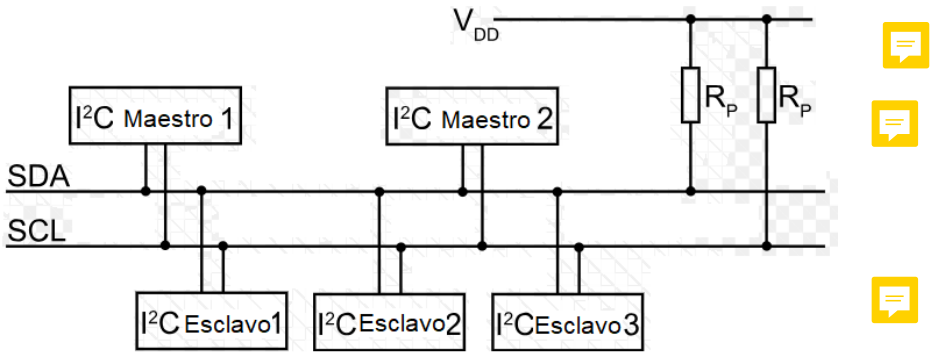
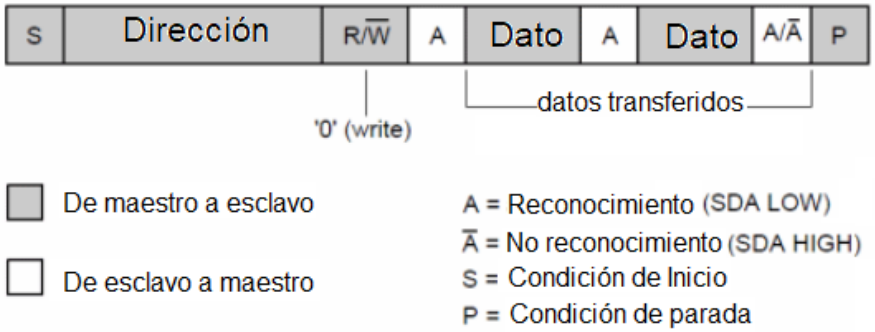
I2C (Inter Integrated Circuit)

I2C es un módulo de comunicación serial que se usa principalmente para comunicar el microcontrolador con uno o varios módulos periféricos o con otros microcontroladores mediante dos pines que son SD, para datos y SCL, para reloj. La ventaja del uso del I2C es que se puede comunicar el microcontrolador con varios módulos periféricos dispuestos en paralelo. En este tipo de comunicaciones existe el rol de maestro y esclavo, pero otra de las ventajas del módulo I2C es que este rol puede cambiar y en un momento dado puede pasar un módulo esclavo a ser un módulo maestro. I2C tiene dos frecuencias de comunicación: 100 khz y 400 khz.



Para poder hacer la comunicación del microcontrolador con múltiples módulos periféricos, a cada periférico se le asigna una dirección que viene programada por fábrica. Esta dirección es de 7 bits, pero además, en algunos casos, cada periférico tiene una dirección interna para ser más específica la dirección de cada uno. El protocolo de comunicación define una condición de inicio, luego una transmisión de dirección, que puede ser uno o dos bytes, enseguida un reconocimiento de llegada, y por último el dato con su respectivo bit de reconocimiento y una condición de parada. Toda información que se envie es de 8 bits. En el caso de una dirección de 7 bits, se envía un solo byte. En el caso de una dirección de 10 bits, se envían dos bytes. La dirección de 7 bits lo complementa un bit menos significativo (R/W) para indicar si el dato que sigue es para enviar desde el maestro o para recibir.



