



I2C 2005-1 Demonstration Board

I²C-bus Protocol

Oct, 2006

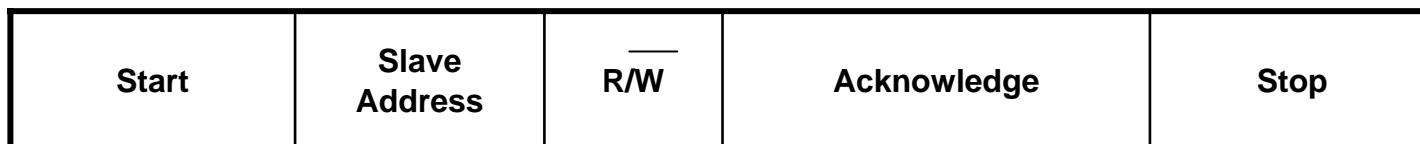


I²C Introduction

- I²C-bus = Inter-Integrated Circuit bus
- Bus developed by Philips in the early 80s
- Simple bi-directional 2-wire bus:
 - serial data (SDA)
 - serial clock (SCL)
- Has become a worldwide industry standard and used by all major IC manufacturers
- Multi-master capable bus with arbitration feature
- Master-Slave communication; two-device only communication (generally)
- Each IC on the bus is identified by its own address code
- The slave can operate as a:
 - receiver (slave – receiver)
 - transmitter (slave – transmitter)

