Electronic Basics #19: I2C and how to use it

Introduction to I2C

I2C (Inter-Integrated Circuit) is a popular serial communication protocol used for connecting multiple low-speed devices such as sensors, EEPROMs, displays, and microcontrollers over two communication lines: SDA (Serial Data) and SCL (Serial Clock). It is a multimaster, multi-slave protocol, meaning that multiple devices can be connected to a single I2C bus, and the communication can be initiated by any device.

I2C allows communication between devices using only two wires, reducing complexity and saving space compared to other protocols like SPI**.**

I2C Structure

- **SDA (Serial Data):** This is the data line used for transferring data between devices.
- **SCL (Serial Clock):** This line carries the clock signal to synchronize data transfer between devices.
- **Master Device:** The device that controls the communication, initiating data transfer and generating the clock signal.
- **Slave Device:** Devices that respond to the master's requests. They have unique addresses on the bus.
- Pull-up Resistors: Both the SDA and SCL lines require pull-up resistors to ensure the lines are high when not actively driven low. Typical values are 4.7kΩ.

I2C operates in half-duplex mode, meaning data is transmitted in one direction at a time. The protocol defines specific addressing schemes for devices, and each device on the bus must have a unique address.

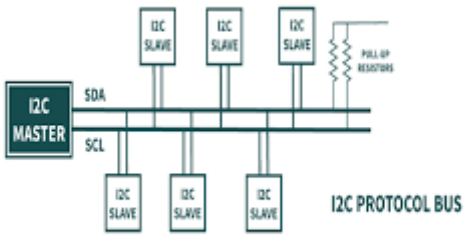

Fig19.1: I2C Protocol

I2C Addressing

Each I2C device is assigned a unique 7-bit address (or 10-bit in some cases). The address is used to identify which device is being communicated with. The 7-bit address is specified in the device's datasheet.

For example, if a device has a 7-bit address of 0x48, the communication would be directed to this device by using its address.

- The I2C Address is usually listed in the datasheet of the device.
- The address is transmitted by the master at the start of communication, and the slave responds to it.
- Some devices allow the address to be configured using pins (sometimes referred to as A0, A1, A2 pins).

How I2C Works

1. Master Initiates Communication: The master sends a start signal and the address of the slave device.
2. Slave Acknowledgment: The slave device responds with an acknowledge bit (ACK).
3. Data Transfer: Data is transferred between the master and slave. The master sends a clock signal to synchronize data transfer on the SDA line.
4. Stop Signal: The master sends a stop signal to end communication.

The master device controls the clock (SCL) while slaves respond to the data transfer requests.

Using I2C with Arduino Nano

To use I2C communication with an Arduino Nano, you need the Wire library, which is built into the Arduino IDE. This library simplifies the process of sending and receiving data over I2C.

Components Needed:

- Arduino Nano
- I2C Device (e.g., LCD display, sensor)
- Pull-up Resistors (typically 4.7kΩ)

Wiring:

1. Connect the SDA pin on the I2C device to the A4 pin on the Arduino Nano.
2. Connect the SCL pin on the I2C device to the A5 pin on the Arduino Nano.
3. Connect the VCC and GND pins of the I2C device to the 5V and GND pins on the Arduino Nano.
4. Add pull-up resistors (4.7kΩ) between the SDA and SCL lines and the 5V pin.

Arduino Code Example for I2C Communication

Here's an example code that demonstrates how to use I2C communication with an I2C-compatible device (e.g., an I2C LCD display):

```
#include <Wire.h>


void setup() {
  // Start I2C communication
  Wire.begin();  // Join I2C bus as master
  Serial.begin(9600); // Begin serial communication for debugging

  // Initialize LCD or other I2C device
  Wire.beginTransmission(0x3C);  // Address of the I2C device (e.g., 0x3C for an LCD)
  Wire.write(0x00);  // Command to initialize LCD
  Wire.endTransmission();
}

void loop() {
  // Send data to I2C device
  Wire.beginTransmission(0x3C);  // Address of the device
  Wire.write(0x01);  // Write command/data
  Wire.endTransmission();

  delay(1000); // Delay for 1 second
}
```

In this code:

- Wire.begin() initializes I2C communication.
- Wire.beginTransmission(address) starts communication with a device with a specific I2C address.
- Wire.write(data) sends data to the I2C device.
- Wire.endTransmission() ends the transmission.

Necessary Data from Datasheet

When working with an I2C device, you should refer to the datasheet of the component to gather necessary information such as:

- I2C Address: The unique 7-bit address of the device.
- Power Supply Requirements: The voltage and current requirements for the device.
- Commands and Data Registers: Information on how to write/read from the device.
- Timing Requirements: Clock speed, timing diagrams, and other relevant details that specify how communication should occur.

I2C Communication Speed and Limitations

- Standard Mode: Up to 100 kHz.
- Fast Mode: Up to 400 kHz.
- High-Speed Mode: Up to 3.4 MHz.

However, the actual speed may be limited by the devices involved and the length of the I2C bus.

Advantages and Disadvantages of I2C

Advantages:

- Low Pin Count: Only two wires (SDA and SCL) are used to connect multiple devices.
- Multiplexing: Supports multiple devices on a single bus.
- Simplicity: Easy to use and integrate with many components.

Disadvantages:

- Speed: I2C is slower than other protocols like SPI or UART.
- Address Conflicts: Multiple devices must have unique addresses; address conflicts can arise.
- Limited Distance: I2C is best suited for short-distance communication (a few meters).

Conclusion

I2C is a versatile and efficient protocol for communication between microcontrollers and peripheral devices, offering ease of use with only two wires. By utilizing the Wire library in Arduino, you can easily interface with I2C devices. When designing circuits using I2C, it's crucial to check the device datasheet for correct addressing, commands, and timing constraints.