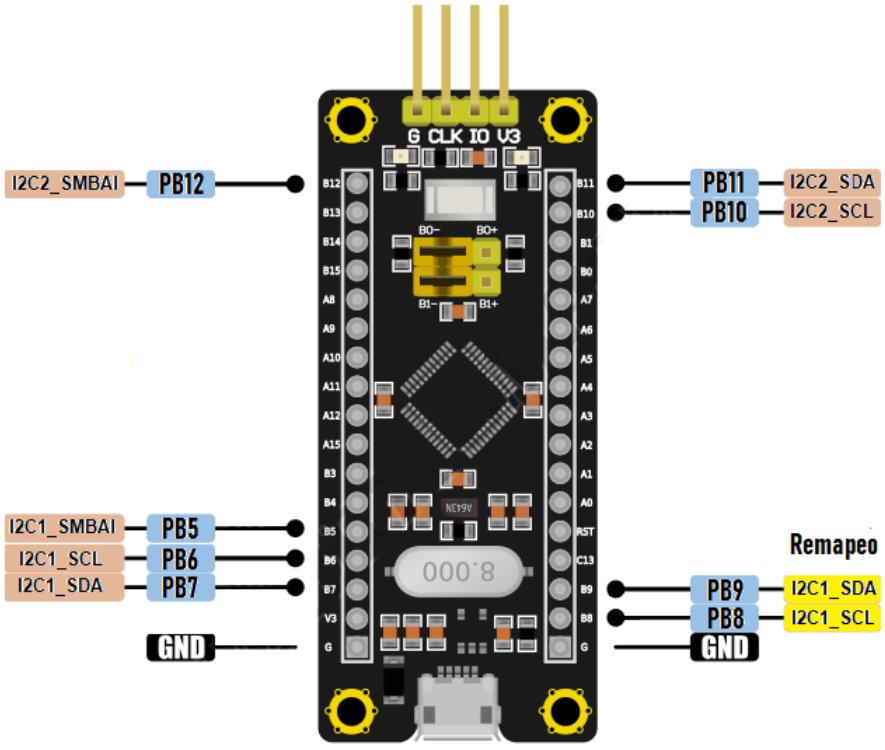
I2C Modo Esclavo.

Por defecto el microcontrolador está en modo esclavo. La frecuencia de entrada al I2C debe ser por lo menos de 2Mhz para una frecuencia de reloj de 100khz, frecuencia baja, sm, o de 4 Mhz, para una frecuencia de reloj de 400khz, frecuencia alta, fm. El reloj lo genera el maestro y lo pone en SL. Tan pronto se recibe la condición de inicio, en la línea SDA se empieza a recibir la dirección y luego esta se compara con la dirección del periférico, si hay una segunda dirección, también se tiene en cuenta o si hay una dirección de llamado general. En modo esclavo, el microcontrolador puede ser receptor o transmisor. Después de recibir o enviar el último dato y luego de emitir o recibir el bit de reconocimiento, se cierra la comunicación con el bit de parada.

I2C Modo Maestro.

El microcontrolador en modo maestro, genera el reloj por el pin SCL y se encarga de iniciar la comunicación con el bit de inicio. El Maestro como transmisor, envía la dirección por la línea SDA y espera por el bit de reconocimiento, luego envía datos al esclavo. Después de enviar el último dato, se cierra la comunicación con el bit de parada. El maestro como receptor, recibe datos del esclavo después de la dirección y emite el bit de reconocimiento. Igualmente, después del último dato recibido, el maestro cierra la comunicación con un bit de parada.

El microcontrolador STM32F103C8T6 tiene dos I2Cs, I2C1 e I2C2. La ubicación de estos dos periféricos con sus respectivos pines se muestra en la siguiente figura.



