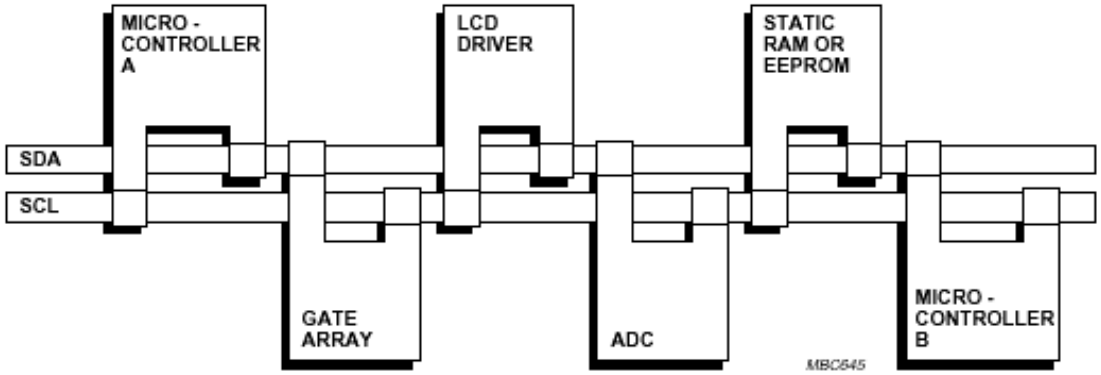
Lab Cortex-M4

I2C Oled Display

I2C protocol – Background

- Inter-Integrated Circuit Protocol
- I2C is a low-bandwidth, short-distance, two-wire interface for communication amongst ICs and peripherals
- Originally developed by Philips for TV circuits



- Only two bus lines are required
  - The SDA(for data) and SCL(for clock)
- Each device connected to the bus is software addressable
  - Devices can be 7-bit or 10-bit addressed

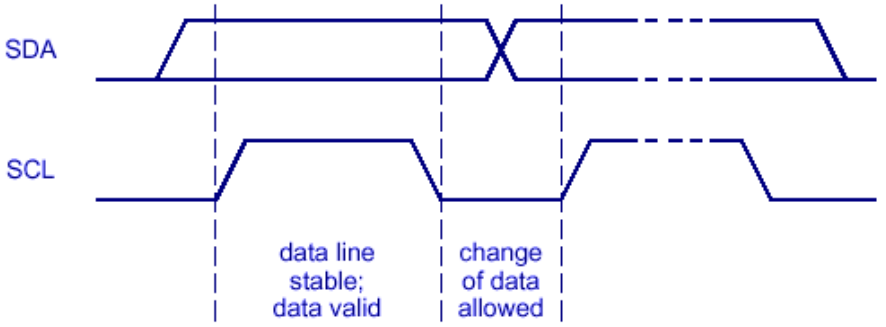
I2C Features

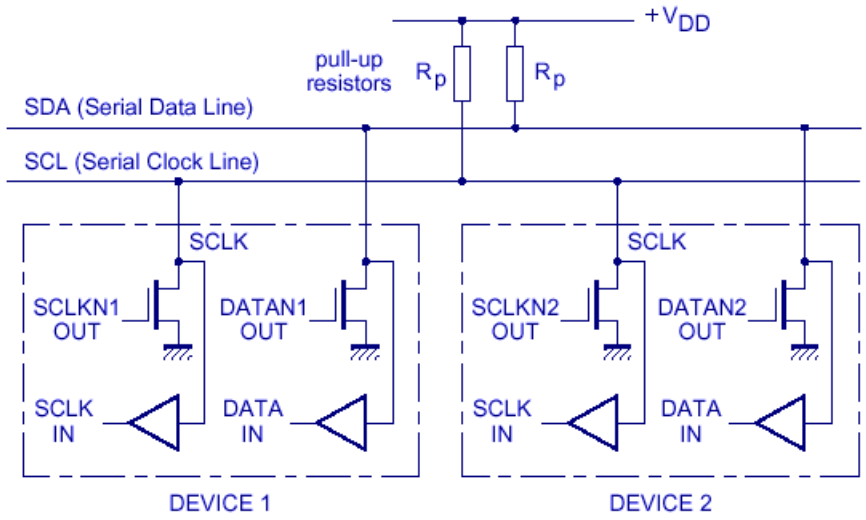
- Simple master-slave relation amongst devices (either can be receiver or transmitter)
- Multi-bus master collision detection and arbitration is supported
- Serial, 8-bit oriented, bi-directional data transfer can be achieved up to100 kbit/s (and up to 3.4Mbit/s in the high speed mode)

- Suppose Micro-controller A wants to send info to micro-controller B
  1. A (master) addresses B(slave)
  2. A (master-transmitter) sends data to B(slave-receiver)
  3. A terminates the transfer
- Suppose Micro-controller A wants to receive info from micro-controller B
  1. A (master) addresses B(slave)
  2. A (master-receiver) receives data from B(slave-transmitter)
  3. A terminates the transfer

