

Application Note

Understanding the I²C Bus



Jonathan Valdez, Jared Becker

Abstract

The I²C bus is a very popular and powerful bus used for communication between a controller (or multiple controllers) and a single or multiple target devices. The figure below illustrates how many different peripherals may share a bus which is connected to a processor through only 2 wires, which is one of the largest benefits that the I²C bus can give when compared to other interfaces.

This application note is aimed at helping users understand how the I²C bus works.

The figure below shows a typical I²C bus for an embedded system, where multiple target devices are used. The microcontroller represents the I²C controller, and controls the IO expanders, various sensors, EEPROM, ADCs/DACs, and much more. All of which are controlled with only 2 pins from the controller.

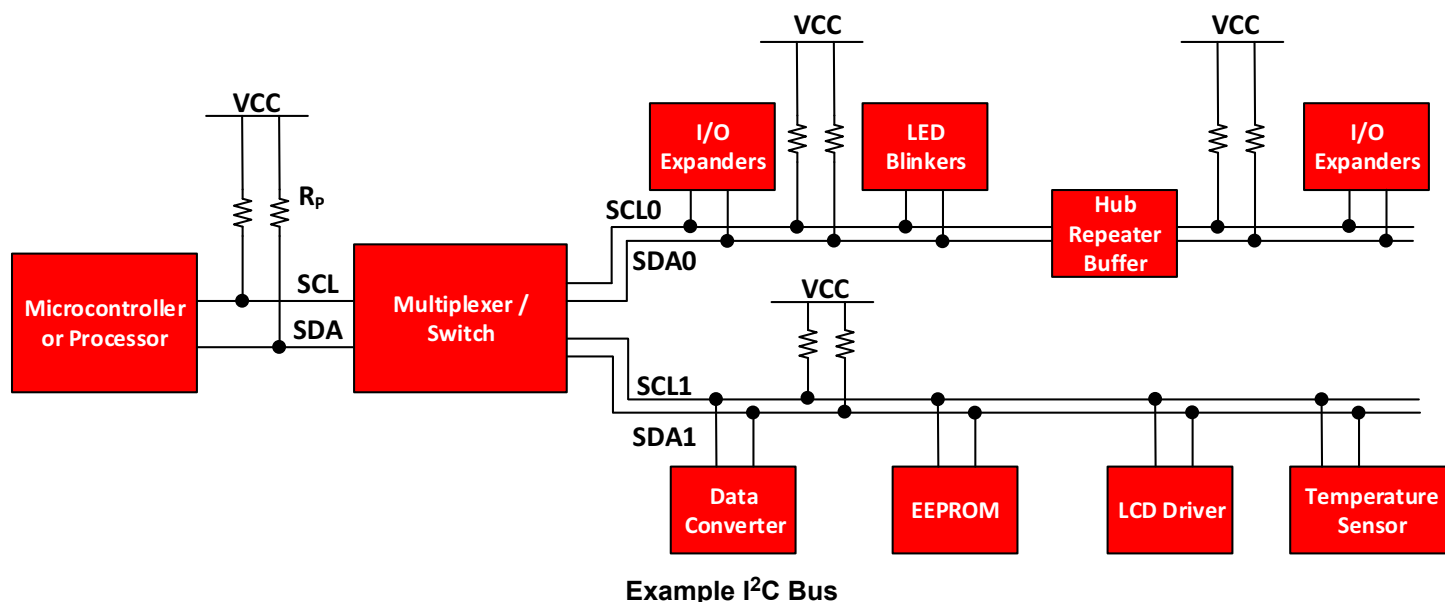


Table of Contents

1 Electrical Characteristics	2
1.1 Open-Drain for Bidirectional Communication	2
2 I²C Interface	3
2.1 General I ² C Operation	3
2.2 Data Validity and Byte Format	4
2.3 Acknowledge (ACK) and Not Acknowledge (NACK)	5
3 I²C Data	5
3.1 Writing to a Target On The I ² C Bus	6
3.2 Reading From a Target On The I ² C Bus	6
4 Revision History	7

Trademarks

All trademarks are the property of their respective owners.