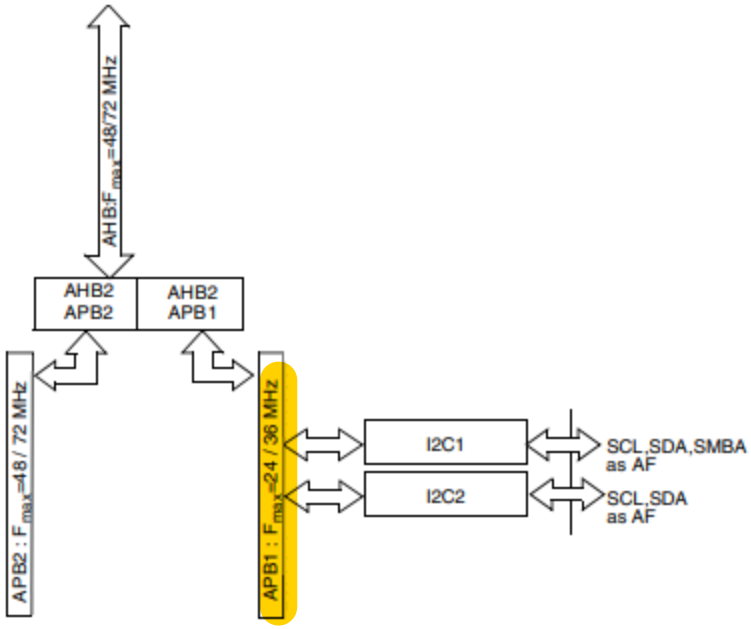
SCL es para el reloj y SDA para datos. Cada una de estas líneas debe tener una resistencia conectada a Vcc entre 1k y 10k. Sólo debe haber una resistencia, así se conecten varios dispositivos en paralelo. La configuración de estas líneas debe ser en modo de funciones

alternas y con drenaje abierto. Esto es para que el periférico pueda detectar si las líneas están siendo usadas por otro periférico al momento de iniciar la comunicación.

Para la configuración del módulo I2C se habilita el I2C poniendo el reloj respectivo, se configuran los pines que va a usar en modo de funciones alternas y drenaje abierto para cada pin y luego se programan las características del I2C. Para esto se usa una estructura del tipo `I2C_InitTypeDef`. Esta inicialización se puede hacer con la función `I2C_StructInit( I2C_InitStruct)` o asignando un valor a cada elemento de la estructura, que por defecto en la función de inicialización tiene los siguientes valores:

- `I2C_InitStruct.I2C_ClockSpeed = 100000;` // Velocidad de comunicación, menor o igual a 400kHz.
- `I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;` // Especifica el modo I2C.
- `I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;` // Define el ciclo útil para el reloj. Puede ser `I2C_DutyCycle_16_9` o `I2C_DutyCycle_2`.
- `I2C_InitStruct.I2C_OwnAddress1 = 0;` // Define la dirección, que puede ser de 7 o 10 bits.
- `I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;` // Habilita o no el bit de reconocimiento. Puede ser `I2C_Ack_Enable` o `I2C_Ack_Disable`.
- `I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;` // Indica si se reconoce una dirección de 7 o de 10 bits.

Con esta información en la estructura, se puede iniciar la configuración del I2C con la función `I2C_Init(I2Cx, &I2C_InitStruct)`, para el módulo I2C1 con x= 1, o I2C2 con x= 2. El STM32F103C8T6 tiene dos módulos I2C que son I2C1 y I2C2. Los dos módulos están unidos al bus APB1.



Un ejemplo de configuración del I2C2 es el siguiente.

```
void I2C2_Configuracion(void)
{
```

```
    I2C_DeInit(I2C2);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);

    // Configura pines I2C2: SCL y SDA
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_10 | GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    // características del i2c.
    I2C_InitTypeDef I2C_InitStructure;
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    I2C_InitStructure.I2C_OwnAddress1 = I2CMASTER_ADDR;
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = I2C1_CLOCK_FRQ;

    // habilita I2C2
    I2C_Cmd(I2C2, ENABLE);
    // aplica configuración I2C2
    I2C_Init(I2C2, &I2C_InitStructure);
}
```

La forma de comunicación de I2C es por estados, o sea, se debe monitorear el estado de la comunicación para seguir al próximo estado. En la librería SPL los estados para comunicación entre el maestro y el esclavo son los siguientes.