I2C Bit-Transfer

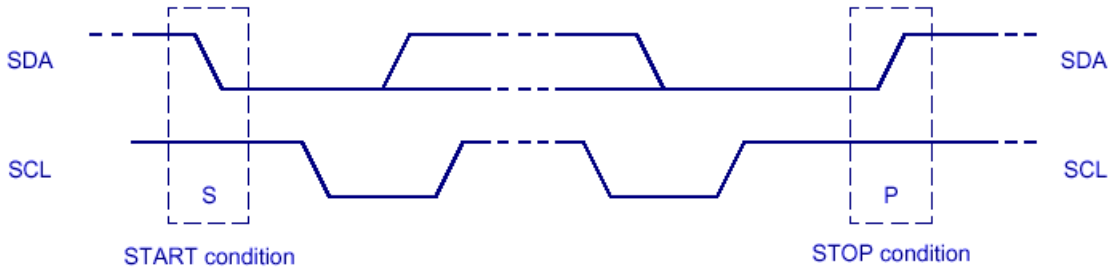
- One clock pulse is generated for each data bit that is transferred
- Data Validity
  - The data on the SDA line must be stable during the HIGH(1) period of the clock. The data line(SDA) can change data only when the clock signal (SCL) is LOW(0)
- Wired-and function
  - open-drain or open-collector





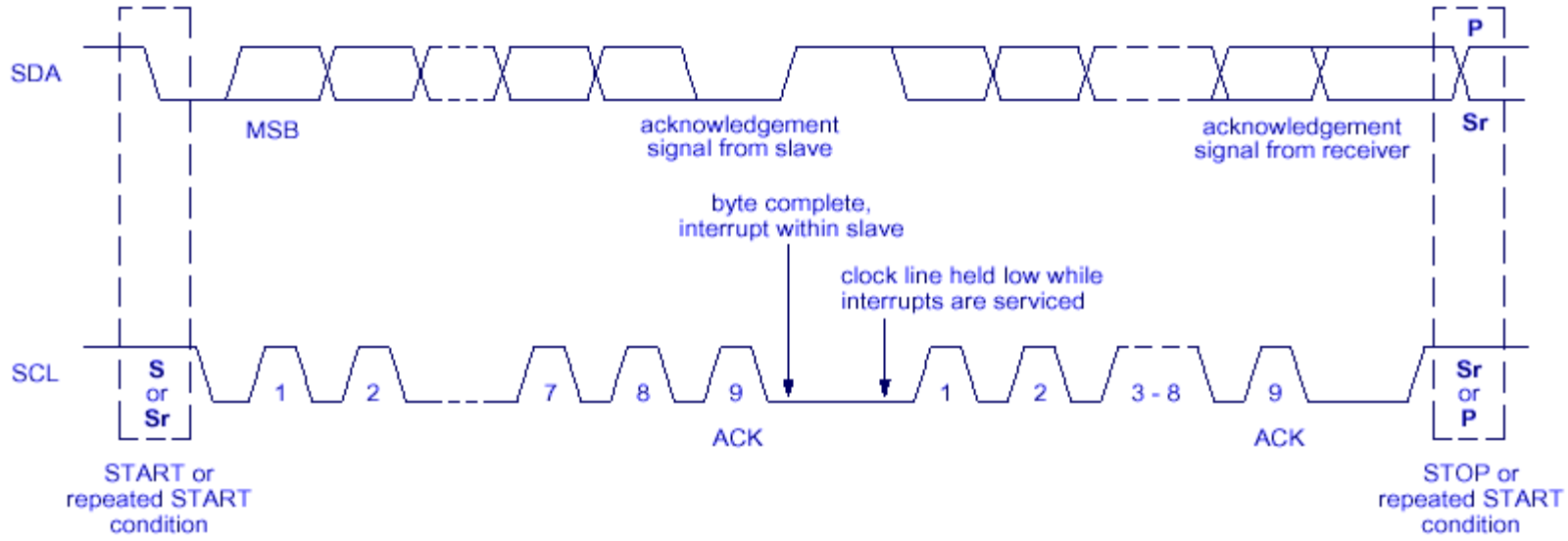
I2C START/STOP Conditions

- START condition: Signals begin of transfer (occupies the bus)
  - A HIGH to LOW transition on the SDA line while the SCL is HIGH
- STOP condition: Signals end of transfer (releases the bus)
  - A LOW to HIGH transition on the SDA line while the SCL is HIGH
- Both these are always generated by the Master
- Repeated START condition is allowed
  - Repeated start is used for changing the slave, or changing the direction of data transfer (Send/Receive) for the same slave



