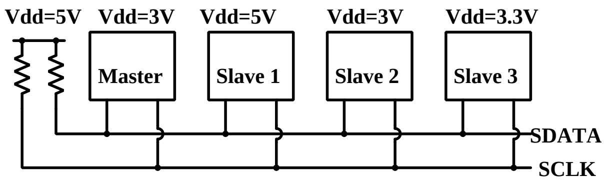
Synchronous Serial Data Transfer: I^{2}C Bus

Last time, we had seen how asynchronous serial data transfer is used. This time we shall use a synchronous data transfer protocol known as I^{2}C protocol.

1 Introduction

The I^{2}C bus is a 2 wire serial interface original 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. It is used widely for connecting peripherals to processors and by smart home appliances to communicate with each other. 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. We shall be using built in SMBUS modules in the microcontroller for this experiment. Restricted versions of this protocol are also known as the '2 wire protocol'.

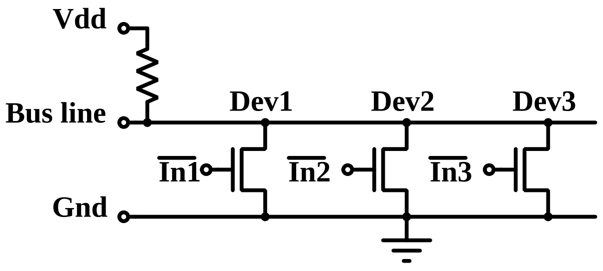
Devices communicating with this protocol are connected in parallel across 2 wires, one of which carries serial data while the other carries the clock.



Each device has a unique address, which is normally 7 bit wide. All devices drive the bus lines using open drain drivers. Each line has a pull up resistor.

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 device provides the clock. It can act as the transmitter or the receiver. The addressed slave device then assumes a complementary role (receiver/transmitter). The 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 multi-master capability. Data is normally sent at 100 Kilobits per second, though some systems use a 'fast' version which runs at 400 Kbps. The protocol does not restrict one from using much lower clock frequencies.

Since both lines use open drain drivers, 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.

Also, different devices using different supply voltages can be easily connected to each other, since the drivers provide only a pull down function. This is a desirable feature when connecting devices from different manufacturers.

The actual data transfer is controlled by the Master. The Master 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. 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.

Suppose the microcontroller is the master and it wants to send data serially to a slave device. It generates the 'Start' message on the bus. 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/W bit as the least significant bit. This bit is 0 for a write operation. After the transmitter has sent these 8 bits on the data 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). This is called the 'Ack' message. If the receiver driver transistor is also 'OFF' during the ninth bit period, the data line will be seen to be 'High' during this time. This is known as a 'Nack' message. 'Nack' is used by the slave to signal failure to receive when it is a listener (Write operation by master). After the address has been acknowledged by the receiver, master, which is the transmitter, 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 of the line.

When the master is the listener, it sends 'Ack's after receiving each data byte (except the last one), inviting the slave talker to send the next byte. It sends 'Nack' after the last byte is received and follows it up with a 'Stop' condition to terminate data transfer.

Please refer to the accompanying document (written by Yogesh Patil) for details about using the protocol for this problem.

2 Problem Statement

We shall use the I2C protocol to connect the new 51S card with a real time clock chip DS 1307, which provides data using I2C. You can use either C or Assembly to do the programming.

1. Connect up DS 1307 to the card using appropriate port lines. (Remember to provide power and ground lines to DS 1307, apart from Serial Data and clock!)
2. Program the card to read data from the real time clock. Also, use a port pin to generate a ‘0’ to ‘1’ transition at start of each transfer and a ‘1’ to ‘0’ transition at the end. Use this signal as the trigger for the oscilloscope and display the data or clock as the other channel.
3. Modify your program to read the time from the chip and display it on the LCD display.