- El maestro observa la disponibilidad del bus y emite una condición de inicio. Pasa al estado `I2C_EVENT_MASTER_MODE_SELECT`.
- El maestro envía la dirección del esclavo con la función I2C_Send7bitAddress(). El bit menos significativo lo usa para que el dato sea para que lo envíe el maestro o el esclavo. Pasa entonces al siguiente estado que sería `I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED` o `I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED`.
- El esclavo reconoce su dirección y emitirá cualquiera de los siguientes dos estados: `I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED` o `I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED`.
- El maestro enviará el dato, si está como transmisor o recibirá el dato, si está como receptor y pasará al estado `I2C_EVENT_MASTER_BYTE_TRANSMITTED` o `I2C_EVENT_MASTER_BYTE_RECEIVED`.
- El esclavo recibirá el dato o enviará el dato que le pide el maestro y pasa al estado `I2C_EVENT_SLAVE_BYTE_RECEIVED` o `I2C_EVENT_SLAVE_BYTE_TRANSMITTING`. La librería SPL no garantiza el estado `I2C_EVENT_SLAVE_BYTE_TRANSMITTED`.
- Finalmente la comunicación se cierra por parte del maestro, si no hay más datos para enviar o recibir y el esclavo pasa al estado `I2C_EVENT_SLAVE_STOP_DETECTED`.

Un ejemplo para enviar un dato a una dirección interna del esclavo, es el siguiente.

```
void WriteByte(uint8_t address, uint8_t data)
{
  I2C_GenerateSTART(I2C2,ENABLE); // genea condición de inicio
  while (!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT));

  I2C_Send7bitAddress(I2C2, I2CSLAVE_ADDR, I2C_Direction_Transmitter);
```

```
  while (!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  I2C_SendData(I2C2,address);
  while (!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  I2C_SendData(I2C2,data);
  while (!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  I2C_GenerateSTOP(I2C2,ENABLE);

  while (!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  return;
}
```

Igualmente se presenta a continuación una función para leer desde el maestro al esclavo.

```
uint8_t mem_read(uint8_t reg_addr)
{
    I2C_AcknowledgeConfig(I2C2, ENABLE);
    while(I2C_GetFlagStatus(I2C2, I2C_FLAG_BUSY) == SET); // mientras esté ocupado

    I2C_GenerateSTART(I2C2, ENABLE); // envía condición de inicio
    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(I2C2, I2CSLAVE_ADDR, I2C_Direction_Transmitter); // envía dirección esclavo para
escribir
    while(!I2C_CheckEvent (I2C2, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    I2C_SendData(I2C2, reg_addr); // envía dirección registro interno
    while(!I2C_CheckEvent (I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_GenerateSTART(I2C2, ENABLE); // repite condición de inicio
    while(!I2C_CheckEvent (I2C2, I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(I2C2, I2CSLAVE_ADDR, I2C_Direction_Receiver); // envía dirección esclavo para
lectura
    I2C_AcknowledgeConfig (I2C2, DISABLE);

    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_RECEIVED));
    uint8_t read_reg = I2C_ReceiveData (I2C2);
    I2C_NACKPositionConfig(I2C2, I2C_NACKPosition_Current);
    for(uint32_t t= 0; t<100000; t++);
    I2C_GenerateSTOP (I2C2, ENABLE);

    return read_reg;
}
```

