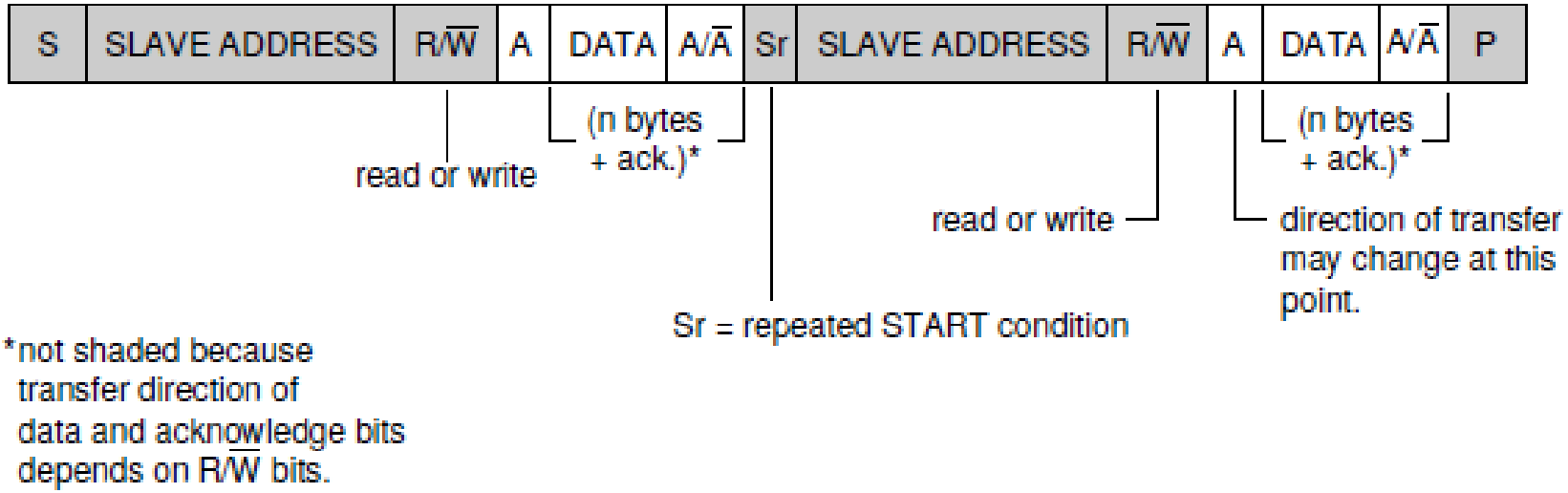
- **Maestro receptor** recibiendo una serie de bytes de un **esclavo transmisor**



2.8.6. Ejemplos de transferencia (2)

Transferencia combinada:

1. Un maestro direcciona a un esclavo y lo lee o escribe
2. A continuación, envía un REPEATED START
3. Vuelve a direccionar al mismo esclavo y lo lee o escribe. La dirección de la segunda transferencia puede ser contraria a la de la primera



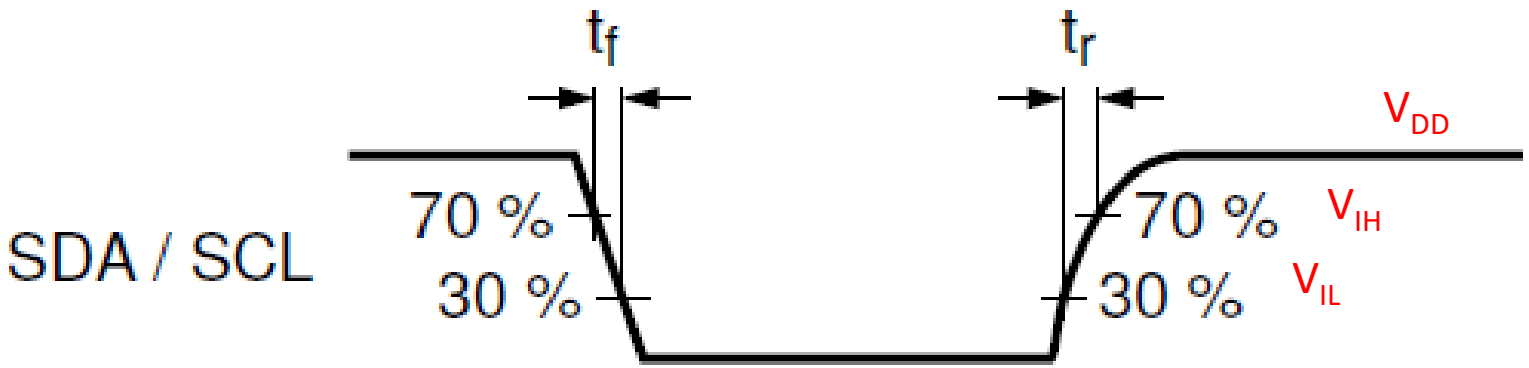
2.8.7. Cálculo de las resistencias de pull-up. Umbrales lógicos

La especificación I2C define como umbrales lógicos de entrada:

