Inter Integrated Circuit Bus: I^{2}C BUS

Dinesh Sharma

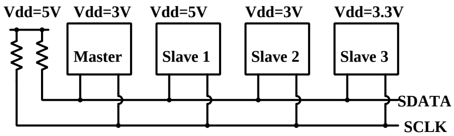
Department Of Electrical Engineering
Indian Institute Of Technology, Bombay

February 18, 2021

Synchronous Serial Data Transfer using I^{2}C Bus

- The I^{2}C bus is a 2 wire serial interface originally developed by Philips.
- The name stands for Inter Integrated circuit bus. Writing ‘I^{2}C’ is difficult in ordinary text, so it is often called the ‘I2C’ bus.
- Its main application is in data communication between ICs on a board – such as peripherals chips and processors and between smart home appliances.
- Subsequent to its introduction by Philips, Intel made some modifications to it to ensure inter-operability between devices from various manufacturers. That version is sometimes referred to as System Management Bus or SMBUS.
- 89C5131 includes modules to support the SMBUS.
- Restricted versions of this protocol are also known as the ‘2 wire protocol’.

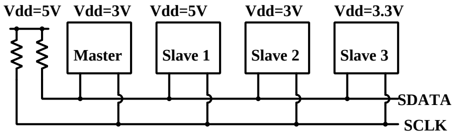
Synchronous Serial Data Transfer using I^{2}C Bus



Devices communicating with this protocol are connected in parallel across 2 wires, one of which carries serial data while the other carries the clock.

- All devices drive the bus lines using open drain drivers.
- Each line has a pull up resistor.
- Since devices provide only the pull down function, individual devices can use different supply voltages.
- The ability to connect devices using different supply voltages and the use of only 2 wires are the most attractive features of the I^{2}C Bus.

Synchronous Serial Data Transfer using I^{2}C Bus



The bus should have a master device, which provides the clock. Multiple slave devices can be connected to the bus.

- Each device has a unique address, which is normally 7 bit wide.
- The number of devices which can be connected across the bus is limited by the address space, or the maximum capacitive load of 400pF that can be placed on the bus.
- The master can act as the transmitter or the receiver.
- The addressed slave device then assumes a complementary role (receiver/transmitter).
- Multi master protocols are also supported.

I^{2}C Bus with multiple masters

- The I^{2}C bus protocol provides for multiple master deices on the bus and arbitration procedures for one of these to become the active master.
- Not all implementations provide this feature and those that do refer to it as I^{2}C bus with multi-master capability.
- A typical implementation will have the processor acting as the only master and several peripheral devices such as real time clocks, serial memories, ADCs and DACs acting as slaves.

Data speed on the I^{2}C Bus

- Because of its open drain drivers and resistive pull ups, this bus is used for low and medium speed communication.
- In the standard mode, Data is transferred at rates of up to 100 kbit/s on this bus.
- The protocol does not restrict one from using much lower clock frequencies.
- Data can be sent at up to 400 kbit/s in the Fast-mode and up to 1 Mbit/s in Fast-mode Plus.
- Recent versions of the bus protocol can support a speed of up to 3.4 Mbit/s in the High-speed mode.

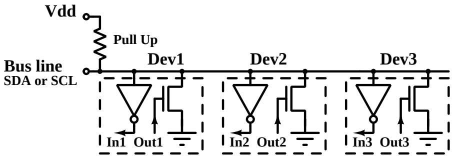
Evolution of I^{2}C Bus Specifications

Year	Version	Max. speed	Comments
1982	Original	100 kbit/s	Introduced by Philips
1992	1	400 kbit/s	Fast-mode and 10-bit addressing.
1998	2	3.4 Mbit/s	High-speed mode added.
2007	3	1 Mbit/s	Fast-mode plus with 20 mA drivers
2012	4	5 Mbit/s	Unidirectional Ultra Fast-mode (UFm) using push-pull logic without pull-up resistors

To use the 10-bit address scheme, the address frame consists of two bytes instead of one. A specific combination ('11110') of five most significant bits of the first byte is used to signal the 10-bit address mode. These 5 bits, the 10 bit address and the read/$\overline{\text{write}}$ bit make up the 16 bit address frame.

