Direcciones de 10 bits

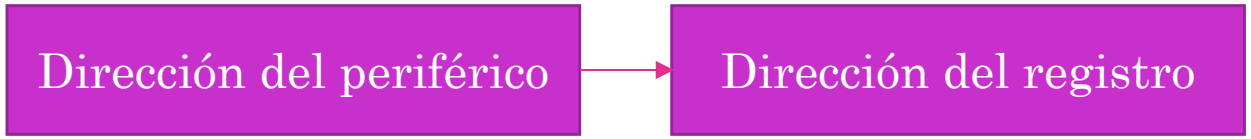
Se necesitan dos frames para transmitir la dirección

1er frame: Contiene **11110xyz**, donde *x* es el noveno bit de la dirección, *y* es el octavo y *z* es bit de lectura/escritura.
El ACK es colocado por todos los periféricos cuyos dos bits de dirección coinciden con lo transmitido.

2do frame: El resto de los ocho bits de dirección.

Periféricos con direcciones de 10 bits pueden coexistir con los de 7 bits, ya que los bits 11110 no son parte de ninguna dirección de 7 bits válida.

Byte de comando



(Frame de datos)

Escritura del periférico

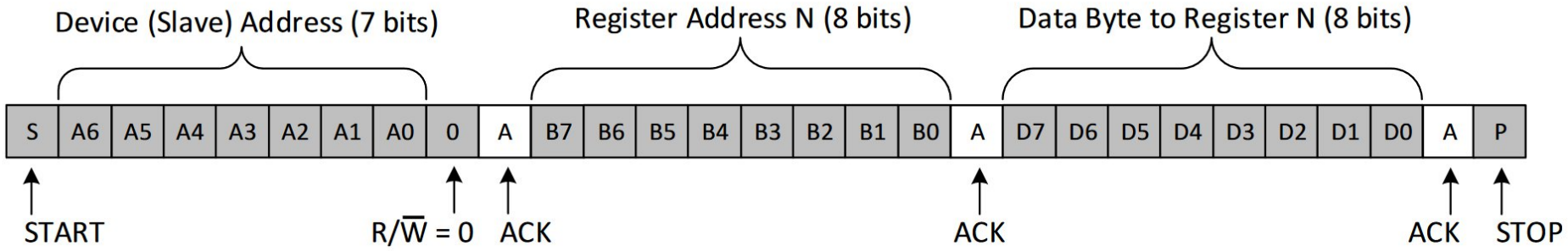
1. El controlador envía la condición de inicio, la dirección del periférico y el bit R/~W en bajo para una escritura.
2. El periférico envía un ACK.
3. El controlador envía la dirección del registro.
4. El periférico envía un ACK.
5. El controlador envía los datos a escribir en el registro (uno o más bytes, cada byte recibe un ACK).
6. El controlador termina la transmisión por medio de la condición de fin.