- Tensión máxima de entrada a nivel bajo: VIL(max) = 0,3 · VDD
- Tensión mínima de entrada a nivel alto: VIH(mín) = 0,7 · VDD

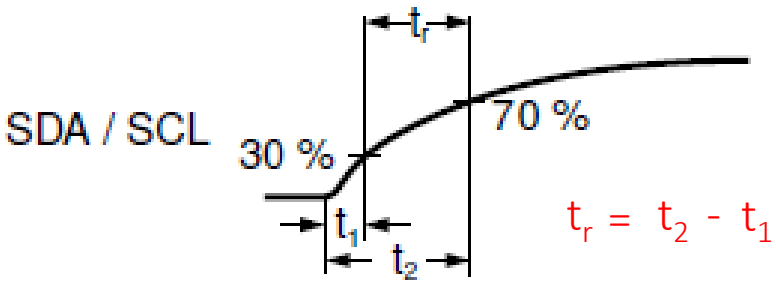
VDD es la tensión de alimentación

Los tiempos de bajada (tf) y subida (tr) se miden entre estos umbrales



2.8.7. Cálculo de las resistencias de pull-up

- Cuando una línea SDA/SCL cambia de bajo a alto, la capacidad de la línea Cb debe cargarse a través de Rp



$$V_{L\rightarrow H}(t) = V_{DD}(1 - e^{\frac{-t}{R_p C_b}})$$

- La capacidad de una línea SDA/SCL es la suma de:
  - La capacidad de cada pin conectado a la línea (≈ 10 pF)
  - La capacidad de la pista PCB
- La especificación I2C impone un tiempo de subida tr máximo para las señales SDA y SCL (1 µs en Standard-mode y 300ns en Fast-mode)
- La capacidad de las líneas del bus y el tr máximo determina la Rp máxima