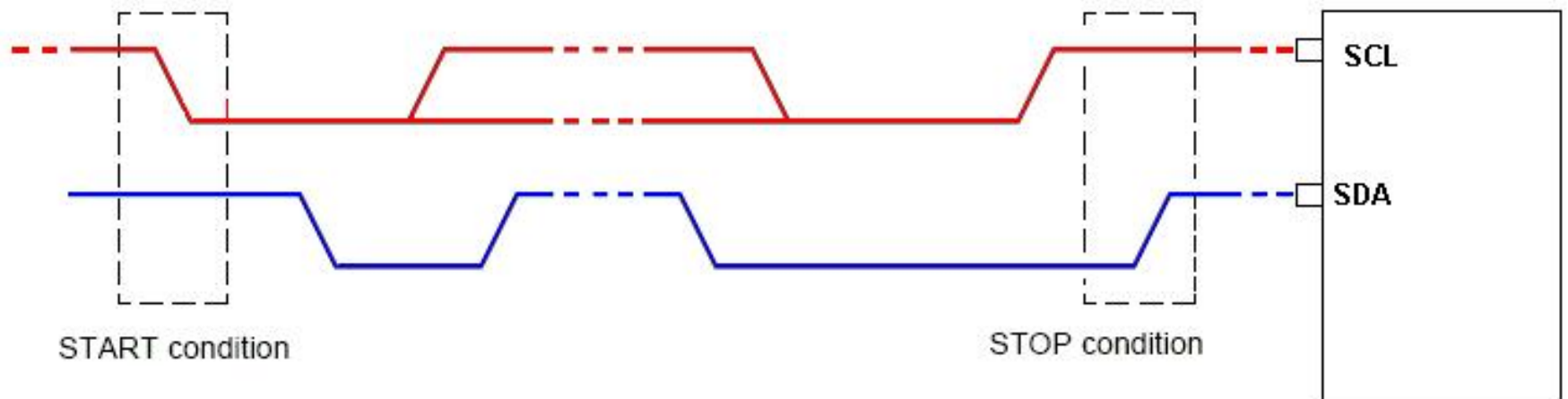


I²C-Bus Protocol Basic

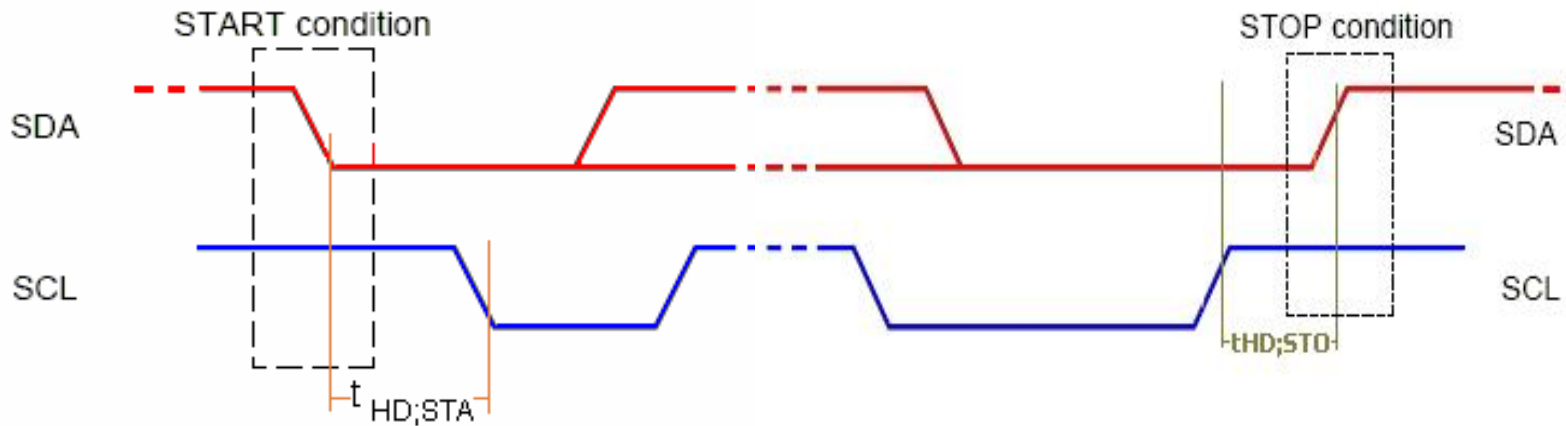


START and STOP Conditions

Question: What is wrong with this figure?