Escritura del periférico

Master Controls SDA Line

Slave Controls SDA Line

Write to One Register in a Device



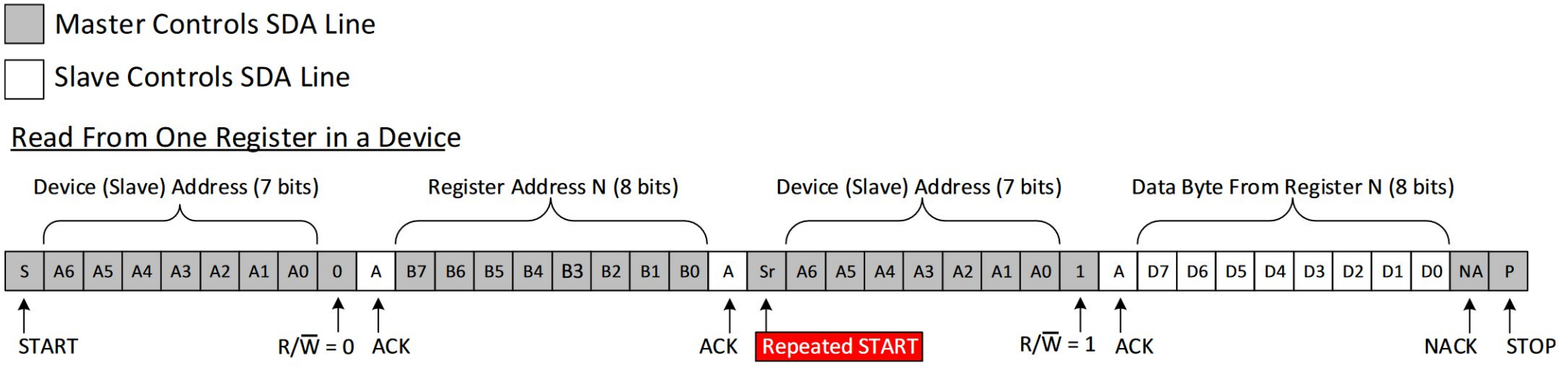
Lectura del periférico

Es similar a la escritura pero con pasos adicionales.

1. El controlador envía la condición de inicio, la dirección del periférico y el bit R/~W en bajo para una escritura.
2. El periférico envía un ACK.
3. El controlador envía la dirección del registro.
4. El periférico envía un ACK.
5. El controlador envía una condición de inicio repetida.
6. El controlador envía la dirección del periférico con el bit R/~W en alto para una lectura.
7. El periférico envía el ACK.

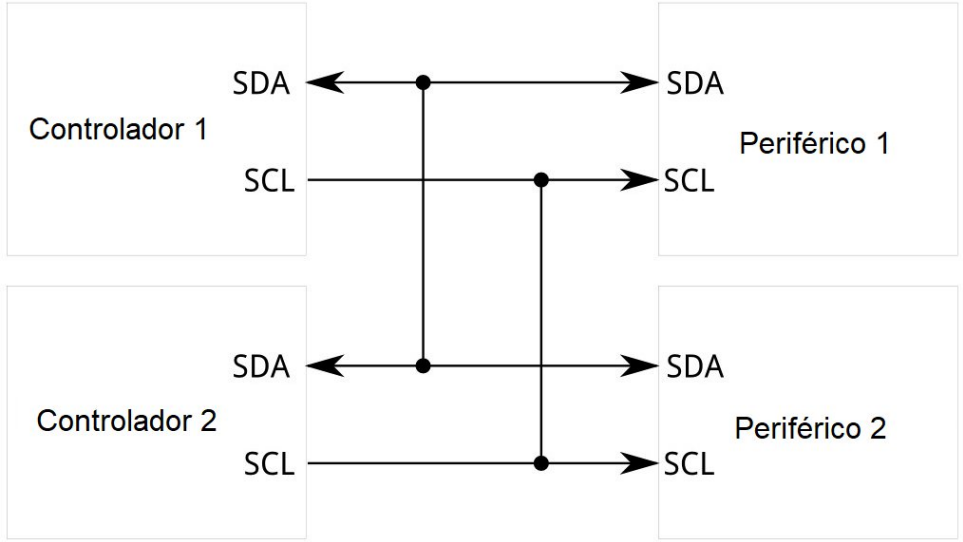
8. El controlador libera la línea SDA pero continua generando pulsos de reloj en SCL.
9. El periférico envía los bytes de datos. Por cada byte, el controlador envía un ACK.
10. Cuando el controlador ha recibido la cantidad de bytes que esperaba, envía un NACK, indicando al periférico que termine la transmisión y libere el bus.
11. El controlador envía una condición de fin.

Lectura del periférico



Condiciones de inicio repetidas

- En algunas ocasiones es necesario que un controlador pueda intercambiar varios mensajes en un mismo ciclo de comunicación, sin dejar que otros controladores interfieran en el bus.
- En estos casos, el controlador puede usar una condición de inicio repetida.