2.8.7. Cálculo de las resistencias de pull-up

$R_p < R_p(max)$ para no superar el t_r máximo

$Tomando\ V_{IH} = 0{,}7V_{DD}$ y $\mathrm{V_{IL}} = 0{,}3\mathrm{V_{DD}}$

$$V(t_1) = 0{,}3V_{DD} = V_{DD}(1 - e - t_1/R_pC_b)$$

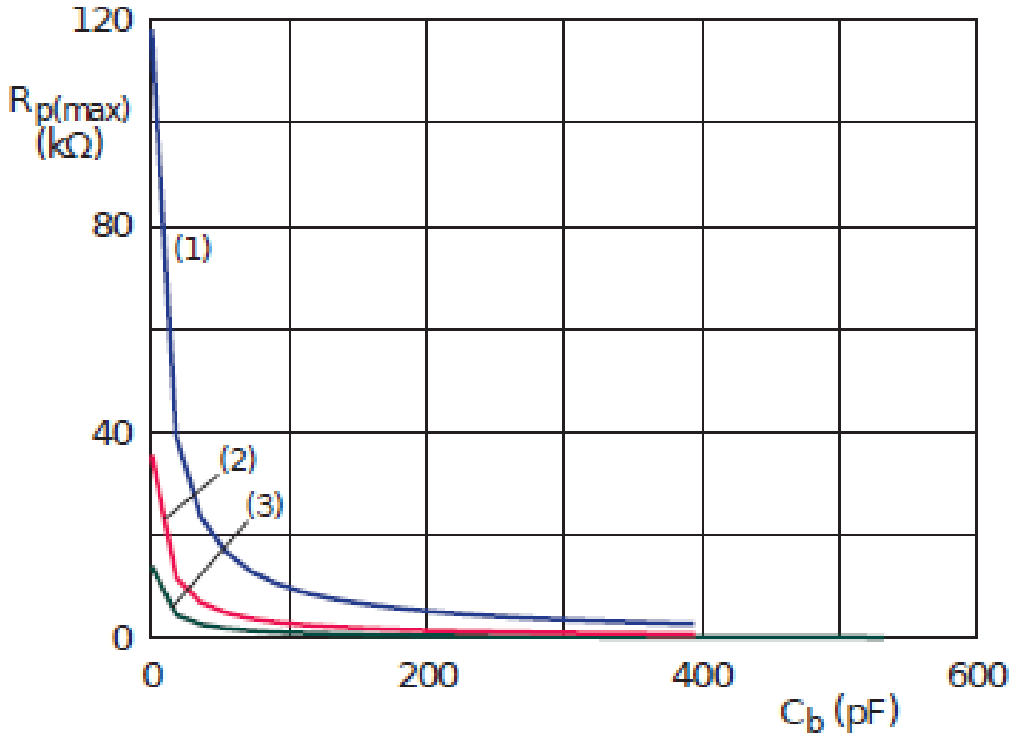
$$t_1 \approx 0{,}3566749\ R_pC_b$$

$$V(t_2) = 0{,}7V_{DD} = V_{DD}(1 - e - t_2/R_pC_b)$$

$$t_2 \approx 1{,}2039729\ R_pC_b$$

$$t_r = t_2 - t_1 \approx 0{,}847\ R_pC_b$$

$$\boxed{R_p(máx) = \frac{t_r(máx)}{0{,}847\ C_b}}$$



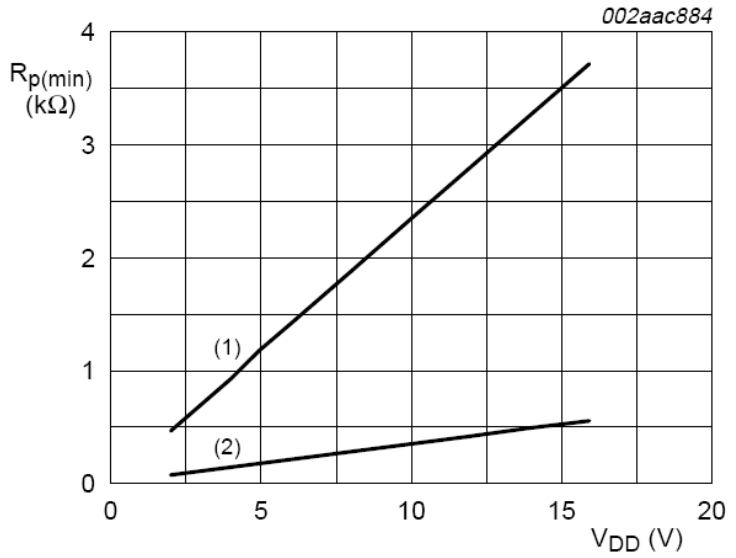
(1) Standard-mode
(2) Fast-mode
(3) Fast-mode Plus

2.8.7. Cálculo de las resistencias de pull-up

- La R_p mínima viene definida por la tensión de alimentación y por la máxima corriente de sink de 3 mA especificada para los modos Standard-mode y Fast-mode (1) y de 20 mA para el modo Fast-mode plus (2)

$$R_p(mín) = \frac{V_{DD} - V_{OL(mín)}}{I_{OL(máx)}}$$

$R_p > R_p(min)$



(1) Fast-mode and Standard-mode

(2) Fast-mode Plus

2.8.8. I2C en el LPC40xx

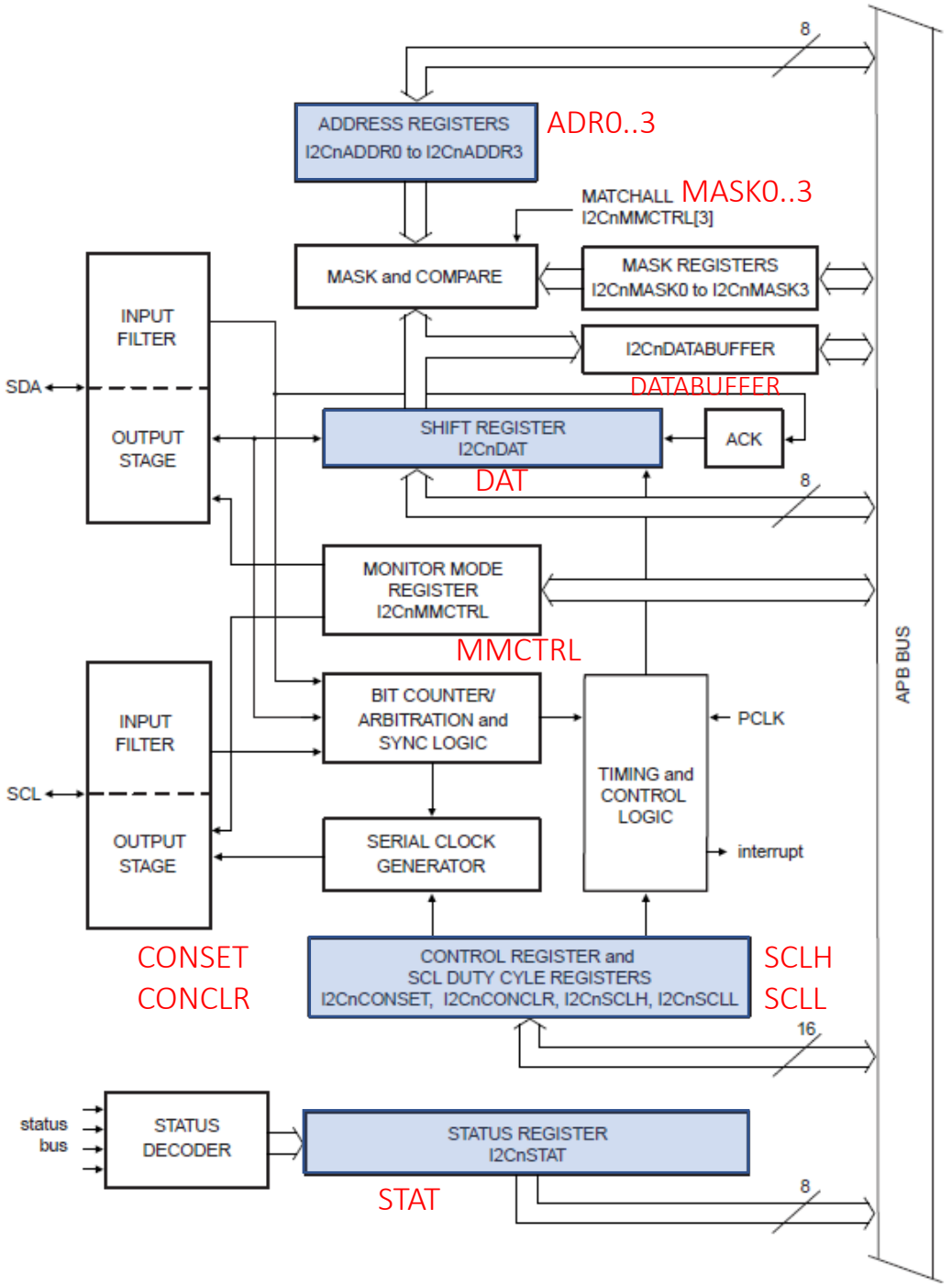
- Cumple con las especificaciones I2C y puede ser configurado como maestro/esclavo. Tres buses I2C0, I2C1 e I2C2
- Arbitraje entre maestros transmitiendo simultáneamente sin corrupción de los datos
- Reloj programable SCL para ajustarse a la relación de transferencias I2C
- La sincronización serie puede ser usada para comunicar dispositivos con diferente relación de bits y como mecanismo handshake para detener y continuar la transferencia serie
- Transferencia de datos bidireccional entre maestro y esclavo
- Puede ser usado para test y diagnósticos de dispositivos
- En el reset todos los I2C son habilitados
- Como esclavo, reconocimiento de hasta 4 direcciones
- Modo monitor para observar el bus sin intervenir en la comunicación

