I2C Notes

Overview

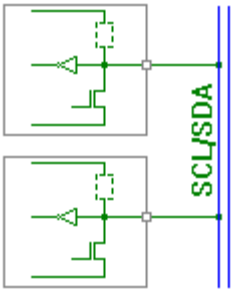
IC2 is a multi-master/slave I2C uses 2 lines, SCL (clock) and SDA (data).

The bus is active low, this means that a HIGH is only weak in that if any other node holds the line LOW it cannot be raised until the LOW has been released.

100 Kbits/s standard mode
400 Kbits/s fast mode

Max capacitance = 400 pF
Max number of nodes = infinite, but must not exceed max capacitance

The bus interface is built around an input buffer and an open drain or open collector transistor.



When idle the bus lines are in the logic HIGH State. Note here that an external PULL-UP resistor is necessary. To put something on the BUS the chip drives its output transistor, thus pulling the BUS to a LOW level. When the bus is IDLE both lines are HIGH. The pull up resistor in the devices is actually a small current source or even non-existent. The nice thing about this concept is that it has a 'built-in' bus mastering technique. If the bus is 'occupied' by a chip that is sending a LOW then all other chips loose their comm's capability.

Protocol Overview

START, ADDRESS, ACKNOWLEDGE, DATA, STOP

Idle condition

SDA = HIGH
SCL = HIGH

Start condition

SCL = HIGH
SDA = HIGH -> LOW transition

Stop condition

SCL = HIGH
SDA = LOW –> HIGH transition

Acknowledgement

After a slave

Data

8bits long MSB first, each byte has to be followed by an acknowledgement bit ACK.

SDA HIGH->LOW transition, LOW->HIGH transition
SCL = LOW

SDA valid
SCL = HIGH

Receiver Busy = receiver takes SCL LOW
Receiver free = receiver takes SCL HIGH

ACK bit

Transmitter SDA = HIGH

Receiver SDA = LOW

Addressing

Address = 7bits + R/W bit

Write = LOW R/W bit
Read = HIGH R/W bit

Clock and synchronisation

All masters generate their own clock. There are 2 mechanisms for synchronisation:

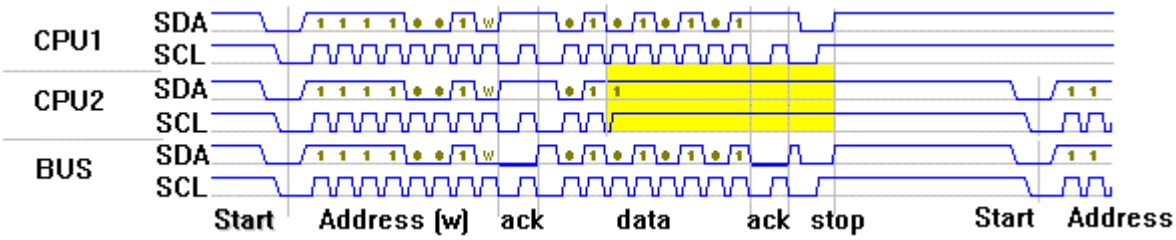
One form of synchronisation mechanism works on the SCL line only. The Slave who want's it master to wait simply pulls the pulls the SCL low as long as needed.
The master is then not able of giving the ACK clock pulse because it cannot raise the SCL line. Of course the master software must check this condition and act appropriately. In this case the master simply waits until it can raise the SCL line and then just goes on with whatever it was doing.

The other mechanism can be used to can prevent another master from taking over the bus. In a 2 master system this is not handy. But the moment you get 3 or more masters this is very useful. A third master cannot interrupt a transfer between master 1 and 2 in this way. For some mission critical situations this can be very handy.
You can make this technique rigid by not pulling only the SCL line low, but also the SDA line. Then any master who is not in the 2 masters talking to each other will immediately back off. Before you continue you first make SDA back high and then SCL and go on. Any master, which attempted to communicate in the mean time, would have detected a BACKOFF situation and would be waiting for a STOP to appear.

Bus Arbitration

The bus structure is a wired AND (if one device pulls a line low it stays low) you can test for bus occupation. When you (as a MASTER) change the state of a line to HIGH, you MUST always check that it has gone to HIGH. If it stays low then BACKOFF, it's occupied. Some other device is pulling the line low. So the general rule of thumb is if you can't get a certain line high then back off and wait until a stop condition is seen before resuming. This back off condition has to be maintained until a valid STOP condition has been seen on the bus. Then and only then an attempt can be made to start talking again.

The first rule says that you loose arbitration when you cannot raise either SCL or SDA. It is the device that is sending the LOW that rules the bus. You cannot disturb the other CPU's transmission because if you can't raise one of the lines you back off, and if it is the other one that can't raise one of the lines they have to back-off.
This back off condition will only occur the moment that the 2 transmitted levels are not the same.



The 2 CPU's are accessing a slave in write mode at address `1111001`. The slave ACK's this.

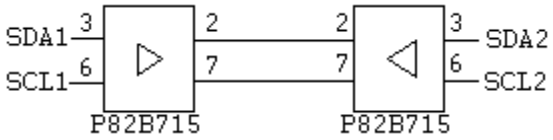
CPU1 wants to transmit `01010101` to the slave while CPU 2 wants to transmit `01100110` to the slave. The moment that the data bits does not match anymore (what the CPU sends are different then what is present on the bus.) One of them has to loose arbitration. Obviously this is the CPU which did not get his data on the bus. For as long as there has been no STOP present on the bus he won't touch the bus and leave the SDA and SCL line high. (Yellow zone).
The moment a STOP was detected CPU2 can attempt to transmit again.

So from the above story we can conclude that is the one that is pulling the line LOW that always wins. The one which wanted the line to be HIGH when it is being pulled low by the other looses the BUS .We call this a loss of arbitration or a BACKOFF condition.

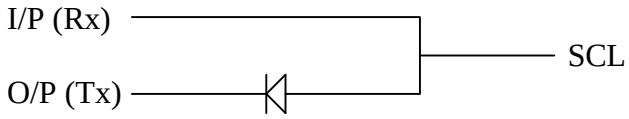
When the CPU gets a BACKOFF situation then it has to wait for a STOP condition to appear on the bus. Then it knows that the transmission has been completed.

I2C drivers

Many microcontrollers, memory IC's etc have an I2C interface built in. If the device you are using doesn't then an I2C bus extender is usually required unless the transmission distance is less than a few metres. With Phillips 82B715 distances of around 30 metres can be achieved. The 82B715 are essentially 2 buffer circuits for SCL and SDA.



Driving I2C from a PC parallel port



The same circuit is used for SDA. So in total there will be 4 pins used on the parallel port, 2 input and 2 output.