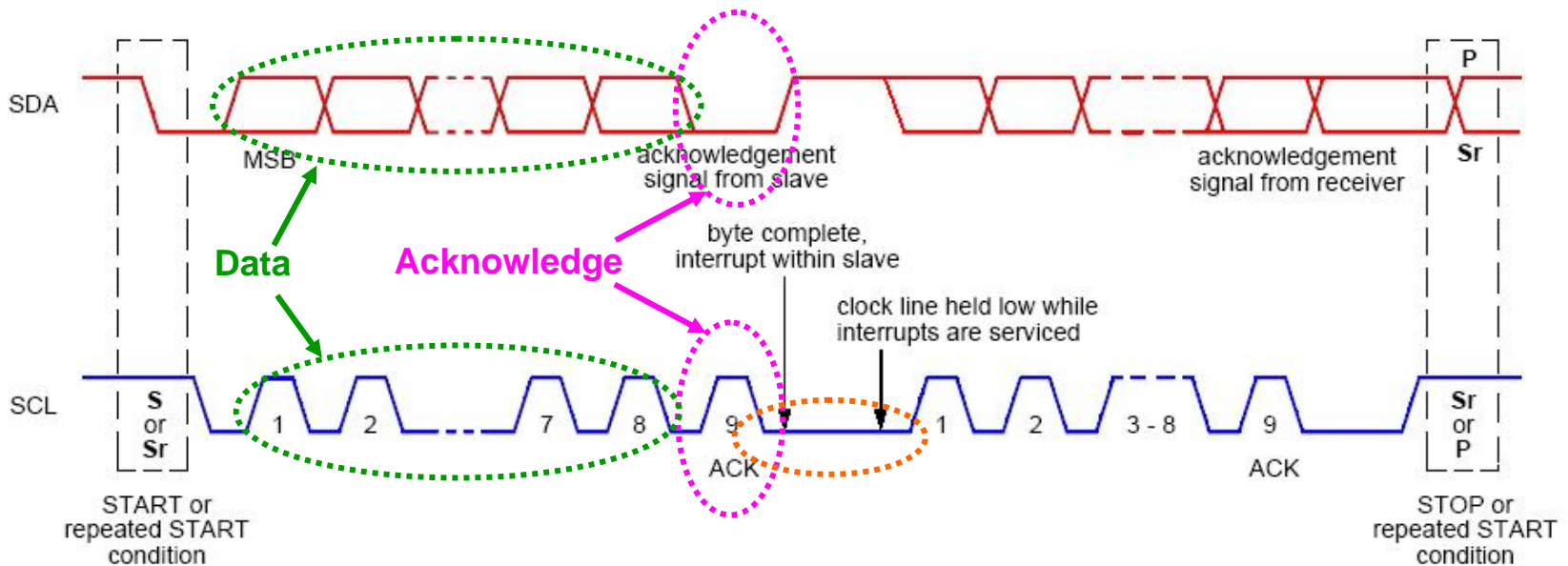
Start and Stop Conditions



Start Condition - a HIGH to LOW transition on the SDA line while SCL is HIGH

Stop Condition - a LOW to HIGH transition on the SDA line while SCL is HIGH

Data Transfer



- During data transfer, SDA must be stable when SCL is High
- Each byte has to be followed by an acknowledge bit
- Number of bytes transmitted per transfer is unrestricted
- If a slave can't receive or transmit another complete byte of data, it can hold the clock line SCL LOW to force the master into a wait state

Acknowledge and Not Acknowledge

The I²C specification says:

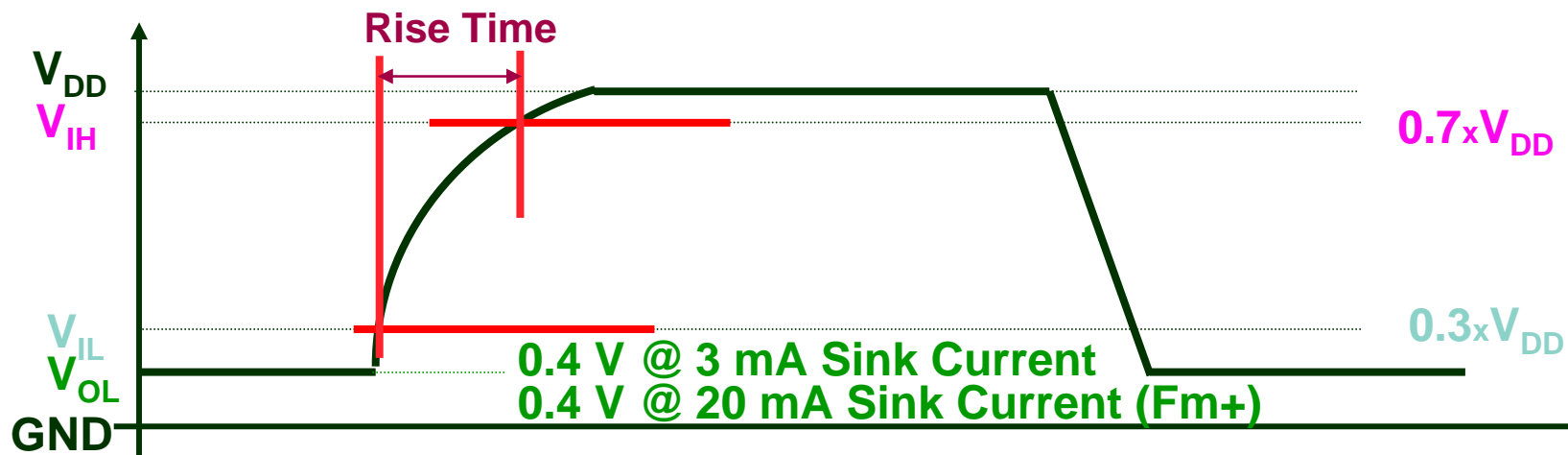
Data transfer with acknowledge is obligatory. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse.

However, there are a few scenarios where there is not an acknowledge (SDA staying HIGH). Which ones?