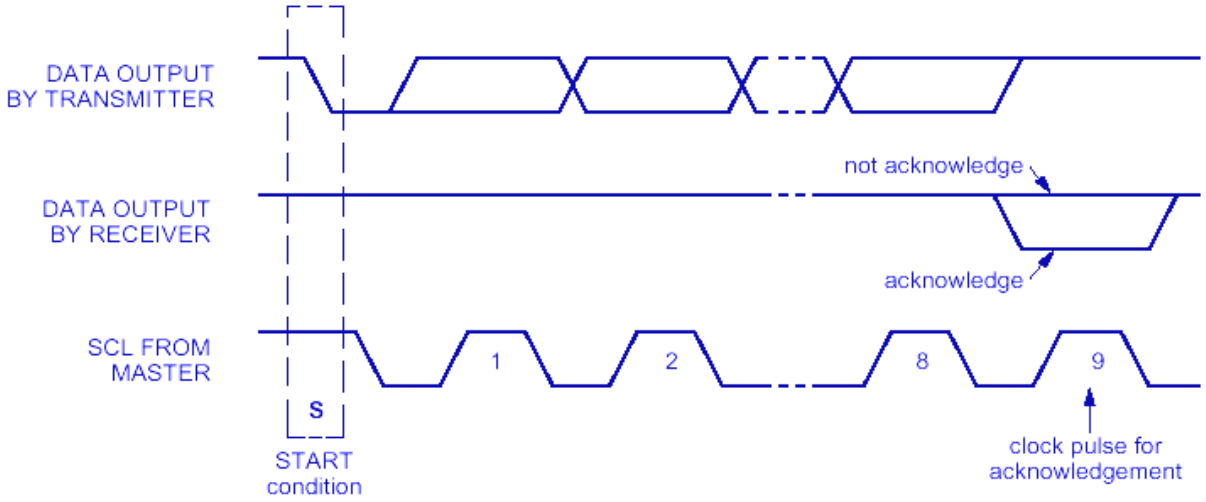
I2C Data Transfer

- Every byte on the SDA line must be 8-bits long
- Each byte must be followed by an acknowledgement from the receiver
- Data byte is transferred bit-wise with the MSB as the first bit sent
- A slave can force the master to wait by holding the clock line SCL LOW



Acknowledgement Scheme

- The acknowledge-related clock-pulse is generated by the master
- The transmitter (master or slave) releases the SDA line i.e. SDA is HIGH for the ACK clock pulse
- The receiver must pull-down the SDA line during the acknowledge clock pulse (stable LOW) during the HIGH period of the clock pulse

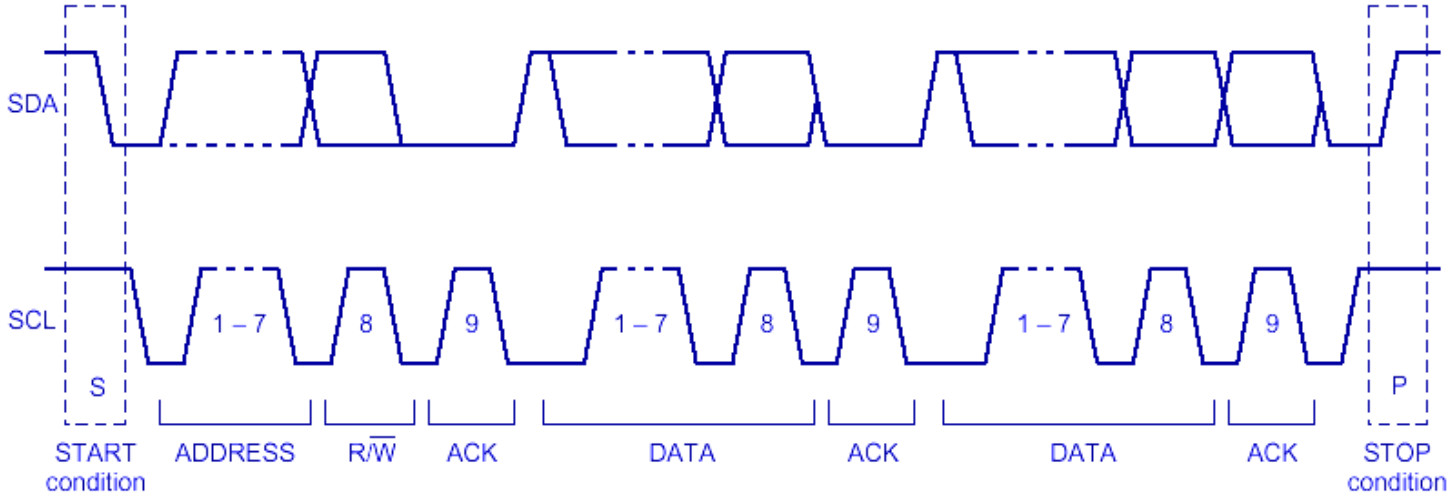


Acknowledgement Scheme

- The receiver is obliged to generate an acknowledge after each byte received.
- When a slave does not acknowledge slave address (when busy), it leaves the data line HIGH. The master then generates either STOP or attempts repeated START.
- If a slave-receiver does ack the slave address, but some time later during the transfer can not receive more data (this is done by leaving SDA HIGH during the ack pulse), then the master either generates STOP or attempts repeated START.
- If a master-receiver is involved in a transfer, it must signal the end of data to the slave-transmitter by not generating an ack on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