Open Drain drivers

Both lines (SDA and SCL) use open drain drivers. So it is worth recalling some characteristics of open drain logic.

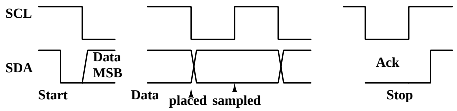


- Each driver has only a pull down transistor. Pull up is provided by the common pull up resistor on the bus.

- The bus can be 'High' only if *all* driver transistors are OFF.
- Any device can pull down the bus wire unconditionally.
- Different devices using different supply voltages can be easily connected as pull down devices.
- The bus driver transistor should be able to withstand the voltage provided by external V_{DD} – typically 5V.

Data transfer protocol

The actual data transfer is controlled by the Master, which provides the clock used for data transmission.

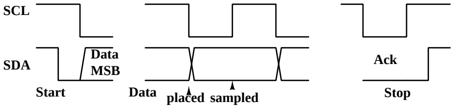


- It signals the 'Start' of transmission by a 'High' to 'Low' transition on the data line while the clock line is high.
- Similarly, the 'Stop' message is signaled by a 'Low' to 'High' transition on the data line while the clock line is high.

- These are the only instances when there are transitions on the data bus while the clock is high.
- Therefore these messages are easily distinguished from data transitions.

Data transfer protocol

For normal data transmission, the transmitter places data on the data line at the falling edge of clock.



Therefore all data transitions are seen after the clock goes low.

- The receiver samples data on the rising edge of the clock so that the data is stable at the time of sampling.
- In this protocol, the most significant bit is sent first.
- The master device can be the talker or the listener.
- Listener and talker roles are decided during the first message from the master.

Data transfer with master as the talker

Suppose the microcontroller is the master and it wants to send data serially to a slave device.

- The master generates the ‘Start’ message on the bus. (High SCL, ‘High’ to ‘Low’ transition on SDA).
- It then sends 8 bits on the serial data bus (Most significant bit first) which include the 7 bit address of the slave device and a $R/\overline{W}$ bit as the least significant bit.
- The last bit is 0 for a write operation (– Master being the talker in this example).
- After the transmitter has sent these 8 bits on the SDA line, the transmitter driver transistor goes off during the ninth bit time.
- The receiver pulls the data line ‘Low’ during this time to acknowledge successful receipt of the signal. (Recall that this is possible because of the open drain connection discussed earlier).

Data transfer with master as the talker

When the master is the talker, it generates the start condition, then sends the 7 bit address of the slave, followed by a ‘0’ to indicate that the master is the talker and the slave is the listener.

- During the ninth bit time, the driver transistor of the master is turned off and the receiver should pull down the SDA line if it received the data successfully.
- This is called the ‘Ack’ message.
- If the receiver driver transistor is also ‘OFF’ during the ninth bit period, the SDA line will be seen to be ‘High’ during this time.
- This is known as a ’Nack’ message. It is used by the slave to signal failure to receive when it is a listener (during write operation by master).

Data transfer with master as the talker

To summarize:

- The first transmission from the master establishes it as the talker and the slave as the listener if the last bit sent from the master is '0'.
- After the address has been acknowledged by the receiver, the master, (which is the talker), sends data serially to the slave receiver, checking for the 'Ack' signal at the end of each byte.
- After all bytes have been sent, the transmission is terminated by a 'Stop' signal on the line.

Data transfer with master as the listener

- When the master wants to be the listener, it creates the start condition, followed by a 7 bit address and then a ‘1’.
- As before, during the 9th bit time, the slave should pull down the SDA line to acknowledge that the byte has been received.
- After receiving the ‘Ack’ signal, the master releases the SDA line (turns off its open drain driver). It continues to drive SCL.
- The slave now becomes the transmitter and sends data. Each data byte is acknowledged by the master after reading it, through an ‘Ack’ signal.
- After reading the last byte, the master sends the ‘Nack’ signal to stop the slave from transmitting any further.
- The master follows up the ‘Nack’ signal with a stop condition to terminate this round of communication.