1 Electrical Characteristics

I²C uses an open-drain/open-collector with an input buffer on the same line, which allows a single data line to be used for bidirectional data flow.

1.1 Open-Drain for Bidirectional Communication

Open-drain refers to a type of output which can either pull the bus down to a voltage (ground, in most cases), or "release" the bus and let it be pulled up by a pull-up resistor. In the event of the bus being released by the controller or a target, the pull-up resistor (R_{PU}) on the line is responsible for pulling the bus voltage up to the power rail. Since no device may force a high on a line, this means that the bus will never run into a communication issue where one device may try to transmit a high, and another transmits a low, causing a short (power rail to ground). I²C requires that if a controller in a multi-controller environment transmits a high, but see's that the line is low (another device is pulling it down), to halt communications because another device is using the bus. Push-pull interfaces do not allow for this type of freedom, which is a benefit of I²C.

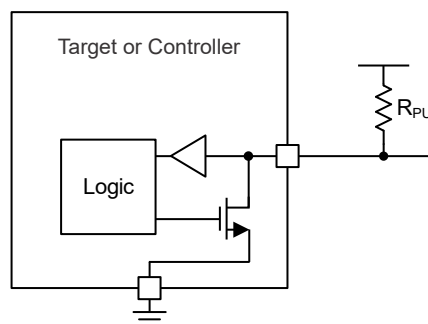


Figure 1-1. Basic Internal Structure of SDA/SCL Line

Figure 1-1 shows a simplified view of the internal structure of the target or controller device on the SDA/SCL lines, consisting of a buffer to read input data, and a pull-down FET to transmit data. A device is only able to pull the bus line low (provide short to ground) or release the bus line (high impedance to ground) and allow the pull-up resistor to raise the voltage. This is an important concept to realize when dealing with I²C devices, since no device may hold the bus high. This property is what allows bidirectional communication to take place.

1.1.1 Open-Drain Pulling Low

As described in the previous section, the Open-Drain setup may only pull a bus low, or "release" it and let a resistor pull it high. Figure 1-2 shows the flow of current to pull the bus low. The logic wanting to transmit a low will activate the pull-down FET, which will provide a short to ground, pulling the line low.

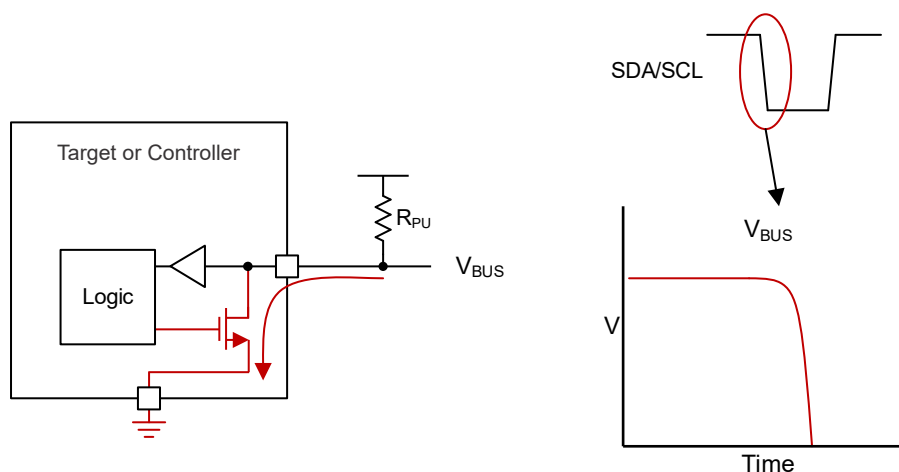


Figure 1-2. Pulling the Bus Low With An Open-Drain Interface

1.1.2 Open-Drain Releasing Bus

When the target or controller wishes to transmit a logic high, it may only release the bus by turning off the pull-down FET. This leaves the bus floating, and the pull-up resistor will pull the voltage up to the voltage rail, which will be interpreted as a high. Figure 1-3 shows the flow of current through the pull-up resistor, which pulls the bus high.