2.8.8. Selección de los pines del I2C

Register	000	001	010	011	100	101	110	111
IOCON_P0_0	P0[0]	CAN_RD1	U3_TXD	I2C1_SDA	U0_TXD			
IOCON_P0_1	P0[1]	CAN_TD1	U3_RXD	I2C1_SCL	U0_RXD			
IOCON_P0_10	P0[10]	U2_TXD	I2C2_SDA	T3_MAT0				LCD_VD[5]
IOCON_P0_11	P0[11]	U2_RXD	I2C2_SCL	T3_MAT1				LCD_VD[10]
IOCON_P0_27	P0[27]	I2C0_SDA	USB_SDA1					
IOCON_P0_28	P0[28]	I2C0_SCL	USB_SCL1					
IOCON_P1_15	P1[15]	ENET_RX_CLK		I2C2_SDA				
IOCON_P2_14	P2[14]	EMC_CS2	I2C1_SDA	T2_CAP0				
IOCON_P2_15	P2[15]	EMC_CS3	I2C1_SCL	T2_CAP1				
IOCON_P5_2	P5[2]		SSP2_SCK	T3_MAT2		I2C0_SDA		
IOCON_P5_3	P5[3]		SSP2_SSEL		U4_RXD	I2C0_SCL		

Tipo I Tipo D

2.8.9. Arquitectura del bus I2C



2.8.10. Registros (1)

Nombre	Descripción	Tipo	Reset
CONSET	**I2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I2C control register is set. Writing a zero has no effect on the corresponding bit in the I2C control register.	R/W	0x00
STAT	**I2C Status Register.** During I2C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8
DAT	**I2C Data Register.** During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register.	R/W	0x00
ADR0	**I2C Slave Address Register 0.** Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00
SCLH	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I2C clock.	R/W	0x04
SCLL	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I2C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I2C master and certain times used in slave mode.	R/W	0x04
CONCLR	**I2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I2C control register is cleared. Writing a zero has no effect on the corresponding bit in the I2C control register.	WO	NA
MMCTRL	**Monitor mode control register.**	R/W	0x00

2.8.10. Registros (2)

Nombre	Descripción	Tipo	Reset
ADR1	**I2C Slave Address Register 1.** Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00
ADR2	**I2C Slave Address Register 2.** Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00
ADR3	**I2C Slave Address Register 3.** Contains the 7-bit slave address for operation of the I2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00
DATA_ BUFFER	**Data buffer register.** The contents of the 8 MSBs of the I2DAT shift register will be transferred to the I2DATA_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus.	RO	0x00
MASK0	**I2C Slave address mask register 0**. This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address (‘0000000’).	R/W	0x00
MASK1	**I2C Slave address mask register 1.** Idem Register 0	R/W	0x00
MASK2	**I2C Slave address mask register 2.** Idem Register 0	R/W	0x00
MASK3	**I2C Slave address mask register 3.** Idem Register 0	R/W	0x00

2.8.10. Registro CONSET

- Permite poner a 1 bits de flag y control del interfaz
- Al escribir:
  - Los unos ponen a 1 los correspondientes bits
  - Los ceros no tienen efecto
- Al leer se retorna el estado actual de los bits