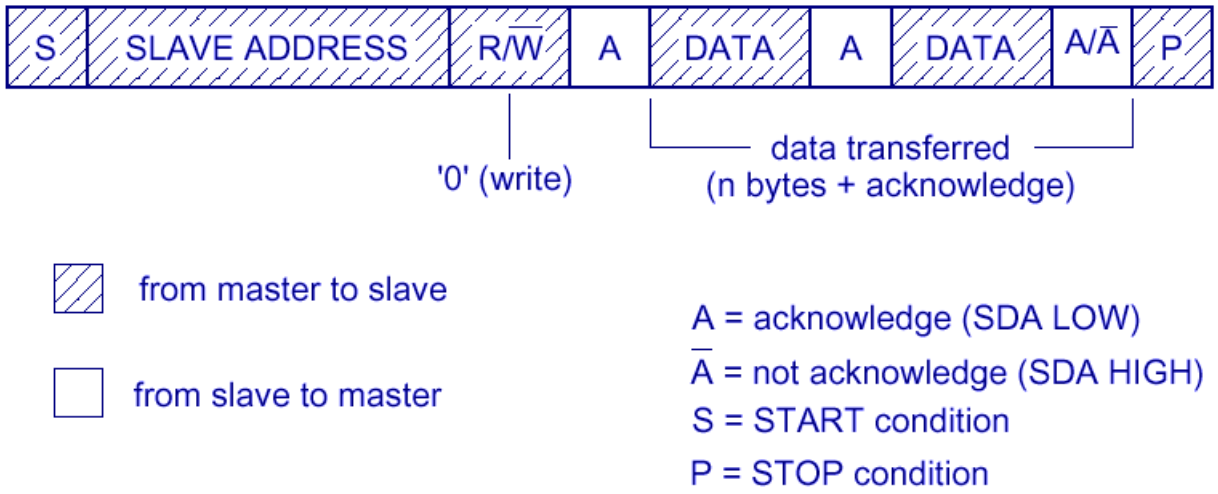
Data Transfer With 7-Bit Device Address

- After START condition (S), a slave address(7-bit) is sent.
- A read/write (R/W') direction is then sent(8th bit)
- Data transfer occurs, and then always terminated by STOP condition. However, repeated START conditions can occur.



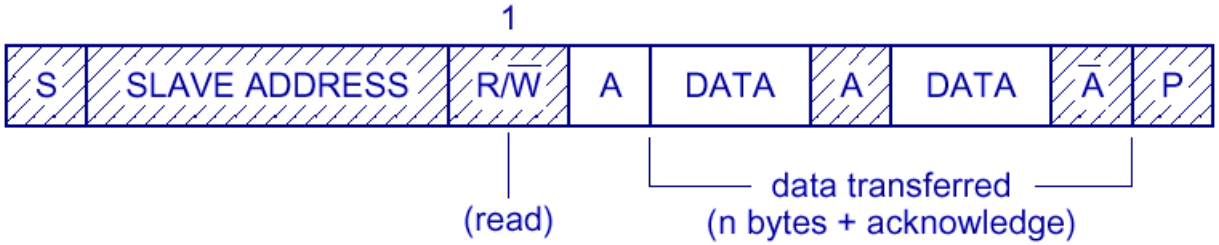
Master-Transmitter to Slave-Receiver Data Transfer

- In this, the transmission direction never changes. The set-up and transfer is straight-forward



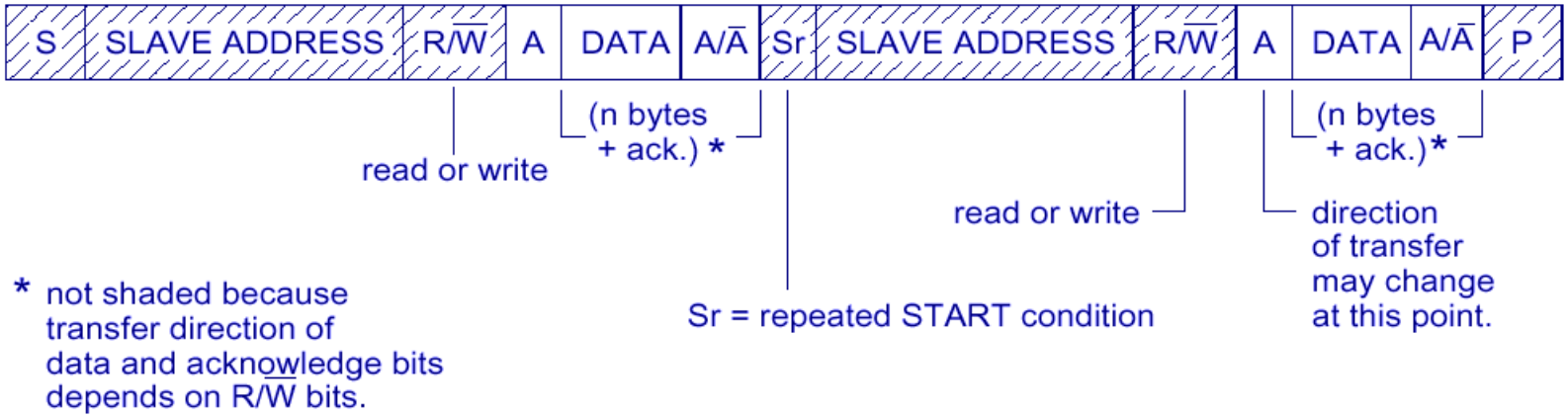
Master-Receiver and Slave-Transmitter Data Transfer

- Master initiates the data transfer by generating the START condition followed by the start byte (with read/write bit set to 1 i.e. read mode)
- After the first ack from the slave, the direction of data changes and the master becomes receiver and slave transmitter.
- The STOP condition is still generated by the master (master sends not-ACK before generating the STOP)