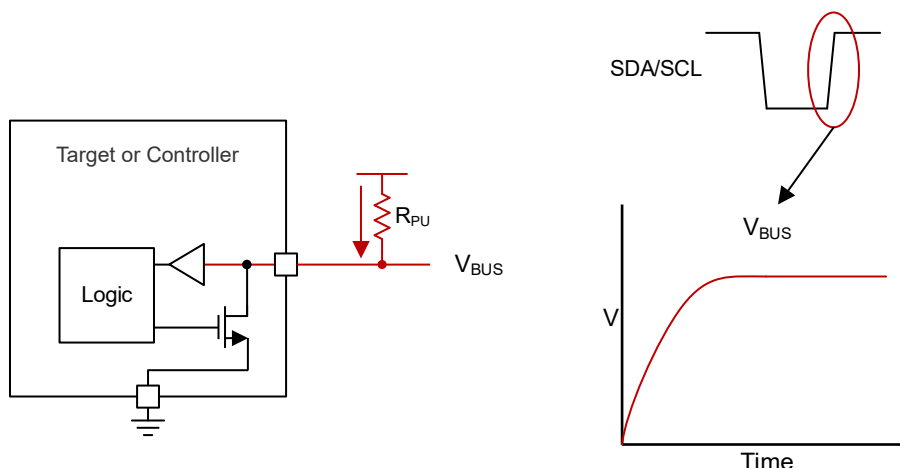


Figure 1-3. Releasing the Bus With An Open-Drain Interface

2 I²C Interface

2.1 General I²C Operation

The I²C bus is a standard bidirectional interface that uses a controller to communicate with target devices. A target cannot transmit data unless the target has been addressed by the controller. Each device on the I²C bus has a specific device address to differentiate between other devices that are on the same I²C bus. Many target devices require configurations upon startup to set the behavior of the device. This is typically done when the controller accesses the target's internal register maps, which have unique register addresses. A device can have one or multiple registers where data is stored, written, or read.

The physical I²C interface consists of the serial clock (SCL) and serial data (SDA) lines. Both SDA and SCL lines must be connected to V_{CC} through a pull-up resistor. The size of the pull-up resistor is determined by the amount of capacitance on the I²C lines (for further details, refer to *I²C Pull-up Resistor Calculation* (SLVA689)). Data transfer can be initiated only when the bus is idle. A bus is considered idle if both SDA and SCL lines are high after a STOP condition.

The general procedure for a controller to access a target device is the following:

1. Suppose a controller wants to send data to a target:
 - Controller-transmitter sends a START condition and addresses the target-receiver
 - Controller-transmitter sends data to target-receiver
 - Controller-transmitter terminates the transfer with a STOP condition
2. If a controller wants to receive/read data from a target:
 - Controller-receiver sends a START condition and addresses the target-transmitter
 - Controller-receiver sends the requested register to read to target-transmitter
 - Controller-receiver receives data from the target-transmitter
 - Controller-receiver terminates the transfer with a STOP condition

2.1.1 START and STOP Conditions

I²C communication with this device is initiated by the controller sending a START condition and terminated by the controller sending a STOP condition. A high-to-low transition on the SDA line while the SCL is high defines a START condition. A low-to-high transition on the SDA line while the SCL is high defines a STOP condition.