Bit	Symbol	Descripción	Reset
1:0	-	Reserved. Read value is undefined, only zero should be written	NA
2	AA	Assert acknowledge flag.	0
3	SI	I2C interrupt flag.	0
4	STO	STOP flag.	0
5	STA	START flag.	0
6	I2EN	I2C interface enable.	0
31:7	-	Reserved. Read value is undefined, only zero should be written	NA

2.8.10. Registro CONCLR

- Permite poner a 0 bits de flag y control del interfaz
- Al escribir:
  - Los unos ponen a 0 los correspondientes bits
  - Los ceros no tienen efecto
- Al leer se retorna el estado actual de los bits

Bit	Symbol	Descripción	Reset
1:0	-	Reserved. Read value is undefined, only zero should be written	NA
2	AAC	Assert acknowledge Clear bit.	0
3	SIC	I2C interrupt Clear bit.	0
4	-	Reserved. Read value is undefined, only zero should be written.	NA
5	STAC	START flag Clear bit.	0
6	I2ENC	I2C interface Disable bit.	0
31:7	-	Reserved. Read value is undefined, only zero should be written	NA

2.8.10. Registro DAT

- Registro de datos
  - Se escribe/lee para enviar/recibir datos a través del interfaz
  - Sólo debe escribirse/leerse cuando el flag SI está a 1

Bit	Symbol	Descripción	Reset
7:0	Data	This register holds values that have been received or are to be transmitted.	0
31:8	-	Reserved. Read value is undefined, only zero should be written	NA

2.8.10. Registros SCLH y SCLL

- Permiten seleccionar la frecuencia y ciclo de trabajo de SCL cuando el µC funciona como maestro

Bit	Symbol	Descripción	Reset
15:0	SCLH	Count for SCL HIGH time period selection.	0x0004
31:16	-	Reserved. Read value is undefined, only zero should be written	NA

Bit	Symbol	Descripción	Reset
15:0	SCLL	Count for SCL LOW time period selection.	0x0004
31:16	-	Reserved. Read value is undefined, only zero should be written	NA

2.8.10. Calculo frecuencia SCL de I2C

$$f_{SCL} = I^2C_{rate} = \frac{PCLK}{SCLH+SCLL} \qquad SCLH + SCLL \geq 8$$

I2C Rate	I2CSCLL + I2CSCLH (kHz) y PCLK (MHz)													
	6	8	10	12	16	20	30	40	50	60	70	80	90	100
100 kHz (Standard)	60	80	100	120	160	200	300	400	500	600	700	800	900	1000
400 kHz (Fast Mode)	15	20	25	30	40	50	75	100	125	**150**	175	200	225	250
1 MHz (Fast Mode Plus)	-	8	10	12	16	20	30	40	50	60	70	80	90	100

fSCL = PeripheralClock/(SCLL + SCLH) => SCLL + SCLH = PeripheralClock/fSCL

Si PeripheralClock/fSCL es par: SCLL = SCLH = PeripheralClock/fSCL/2

Si PeripheralClock/fSCL es impar: SCLH = PeripheralClock/fSCL/2 y SCLL = SCLH + 1

2.8.10. Registro STAT

Permite conocer el estado actual del interfaz

Bit	Symbol	Descripción	Reset
2:0	-	These bits are unused and are always 0.	0
7:3	Status	These bits give the actual status information about the I^2C interface.	0x1F
31:8	-	Reserved. Read value is undefined, only zero should be written	NA