Using 10 bit addresses

The examples above used a 7 bit address for identifying devices. However the I^2C bus permits one to use 10 bit addresses in an upward compatible mode.

- To use the 10-bit address scheme, the address frame uses two bytes instead of one.
- Of the 16 bits provided by these two bytes, the first five are used to signal that a 10 bit address will be used instead of a 7 bit address.
- A specific combination ('11110') of five most significant bits of the first byte is used to signal the 10-bit address mode.
- Therefore addresses which use this combination as the 5 most significant bits are not allotted even in 7 bit address mode.
- The first byte contains 11110XXY, where XX are the two most significant bits of the 10 bit address and Y is the read/$\overline{\text{write}}$ bit.
- The second byte contains the remaining 8 bits of the address.

Clock stretching using SCL

- I^{2}C bus specification allows a slave to slow down the nominal data transfer rate.
- An addressed slave device may hold the clock line (SCL) low after receiving (or sending) a byte. (This is made possible by the open drain driver design).
- This indicates that it is not yet ready to process more data.
- The master that is communicating with the slave must wait until the clock line is released by the slave.
- The master must wait until it observes the clock line going high, and an additional minimal time (4μs for standard 100 kbit/s speed) before pulling the clock low again.
- Clock stretching is the only time in I^{2}C bus specification where the slave drives the SCL line.

Clock stretching using SCL

- In principle, clock stretching can be used during the transmission of any bit.
- In practice, it is used most commonly in the interval just before or after the acknowledge bit.
- After receiving a byte, the listener needs to check whether the byte was received properly and whether to send an ‘Ack’ or a ‘Nack’. This checking may required more than a bit time.
- In this case, the listener can stretch the last bit, while it decides whether to send an ‘Ack’ or a ‘Nack’ and then release the clock.
- The SMB specification limits the time for which a clock period may be stretched.

Arbitration in multi-master configurations

In a multi-master configuration, all masters monitor the bus for start and stop events and intervene only when the bus is idle.
(After stop, before a start condition).

- No clash will occur when a master has acquired the bus and other masters will wait till they see the 'stop' signal on the bus.
- However, once 'stop' has been seen, two master might start transmission on the idle bus at the same time. This needs to be resolved using arbitration.
- When this happens, the two masters will drive the bus to a 'start' condition at the same time. (Otherwise, the master starting later would have detected the start condition and would have held off).
- This means the two master drive SCL more or less simultaneously, and the 'low' duration of the SCL line will be a little longer (while either of them is driving it low).
- Since a master starts operation with an addrss frame, the clash will occur when the addrss is being sent.

Arbitration in multi-master configurations

If the two master seize the bus at the same time and address different slaves, the clash can be detected during the transmission of the address frame.

- If one transmitter sets SDA to 1 (driving transtor off) and a second transmitter sets it to 0 (driving trnasistor on), the result is that the line is low.
- This disparity can be noticed by the master which was driving the line to a ‘1’ and sees a ‘0’ on the line instead.
- This master is now required to relinquish the bus and try to communicate only after the next ‘stop’ event.
- Notice that a driver trying to drive a line to ‘0’ will see no disparity because a line will definitely be pulled low when a device sends a ‘0’ irrespective of what the other drivers are doing.

Arbitration using SDA

The master which observes that the line is at ‘0’, whereas it was driving it to a ‘1’, loses arbitration and stops driving the SDA line as well as the SCL line and waits for a STOP event.

- Since the losing master was sending a ‘1’ while the winning transistor was sending a ‘0’, the one sending the lower slave address always “wins” arbitration in the address stage.
- The slave with a ‘0’ in its address frame will receive it correctly, and communication will continue between the winning master (which was addressing this slave anyway) and this slave.

Arbitration using SDA

What if both masters were addressing the same slave?

- If two masters were addressing the same slave, they will not notice that some other device is also driving the bus – since they are sending the same address frame.
- In this case, arbitration extends into the data transfer stage.
- The same logic will be applied to data now, and the master which was sending a ‘1’ will lose arbitration to the one which was sending a ‘0’.