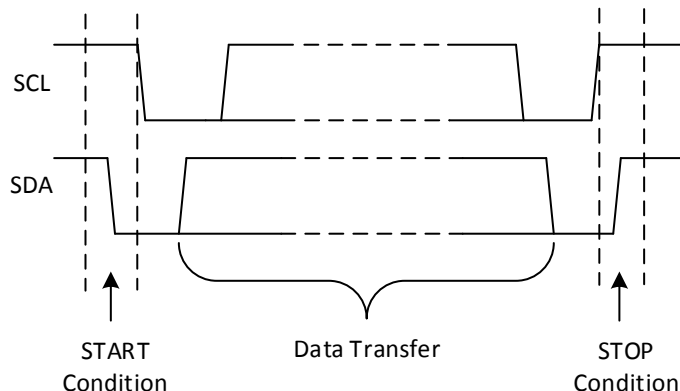


Figure 2-1. Example of START and STOP Condition

2.1.2 Repeated START Condition

A repeated START condition is similar to a START condition and is used in place of a back-to-back STOP then START condition. It looks identical to a START condition, but differs from a START condition because it happens before a STOP condition (when the bus is not idle). This is useful for when the controller wishes to start a new communication, but does not wish to let the bus go idle with the STOP condition, which has the chance of the controller losing control of the bus to another controller (in multi-controller environments).

2.2 Data Validity and Byte Format

One data bit is transferred during each clock pulse of the SCL. One byte is comprised of eight bits on the SDA line. A byte may either be a device address, register address, or data written to or read from a target. Data is transferred Most Significant Bit (MSB) first. Any number of data bytes can be transferred from the controller to target between the START and STOP conditions. Data on the SDA line must remain stable during the high phase of the clock period, as changes in the data line when the SCL is high are interpreted as control commands (START or STOP).

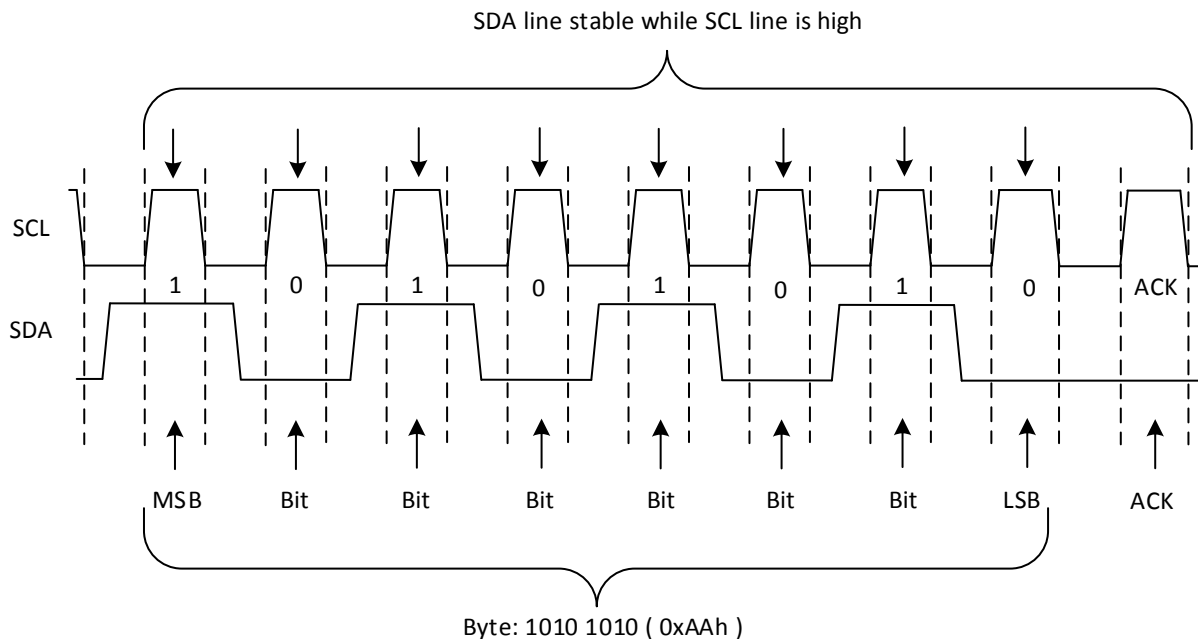


Figure 2-2. Example of Single Byte Data Transfer

2.3 Acknowledge (ACK) and Not Acknowledge (NACK)

Each byte of data (including the address byte) is followed by one ACK bit from the receiver. The ACK bit allows the receiver to communicate to the transmitter that the byte was successfully received and another byte may be sent.