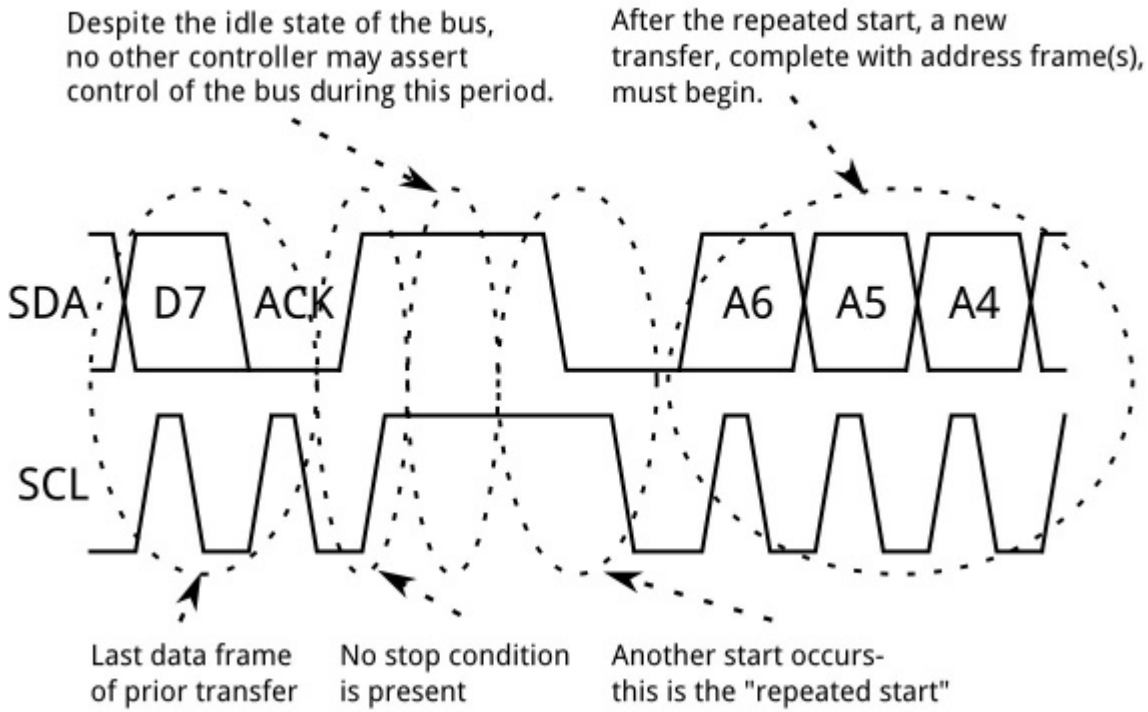
Condiciones de inicio repetidas

Después del último ACK:
1. SDA pasa a alto cuando SCL está en bajo.
2. SDA pasa a bajo mientras SCL está en alto

Ya que no hubo una condición de fin, la comunicación no estuvo realmente completada, así que el controlador puede seguir usando el bus

El controlador puede realizar cualquier número de condiciones de inicio repetidas, el controlador va a mantener control de bus hasta que realice una condición de fin