2.8.10. Estados del interfaz

Modo maestro transmisor, maestro receptor, esclavo receptor, esclavo transmisor

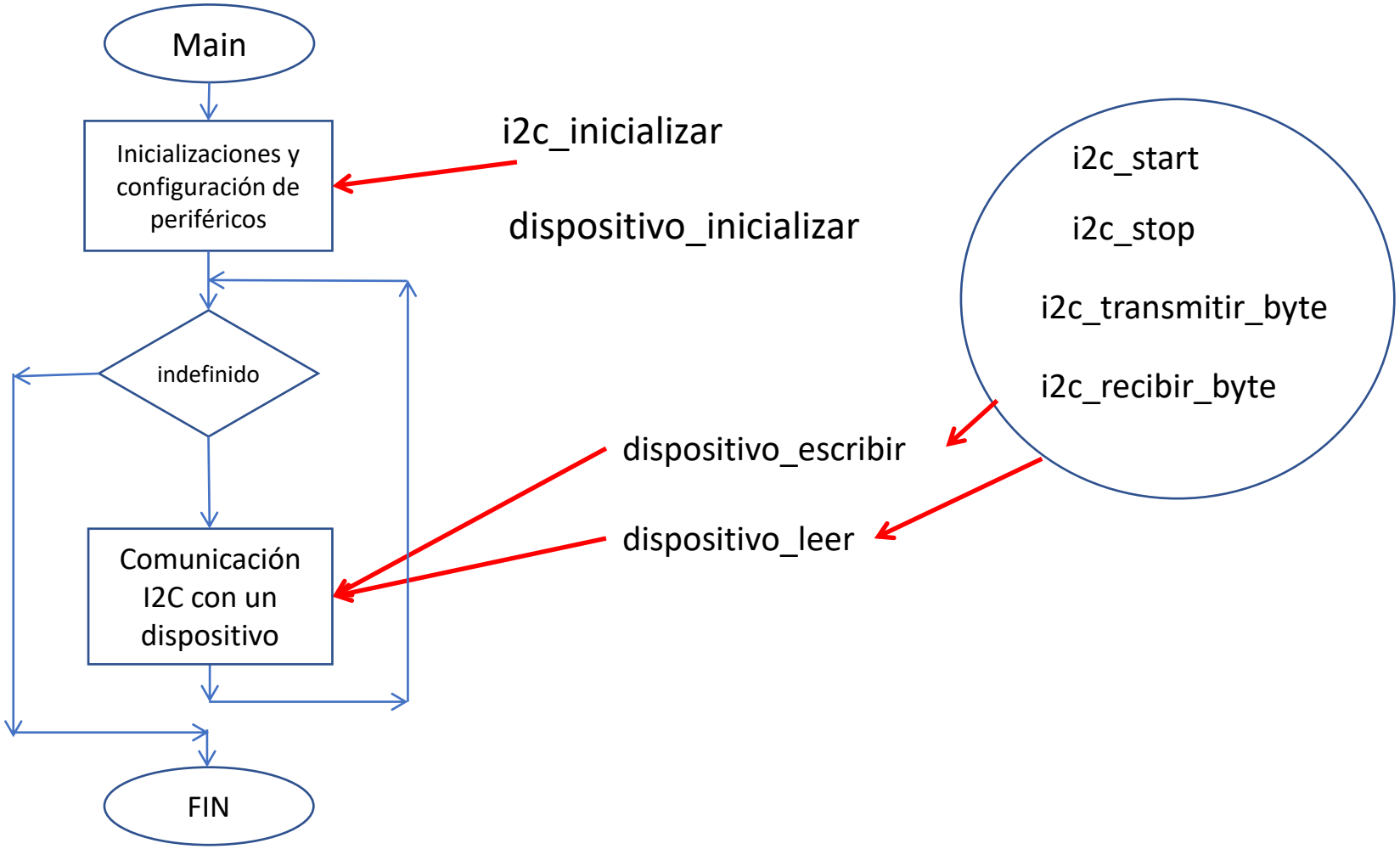
Table 512. Master Transmitter mode

I2CSTAT Status Code	Status of the I2C-bus and hardware	Application software response					Next action taken by I2C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0x08	A START condition has been transmitted.	Load SLA+W; clear STA	X	0	0	X	SLA+W will be transmitted; ACK bit will be received.
0x10	A repeated STARTcondition has been transmitted.	Load SLA+W or	X	0	0	X	As above.
		Load SLA+R; Clear STA	X	0	0	X	SLA+W will be transmitted; the I^2C block will be switched to MST/REC mode.
0x18	SLA+W has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x28	Data byte in I2DAT has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.

2.8.11. Pasos para la programación

- Inicializar el I2C:
  - Alimentar el I2C, bits 7/19/26 de PCONP a 1 para I2C0/1/2
  - Pasar como parámetro de la función la frecuencia SCL deseada
  - Definir los pines de salida P0.27 como SDA0 y P0.28 como SCL0 (IOCON)
  - Borrar los flags, AAC, SIC, STAC y I2ENC
  - Configurar velocidad en SCLL y SCLH
  - Habilitar el I2C0 en CONSET, poniendo a 1 el bit I2EN
- Disponer de funciones:
  - De Start
  - De Restart
  - De Stop
- Disponer de funciones:
  - De transmisión de bytes
  - De recepción de bytes

2.8.11. Funciones I2C sin interrupción



2.8.12. Fichero cabecera I2C (1)

```
/**********************************************************
 * \file   i2c_lpc40xx.h
 * \brief  Funciones de manejo de los interfaces I2C del LPC40x.
 */
#ifndef I2C_LPC40XX_H
#define I2C_LPC40XX_H

#include <LPC407x_8x_177x_8x.h>
#include "tipos.h"

/*===== Constantes ========================================*/
#define I2C0            LPC_I2C0
#define I2C1            LPC_I2C1
#define I2C2            LPC_I2C2
/* Bits de los registros CONSET y CONCLR. */
#define I2C_CON_AA      (1u << 2)
#define I2C_CON_SI      (1u << 3)
#define I2C_CON_STO     (1u << 4)
#define I2C_CON_STA     (1u << 5)
#define I2C_CON_I2EN    (1u << 6)
```