Before the receiver can send an ACK, the transmitter must release the SDA line. To send an ACK bit, the receiver shall pull down the SDA line during the low phase of the ACK/NACK-related clock period (period 9), so that the SDA line is stable low during the high phase of the ACK/NACK-related clock period. Setup and hold times must be taken into account.

When the SDA line remains high during the ACK/NACK-related clock period, this is interpreted as a NACK. There are several conditions that lead to the generation of a NACK:

1. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the controller.
2. During the transfer, the receiver gets data or commands that it does not understand.
3. During the transfer, the receiver cannot receive any more data bytes.
4. A controller-receiver is done reading data and indicates this to the target through a NACK.

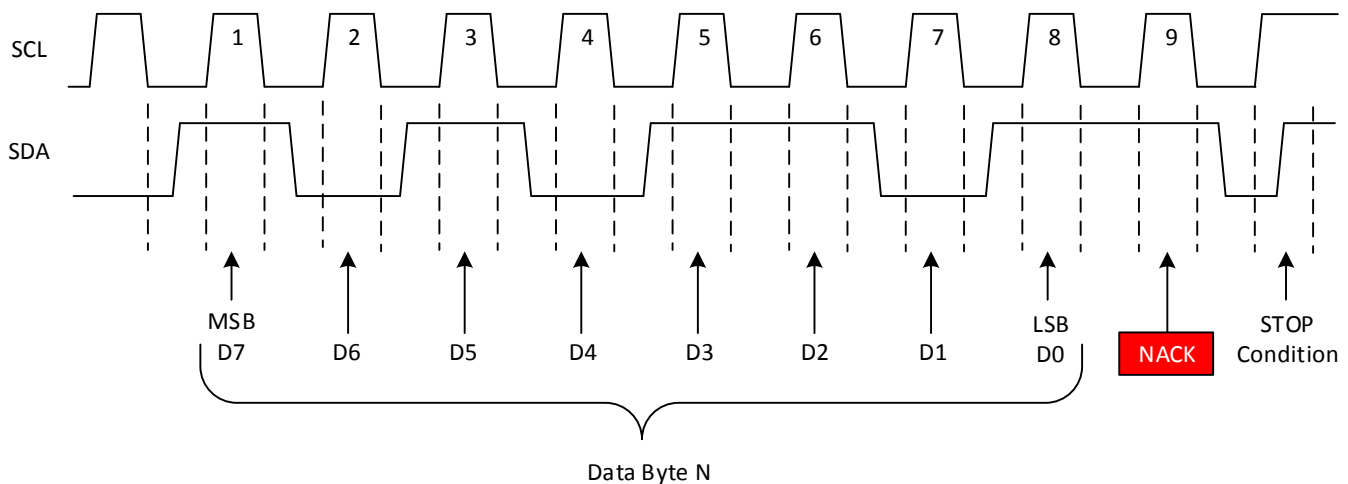


Figure 2-3. Example NACK Waveform

3 I²C Data

Data must be sent and received to or from the target devices, but the way that this is accomplished is by reading or writing to or from registers in the target device.

Registers are locations in the target's memory which contain information, whether it be the configuration information, or some sampled data to send back to the controller. The controller must write information into these registers in order to instruct the target device to perform a task.