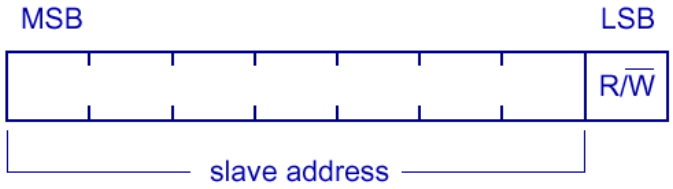
Read and Write in the Same Data Transfer

- Change in direction of data transfer can happen by the master generating another START condition (called the repeated START condition) with the slave address repeated
- If the master was a receiver prior to the change, then the master sends a not-ack (A') before the repeated START condition

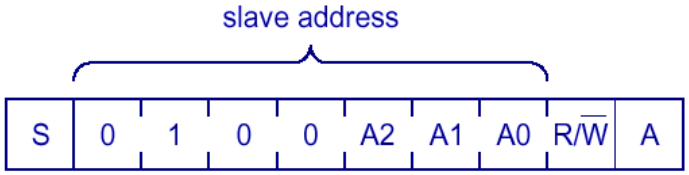


7-Bit Addressing

- The addressing info is contained in the first byte after the START condition
- The first 7 bits contain the address and LSB contains the direction of transfer(R/W' : 0 = write;1= read)

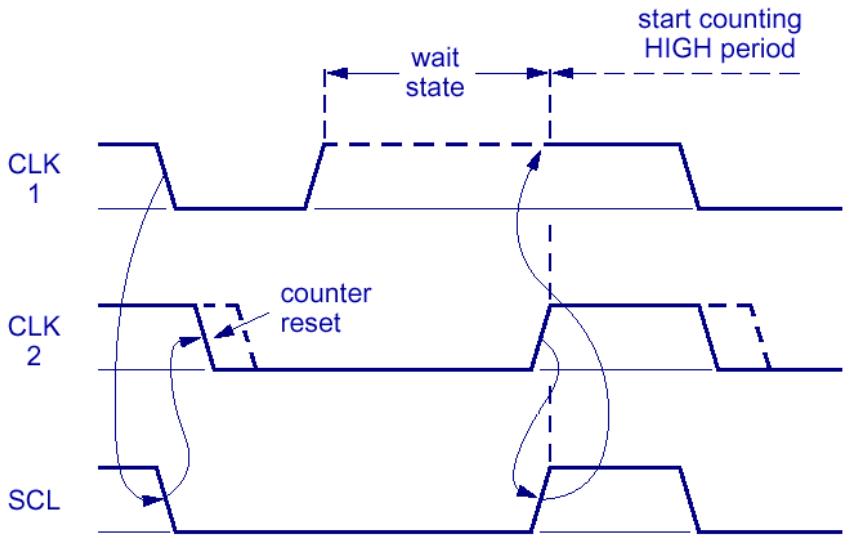


- When an address is sent, each device compares the first seven bits and considers itself addressed.
- A slave address can be made up of a fixed and a programmable part
  - PCF8575