Se puede observar que debe esperar por cada estado del proceso para poder continuar. Como ejemplo de la función de interrupción del esclavo, se observa también la espera de cada evento. Aquí, usando el I2C1 como esclavo haciendo como si controlara una memoria para leer o escribir en ella.

```
void I2C1_EV_IRQHandler(void)
{
    uint32_t event;
    uint8_t wert;

    // lee el último evento
    event = I2C_GetLastEvent(I2C1);

    switch(event)
    {
        case I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED:
        {
            // el maestro ha enviado la dirección del esclavo para envío de datos al esclavo
            i2c1_mode = I2C1_MODE_SLAVE_ADR_WR;
            break;
        }
        case I2C_EVENT_SLAVE_BYTE_RECEIVED:
        {
            // el maestro ha enviado un byte al esclavo
            wert = I2C_ReceiveData(I2C1);

            // revisa dirección
            if(i2c1_mode == I2C1_MODE_SLAVE_ADR_WR)
            {
                i2c1_mode = I2C1_MODE_ADR_BYTE;
                // pone la dirección de memoria
                if(wert > 9) wert = 9;
                i2c1_ram_adr = wert;
            }
            else
            {
                i2c1_mode = I2C1_MODE_DATA_BYTE_WR;
                // guarda byte en memoria
                memoria[i2c1_ram_adr]= wert;
                //próxima posición
                i2c1_ram_adr++;
                if(i2c1_ram_adr > 9) i2c1_ram_adr= 9;
            }
            break;
        }
        case I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED:
        {
            // el maestro ha enviado la dirección del esclavo para leer un byte del esclavo
            i2c1_mode = I2C1_MODE_SLAVE_ADR_RD;
            // lee un byte de memoria
            wert = memoria[i2c1_ram_adr];
            // envía byte al maestro
            I2C_SendData(I2C1, wert);
            //próxima posición
            i2c1_ram_adr++;
            if(i2c1_ram_adr > 9) i2c1_ram_adr= 9;
            break;
        }
        case I2C_EVENT_SLAVE_BYTE_TRANSMITTING:
        {
            // en caso el maestro vaya a leer otro byte del esclavo
            i2c1_mode = I2C1_MODE_DATA_BYTE_RD;
```

```
            // lee byte de memoria
            wert =  memoria[i2c1_ram_adr];
            // envía byte al maestro
            I2C_SendData(I2C1, wert);
            // próxima posición
            i2c1_ram_adr++;
            if(i2c1_ram_adr > 9) i2c1_ram_adr= 9;
            break;
        }
        case I2C_EVENT_SLAVE_STOP_DETECTED:
        {
            //el maestro ha parado la comunicación
            I2C1_ClearFlag();
            i2c1_mode = I2C1_MODE_WAITING;
            break;
        }
        default:
        {
            break;
        }
    }
}
```

Las funciones disponibles en la librería SPL son las siguientes.

```
1.  void I2C_DeInit(I2C_TypeDef* I2Cx);
2.  void I2C_Cmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
3.  void I2C_DMACmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
4.  void I2C_DMALastTransferCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
5.  void I2C_OwnAddress2Config(I2C_TypeDef* I2Cx, uint8_t Address);
6.  void I2C_DualAddressCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
7.  void I2C_GeneralCallCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
8.  void I2C_ITConfig(I2C_TypeDef* I2Cx, uint16_t I2C_IT, FunctionalState NewState);
9.  uint16_t I2C_ReadRegister(I2C_TypeDef* I2Cx, uint8_t I2C_Register);
10. void I2C_SoftwareResetCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
11. void I2C_NACKPositionConfig(I2C_TypeDef* I2Cx, uint16_t I2C_NACKPosition);
12. void I2C_SMBusAlertConfig(I2C_TypeDef* I2Cx, uint16_t I2C_SMBusAlert);
13. void I2C_TransmitPEC(I2C_TypeDef* I2Cx, FunctionalState NewState);
14. void I2C_PECPositionConfig(I2C_TypeDef* I2Cx, uint16_t I2C_PECPosition);
15. void I2C_CalculatePEC(I2C_TypeDef* I2Cx, FunctionalState NewState);
16. uint8_t I2C_GetPEC(I2C_TypeDef* I2Cx);
17. void I2C_ARPCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
18. void I2C_StretchClockCmd(I2C_TypeDef* I2Cx, FunctionalState NewState);
19. void I2C_FastModeDutyCycleConfig(I2C_TypeDef* I2Cx, uint16_t I2C_DutyCycle);
20. ITStatus I2C_GetITStatus(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
21. void I2C_ClearITPendingBit(I2C_TypeDef* I2Cx, uint32_t I2C_IT);
```