Condiciones de inicio repetidas



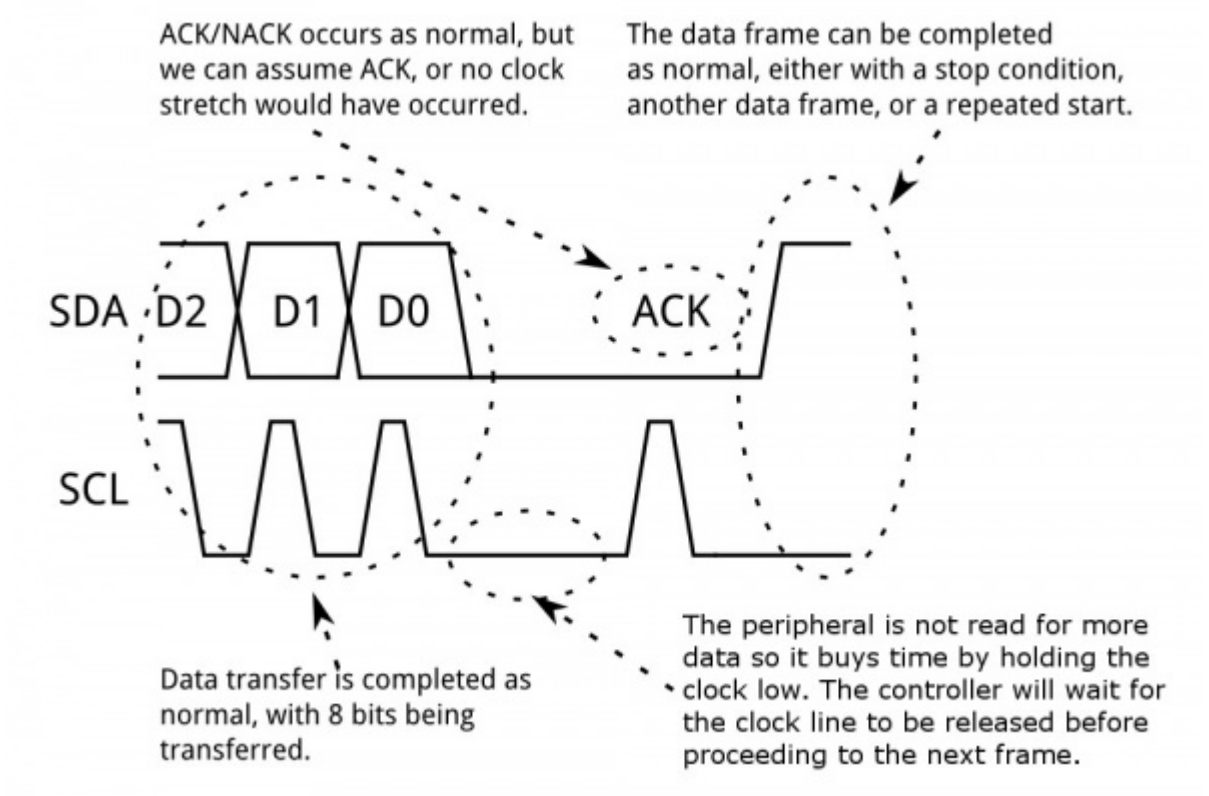
Estiramiento del reloj

En ocasiones la velocidad de transmisión del controlador supera la capacidad del periférico para proveer datos (por ejemplo, cuando conversiones ADC o escrituras a memoria no volátil no han sido completadas en el momento que el controlador pide una lectura de ellas).

En estos casos, algunos periféricos ejecutan un estiramiento del reloj. Normalmente, la señal de reloj es generada por el controlador. Sin embargo, en cualquier punto de la transferencia de datos, el periférico direccionado puede mantener en bajo a SCL después de que el controlador la libera.

El controlador deja de generar pulsos de reloj y transmitir datos hasta que el periférico libera SCL.

Estiramiento del reloj



MASTER

```
#include <stdio.h>
#include "esp_log.h"
#include "driver/i2c.h"

static const char *TAG = "master";

#define I2C_MASTER_SCL_IO           22
#define I2C_MASTER_SDA_IO           21
#define I2C_MASTER_NUM              0
#define I2C_MASTER_FREQ_HZ          400000
#define I2C_MASTER_TX_BUF_DISABLE   0
#define I2C_MASTER_RX_BUF_DISABLE   0
#define I2C_MASTER_TIMEOUT_MS       1000
#define I2C_SLAVE_ADDR              0x04
```

```c
void app_main(void)
{
    uint8_t data;
    esp_err_t ret;
    ESP_ERROR_CHECK(i2c_master_init());
    ESP_LOGI(TAG, "I2C initialized successfully");

    while (1)
    {
        if ((ret = device_read(&data, 1)) == ESP_OK)
        {
            ESP_LOGI(TAG, "Data read = %X", data);
        }
        else
        {
            ESP_LOGI(TAG, "Read error = %X", ret);
        }
        vTaskDelay(100 / portTICK_PERIOD_MS);
    }


    ESP_ERROR_CHECK(i2c_driver_delete(I2C_MASTER_NUM));
    ESP_LOGI(TAG, "I2C unitialized successfully");
}
```

```c
static esp_err_t i2c_master_init(void)
{
    int i2c_master_port = I2C_MASTER_NUM;

    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };

    i2c_param_config(i2c_master_port, &conf);

    return i2c_driver_install(i2c_master_port, conf.mode,
        I2C_MASTER_RX_BUF_DISABLE, I2C_MASTER_TX_BUF_DISABLE, 0);
}
```

```c
static esp_err_t device_read(uint8_t *data, size_t len)
{
    return i2c_master_read_from_device(I2C_MASTER_NUM, I2C_SLAVE_ADDR, data, len,
        I2C_MASTER_TIMEOUT_MS / portTICK_PERIOD_MS);
}
```

```
esp_err_t i2c_master_read_from_device(
i2c_port_t i2c_num, /* Puerto I2C (I2C_NUM_0 o I2C_NUM_1) */
uint8_t device_address,  /* Dirección del dispositivo de 7 bits */
uint8_t *read_buffer,  /* Buffer para almacenar los bytes recibidos */
size_t read_size,  /* Tamaño, en bytes, del buffer de lectura */
TickType_t ticks_to_wait /* Tiempo máximo en ticks que la función esperará */
)
```