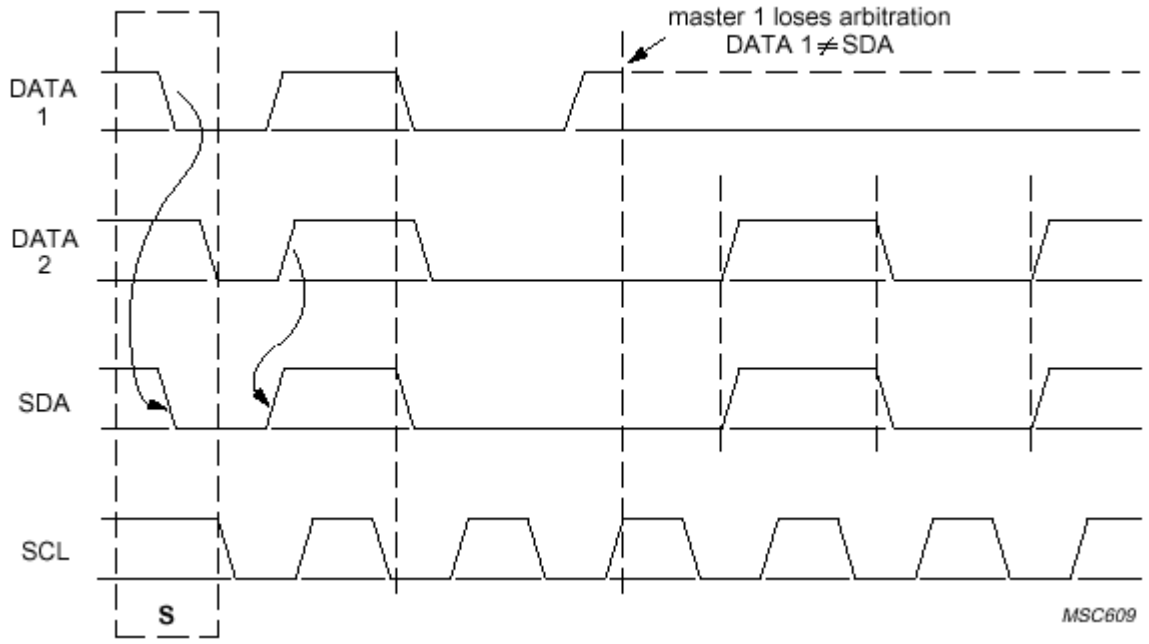
Multi-Master Clock Synchronization

- In the I2C bus, clock synchronization is performed using the wired-AND
  - If at least one master clock goes from HIGH to LOW, then the SCL is held LOW irrespective of the other masters' clock.
  - The SCL line goes to a HIGH state only when all the master clocks are in HIGH.
- The synchronized clock is generated with its LOW period determined by the device with the longest clock LOW period and its HIGH period determined by the one with the shortest clock HIGH period.



Multi-Master Arbitration Using the Clock Syn.

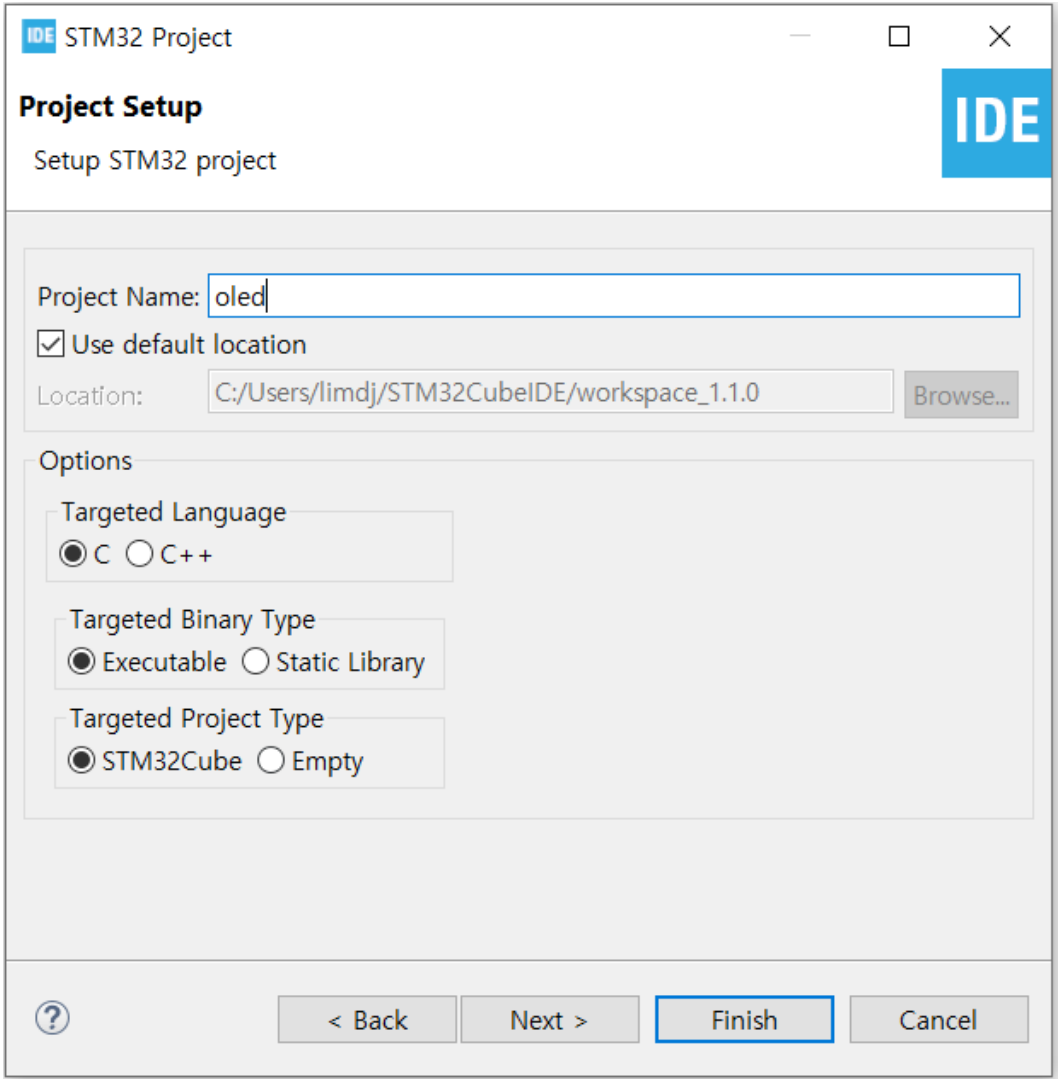
- If more than one device is capable of being a master, then an arbitration mechanism is needed to choose the master that takes control of the bus
- Arbitration takes place on the SDA, while the SCL is at the HIGH line,
  - the master which transmits a HIGH level,
  - another master is transmitting LOW level will switch off its DATA output stage because the level on the bus does not correspond to its own level.