While it is common to have registers in I²C targets, please note that not all target devices will have registers. Some devices are simple and contain only 1 register, which may be written directly to by sending the register data immediately after the target address, instead of addressing a register. An example of a single-register device would be an 8-bit I²C switch, which is controlled via I²C commands. Since it has 1 bit to enable or disable a channel, there is only 1 register needed, and the controller merely writes the register data after the target address, skipping the register number.

3.1 Writing to a Target On The I²C Bus

To write on the I²C bus, the controller sends a start condition on the bus with the target's address, as well as the last bit (the R/ $\overline{W}$ bit) set to 0, which signifies a write. After the target sends the acknowledge bit, the controller sends the register address of the register it wishes to write to. The target acknowledges again, letting the controller know it is ready. After this, the controller starts sending the register data to the target, until the controller has sent all the data it needs to (sometimes this is only a single byte), and the controller terminates the transmission with a STOP condition.

Figure 3-1 shows an example of writing a single byte to a target register.

-  Controller Controls SDA Line
-  Target Controls SDA Line

Write to One Register in a Device

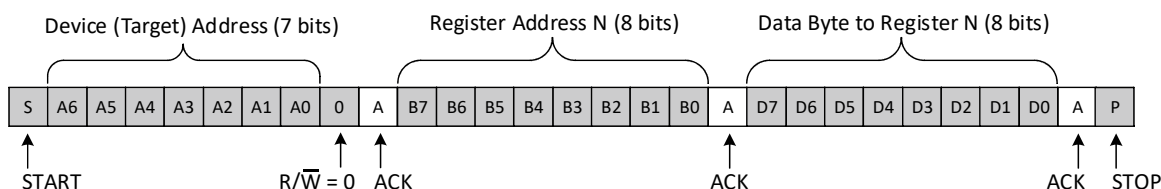


Figure 3-1. Example I²C Write to Target Device's Register

3.2 Reading From a Target On The I²C Bus

Reading from a target is very similar to writing, but with some extra steps. In order to read from a target, the controller must first instruct the target which register it wishes to read from. This is done by the controller starting off the transmission in a similar fashion as the write, by sending the address with the R/ $\overline{W}$ bit equal to 0 (signifying a write), followed by the register address it wishes to read from. Once the target acknowledges this register address, the controller sends a START condition again, followed by the target address with the R/ $\overline{W}$ bit set to 1 (signifying a read). This time, the target acknowledges the read request, and the controller releases the SDA bus, but continues supplying the clock to the target. During this part of the transaction, the controller becomes the controller-receiver, and the target becomes the target-transmitter.

The controller continues sending out the clock pulses, but releases the SDA line, so that the target can transmit data. At the end of every byte of data, the controller sends an ACK to the target, letting the target know that it is ready for more data. Once the controller has received the number of bytes it is expecting, it sends a NACK, signaling to the target to halt communications and release the bus. The controller follows up with a STOP condition.

Figure 3-2 shows an example of reading a single byte from a target register.

-  Controller Controls SDA Line
-  Target Controls SDA Line

Read From One Register in a Device

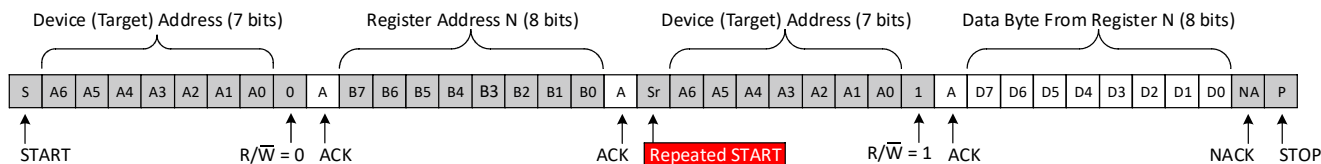


Figure 3-2. Example I²C Read from Target Device's Register

4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (June 2015) to Revision A (October 2026)	Page
• Updated the numbering format for tables, figures, and cross-references throughout the document.....	1
• Changed all instances of legacy terminology to controller and target where I ² C is mentioned.....	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025