2.8.12. Fichero cabecera I2C (2)

```
/*===== Prototipos de funciones solo como Maestro ==========*/
void i2c_inicializar(LPC_I2C_TypeDef *i2c_regs,
                     uint32_t frecuencia_scl,
                     LPC_GPIO_TypeDef *puerto_sda,
                     uint32_t mascara_pin_sda,
                     LPC_GPIO_TypeDef *puerto_scl,
                     uint32_t mascara_pin_scl);

void i2c_start(LPC_I2C_TypeDef *i2c_regs);
void i2c_stop(LPC_I2C_TypeDef *i2c_regs);

bool_t i2c_transmitir_byte(LPC_I2C_TypeDef *i2c_regs,
                           uint8_t byte);
uint8_t i2c_recibir_byte(LPC_I2C_TypeDef *i2c_regs,
                         bool_t ack);
#endif /* I2C_LPC40XX_H */
```

2.8.12. Función inicializar I2C (1)

```
/*********************************************************************
 * \brief       Inicializar un interfaz I2C del LPC40xx.
 * \param[in]   i2c_regs       Puntero a regs. del interfaz a inicializar.
 * \param[in]   frecuencia_scl Frecuencia de la señal SCL.
 * \param[in]   puerto_sda     Puerto que se desea para la función SDA.
 * \param[in]   pin_sda        Pin que se desea para la función SDA.
 * \param[in]   puerto_scl     Puerto que se desea para la función SCL.
 * \param[in]   pin_scl        Pin que se desea para la función SCL.
 */

void i2c_inicializar(LPC_I2C_TypeDef *i2c_regs,
                     uint32_t frecuencia_scl,
                     LPC_GPIO_TypeDef *puerto_sda,
                     uint32_t mascara_pin_sda,
                     LPC_GPIO_TypeDef *puerto_scl,
                     uint32_t mascara_pin_scl){

  uint32_t sclh_mas_scll;

  ASSERT(i2c_regs == I2C0 || i2c_regs == I2C1 || i2c_regs == I2C2,
           "Número de interfaz I2C incorrecto.");

  sclh_mas_scll = PeripheralClock/frecuencia_scl;

  ASSERT(sclh_mas_scll >= 8,
         "No se puede ajustar la frecuencia I2C solicitada.");

/* Los números de puerto y pin se comprueban en las funciones de
 * configuración de pines.*/
```

2.8.12. Función inicializar I2C (2)

```
/* Aplicar alimentación al interfaz elegido y configurar los pines SDA
 * y SCL indicados.
 */

    if (i2c_regs == I2C0){

      LPC_SC->PCONP |= 1u << 7;

      iocon_configurar_pin(puerto_sda, mascara_pin_sda, I2C0_SDA,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

      iocon_configurar_pin(puerto_scl, mascara_pin_scl, I2C0_SCL,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

    }else if (i2c_regs == I2C1){

      LPC_SC->PCONP |= 1u << 19;

      iocon_configurar_pin(puerto_sda, mascara_pin_sda, I2C1_SDA,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

      iocon_configurar_pin(puerto_scl, mascara_pin_scl, I2C1_SCL,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

    }else{

      LPC_SC->PCONP |= 1u << 26;

      iocon_configurar_pin(puerto_sda, mascara_pin_sda, I2C2_SDA,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

      iocon_configurar_pin(puerto_scl, mascara_pin_scl, I2C2_SCL,
            IOCON_NO_PULL_UP_NO_PULL_DOWN | IOCON_FILTER | IOCON_OD);

    }
```

2.8.12. Función inicializar I2C (2)

```
    /* Mientras se configura, borrar los flags del registro de control y
     * deshabilitar el interfaz.*/

    i2c_regs->CONCLR = I2C_CON_I2EN | I2C_CON_STA | I2C_CON_SI | I2C_CON_AA;

    /* Establecer la frecuencia SCL.*/

    if (sclh_mas_scll % 2 == 0){

      /* Si sclh_mas_scll es par, SCLH = sclh_mas_scll / 2
       * y SCLL = sclh_mas_scll / 2.
       */

      i2c_regs->SCLH = sclh_mas_scll / 2;

    }else{

      /* Si sclh_mas_scll es impar, SCLH = sclh_mas_scll / 2 + 1
       * y SCLL = sclh_mas_scll / 2.
       */

      i2c_regs->SCLH = sclh_mas_scll / 2 + 1;

    }

    i2c_regs->SCLL = sclh_mas_scll / 2;


    /* Habilitar el interfaz.
     */

    i2c_regs->CONSET = I2C_CON_I2EN;

}
```

2.8.12. Función inicio transmisión i2c

```
/*****************************************************************
 * \brief   Crear condición de start.
 *
 * \param[in] i2c_regs Puntero al bloque de registros del interfaz.
 */
void i2c_start(LPC_I2C_TypeDef *i2c_regs){

  ASSERT(i2c_regs == I2C0 || i2c_regs == I2C1 || i2c_regs == I2C2,
         "Número de interfaz I2C incorrecto.");

  /* El estado inicial de los bits STA y SI debe ser 0.*/

  i2c_regs->CONCLR = I2C_CON_STA | I2C_CON_SI;


  /* Crear condición START activando el bit STA del registro CON.*/

  i2c_regs->CONSET = I2C_CON_STA;

}
```