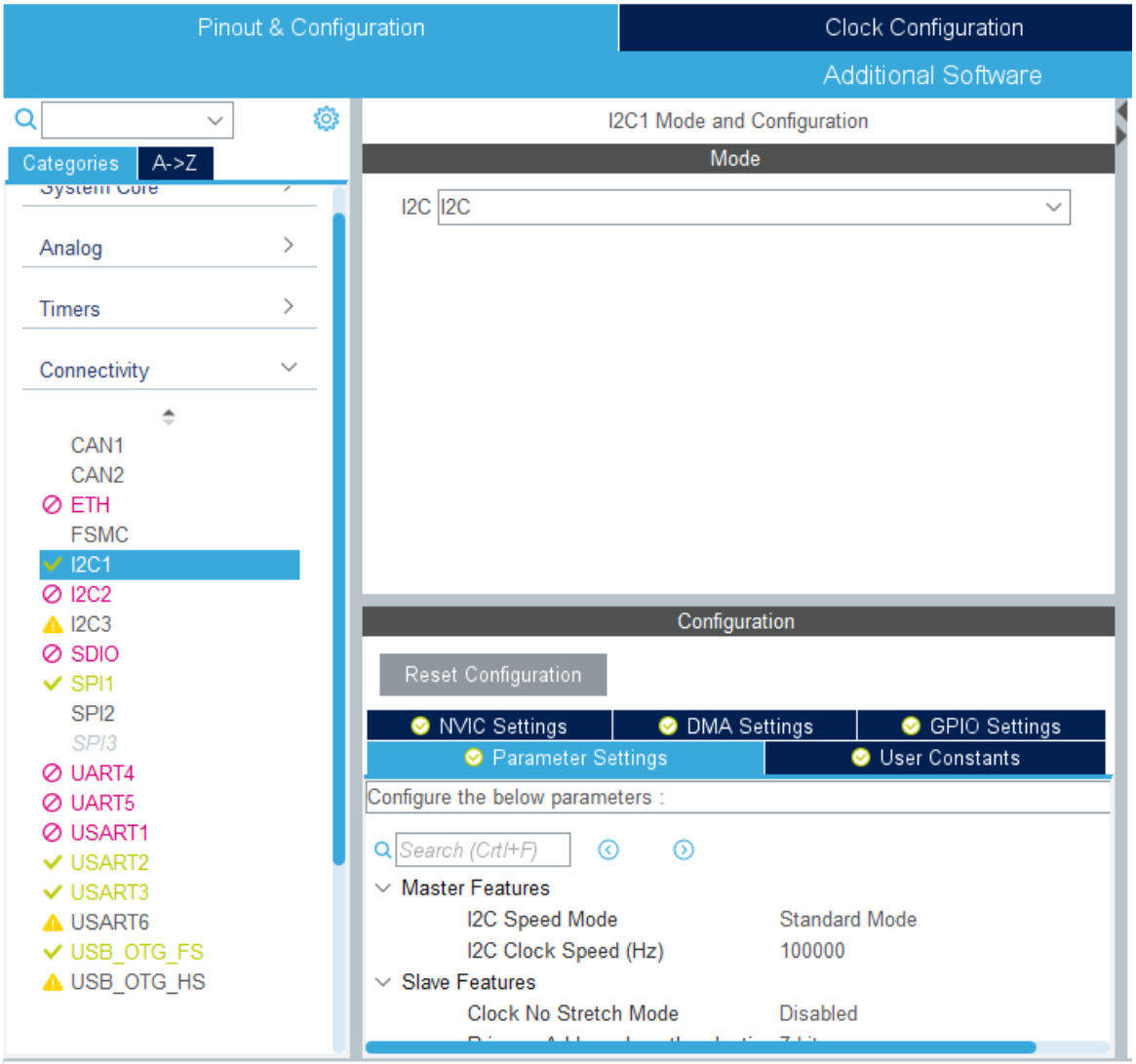
I2C Conclusion

- Compared to other serial bus protocols like SPI and Microwire
  - The pin (and connection) requirements are the least in I2C
  - The noise immunity is higher for I2C
  - There is a feedback to the the transmitter (Ack signal) for conveying the success of the transfer
  - I2C now has fast and high speed modes of operation

New STM32 Project: oled



- I2C, USART2, USART3



- Copy ssd1306.c and fonts.c files to C:\Users\limdj\STM32CubeIDE\workspace_1.1.0\oled\Core\Src
- Copy ssd1306.h and fonts.h files C:\Users\limdj\STM32CubeIDE\workspace_1.1.0\oled\Core\Inc

main.c

```
/* USER CODE BEGIN Includes */
#include "ssd1306.h"
#include "fonts.h"
/* USER CODE END Includes */

 /* USER CODE BEGIN 2 */
		ssd1306_Init();
		HAL_Delay(1000);
		ssd1306_Fill(Black);
		ssd1306_UpdateScreen();

		HAL_Delay(1000);

		ssd1306_SetCursor(0, 0);
		ssd1306_WriteString("Hello World", Font_11x18, White);
		ssd1306_SetCursor(0, 50);
		ssd1306_WriteString("ARM Cortex-M3", Font_7x10, White);
		ssd1306_UpdateScreen();
  /* USER CODE END 2 */
```

```
  /* USER CODE BEGIN WHILE */
              int counter;
              unsigned char string[10];
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
                string[0] = counter / 100 + 0x30;
                string[1] = (counter % 100) / 10 + 0x30;
                string[2] = (counter % 100) % 10 + 0x30;
                string[3] = 0;
                ssd1306_SetCursor(40, 20);
                ssd1306_WriteString(string, Font_16x26, White);
                counter++;
                if (counter > 999) counter = 0;
                ssd1306_UpdateScreen();
                HAL_Delay(10);
}
  /* USER CODE END 3 */
```