SLAVE

```
#include <stdio.h>
#include "esp_log.h"
#include "driver/i2c.h"

static const char *TAG = "slave";

#define I2C_SLAVE_SCL_IO          22
#define I2C_SLAVE_SDA_IO          21
#define I2C_SLAVE_NUM             0
#define DATA_LENGTH               512
#define RW_TEST_LENGTH            128
#define I2C_SLAVE_TX_BUF_LEN      (2 * DATA_LENGTH)
#define I2C_SLAVE_RX_BUF_LEN      (2 * DATA_LENGTH)
#define I2C_SLAVE_ADDR            0x04
```

```
static esp_err_t i2c_slave_init(void)
{
    int i2c_slave_port = I2C_SLAVE_NUM;

    i2c_config_t conf_slave = {
        .sda_io_num = I2C_SLAVE_SDA_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_io_num = I2C_SLAVE_SCL_IO,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .mode = I2C_MODE_SLAVE,
        .slave.addr_10bit_en = 0,
        .slave.slave_addr = I2C_SLAVE_ADDR,
        .clk_flags = 0,
    };
    i2c_param_config(i2c_slave_port, &conf_slave);

    return i2c_driver_install(i2c_slave_port, conf_slave.mode,
        I2C_SLAVE_RX_BUF_LEN, I2C_SLAVE_TX_BUF_LEN, 0);
}
```

```c
void app_main(void)
{
    uint8_t data[DATA_LENGTH];
    uint8_t * ptr;
    uint16_t pos;

    ESP_ERROR_CHECK(i2c_slave_init());
    ESP_LOGI(TAG, "I2C initialized successfully");

    for (uint16_t i = 0; i < DATA_LENGTH; i++)
    {
        data[i] = i;
    }

    pos = 0;
    ptr = &data[pos];

    while (1)
    {
        uint16_t d_size = i2c_slave_write_buffer(I2C_SLAVE_NUM, ptr,
            RW_TEST_LENGTH, 1000 / portTICK_PERIOD_MS);
        if (d_size == 0)
        {
            ESP_LOGI(TAG, "i2c slave tx buffer full");
        }
        else
        {
            pos = (pos + d_size) % DATA_LENGTH;
        }
        ptr = &data[pos];

        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }

    ESP_ERROR_CHECK(i2c_driver_delete(I2C_SLAVE_NUM));
    ESP_LOGI(TAG, "I2C unitialized successfully");
}
```

int **i2c_slave_write_buffer**(
i2c_port_t **i2c_num**, /* Puerto I2C (I2C_NUM_0 o I2C_NUM_1) */
const uint8_t ***data**, /* Bytes a escribir en el ring buffer */
int **size**, /* Longitud de los datos */
TickType_t **ticks_to_wait** /* Tiempo máximo en ticks que la función esperará */
)

Retorna:

ESP_FAIL (-1) Error

Otro (>=0) El número de bytes empujados al I2C buffer.

Escribe bytes en el ring buffer interno de I2C. Cuando el TX FIFO está vacío, la ISR llena el FIFO con los datos del ring buffer.

Tarea:

- Revisar la documentación del ESP-IDF sobre I2C.
- Ejecutar los ejemplos vistos en clase.