2.8.12. Función parada de transmisión I2C

```
/*********************************************************
 * \brief   Crear condición de stop.
 * \param[in]   i2c_regs    Puntero al bloque de registros del interfaz.
 */

void i2c_stop(LPC_I2C_TypeDef *i2c_regs){

  ASSERT(i2c_regs == I2C0 || i2c_regs == I2C1 || i2c_regs == I2C2,
        "Número de interfaz I2C incorrecto.");

  /* Esperar a que el bit SI del registro CON esté a 1. Esto indica que ha
   * terminado la operación anterior sobre el interfaz.*/

  while ((i2c_regs->CONSET & I2C_CON_SI) == 0) {}

  /* Activar el bit STO del registro CON. */

  i2c_regs->CONSET = I2C_CON_STO;

  /* Borrar el bit STA del registro CON para estar seguros de no crear un
   * START antes del STOP que queremos (STA podría estar a 1 ahora).
   * Borrar el bit SI del registro CON. Con esto la máquina de estados del
   * interfaz avanza y crea la condición de STOP.*/

  i2c_regs->CONCLR = I2C_CON_STA | I2C_CON_SI;

}
```

2.8.12. Función transmitir_byte I2C (1)

```
/*************************************************************
 * \brief     Transmitir un byte a un esclavo a través del I2C 0.
 * \param[in] i2c_regs    Puntero a registros del interfaz.
 * \param[in] byte        Byte a transmitir.
 * \return    TRUE => el esclavo reconoció el dato,
 *            FALSE => no lo reconoció.
 */

bool_t i2c_transmitir_byte(LPC_I2C_TypeDef *i2c_regs, uint8_t byte){

  uint8_t status;

  ASSERT(i2c_regs == I2C0 || i2c_regs == I2C1 || i2c_regs == I2C2,
         "Número de interfaz I2C incorrecto.");


  /* Esperar a que el bit SI del registro CON esté a 1. Esto indica que ha
   * terminado la operación anterior sobre el interfaz.
   */

  while ((i2c_regs->CONSET & I2C_CON_SI) == 0) {}

  /* Escribir el byte a enviar en el registro de datos DAT.
   */

  i2c_regs->DAT = byte;
```

2.8.12. Función transmitir_byte I2C (2)

```
 /* Borrar el bit STA del registro CON para estar seguros de no crear un
   * START (STA podría estar a 1 ahora).
   * Borrar el bit SI del registro CON. Con esto la máquina de estados del
   * interfaz avanza y envía el dato en el registro DAT.
   */

  i2c_regs->CONCLR = I2C_CON_STA | I2C_CON_SI;

  /* Esperar a que el bit SI del registro CON esté a 1. Esto indica que ha
   * terminado la transmisión del byte.
   */

  while ((i2c_regs->CONSET & I2C_CON_SI) == 0) {}


  /* Leer el registro de estado para determinar si el esclavo reconoció o no
   * el byte. Retornar TRUE o FALSE según el caso.
   */

  status = i2c_regs->STAT;

  if ((status == 0x18) || (status == 0x28)) return TRUE;

  else return FALSE;

}
```

2.8.12. Función recibir_byte I2C (1)

```
/**********************************************************************
 * \brief      Recibir un byte desde un esclavo a través del I2C 0.
 * \param[in]  i2c_regs Puntero a registros del interfaz.
 * \param[in]  ack      TRUE => generar un ACK,
 *                      FALSE => generar un NACK.
 * \return     Byte recibido del esclavo.
 */


uint8_t i2c_recibir_byte(LPC_I2C_TypeDef *i2c_regs, bool_t ack){

  ASSERT(i2c_regs == I2C0 || i2c_regs == I2C1 || i2c_regs == I2C2,
         "Número de interfaz I2C incorrecto.");


  /* Esperar a que el bit SI del registro CON esté a 1. Esto indica
   * que ha terminado la operación anterior sobre el interfaz.
   */

  while ((i2c_regs->CONSET & I2C_CON_SI) == 0) {}
```

2.8.12. Función recibir_byte I2C (2)

```
  /* Si queremos reconocer el dato, activar el bit AA del registro
   * CON. Si no queremos reconocer el dato, borrar el bit AA.
   *
   * En ambos casos se borra el bit STA del registro CON para estar
   * seguros de no crear una condición de START (STA podría estar a
   * 1 ahora).
   * También se borra el bit SI del registro CON. Con esto la
   * máquina de estados del interfaz avanza y comienza la recepción
   * del byte.
   */

  if (ack){

    i2c_regs->CONSET = I2C_CON_AA;
    i2c_regs->CONCLR = I2C_CON_STA | I2C_CON_SI;

  }else{

    i2c_regs->CONCLR = I2C_CON_STA | I2C_CON_SI | I2C_CON_AA;

  }

  /* Esperar hasta que el bit SI del registro CON a 1. Esto indica
   * que ha terminado la recepción del byte.
   */

  while ((i2c_regs->CONSET & I2C_CON_SI) == 0) {}

  /* Retornar el byte recibido.
   */

  return i2c_regs->DAT;

}
```