Logic Analyzer Capture

ssd1306.c

(global scope)

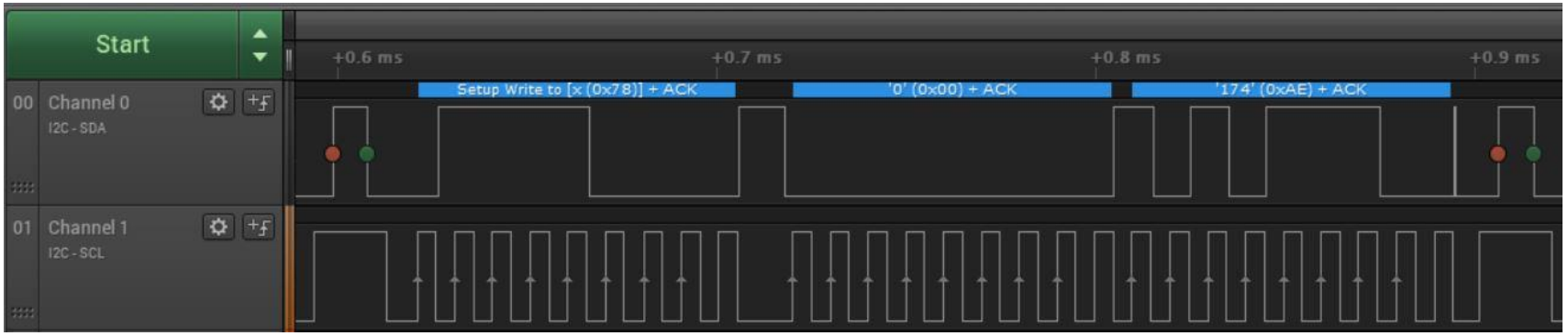
```
void ssd1306_WriteCommand(uint8_t command)
{
    HAL_I2C_Mem_Write(&hi2c1,SSD1306_I2C_ADDR,0x00,1,&command,1,10);
}
//
//  Het scherm initialiseren voor gebruik
//
uint8_t ssd1306_Init(void)
{
    // Even wachten zodat het scherm zeker opgestart is
    HAL_Delay(100);

    /* Init LCD */
    ssd1306_WriteCommand(0xAE); //display off
    ssd1306_WriteCommand(0x20); //Set Memory Addressing Mode
```

stm32f4xx_hal_i2c.c

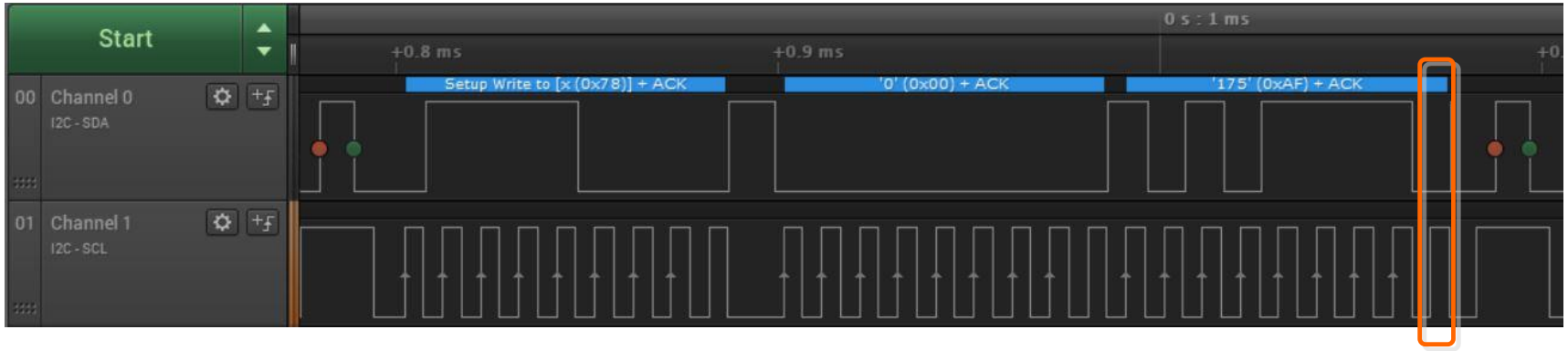
(global scope)

```
HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
{
```



Logic Analyzer Capture

- ACK



Logic Analyzer Capture

```
void ssd1306_UpdateScreen(void)
{
    uint8_t i;

    for (i = 0; i < 8; i++) {
        ssd1306_WriteCommand(0xB0 + i);
        ssd1306_WriteCommand(0x00);
        ssd1306_WriteCommand(0x10);

        // We schrijven alles map per map weg
        HAL_I2C_Mem_Write(&hi2c1,SSD1306_I2C_ADDR,0x40,1,&SSD1306_Buffer[SSD1306_WIDTH * i],SSD1306_WIDTH,100);
    }
}
```