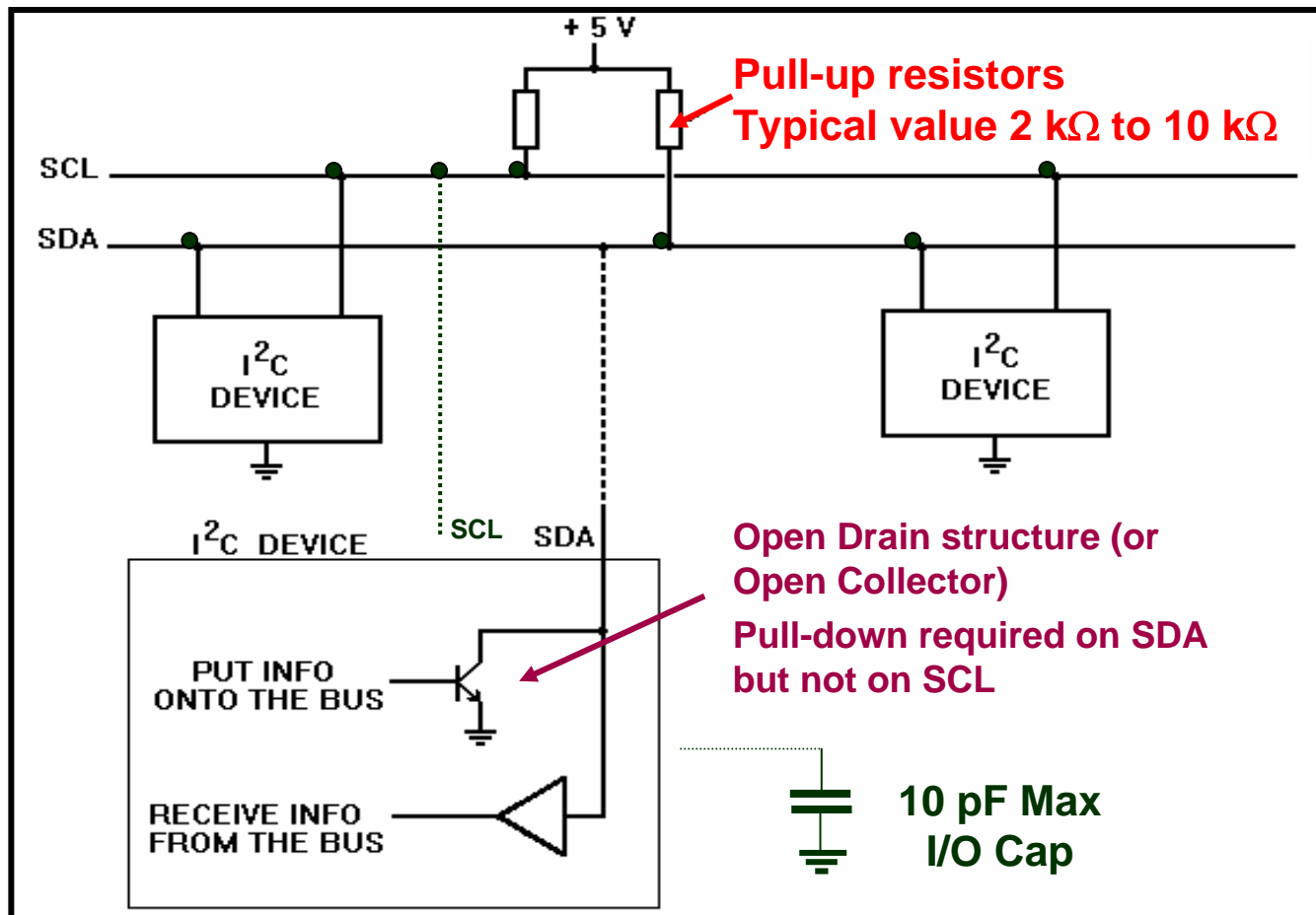
1. A receiver with the address is not present in the I²C bus
2. The receiver is performing real-time tasks and it cannot process the received I²C information
3. The receiver is the master and wants to take control of SDA line again in order to generate a STOP command. The slave transmitter MUST then release the SDA line when it sees the NACK so the master can send the STOP command.

Key numbers to keep in mind

	Standard Mode	Fast Mode	Fast Mode Plus	High Speed Mode	
Bit Rate (kb/s)	0 to 100	0 to 400	0 to 1000	0 to 1700	0 to 3400
Max Load (pF)	400	400	550	400	100
Rise time (ns)	1000	300	120	160	80
Noise filter (ns)	-	50	10	10	10

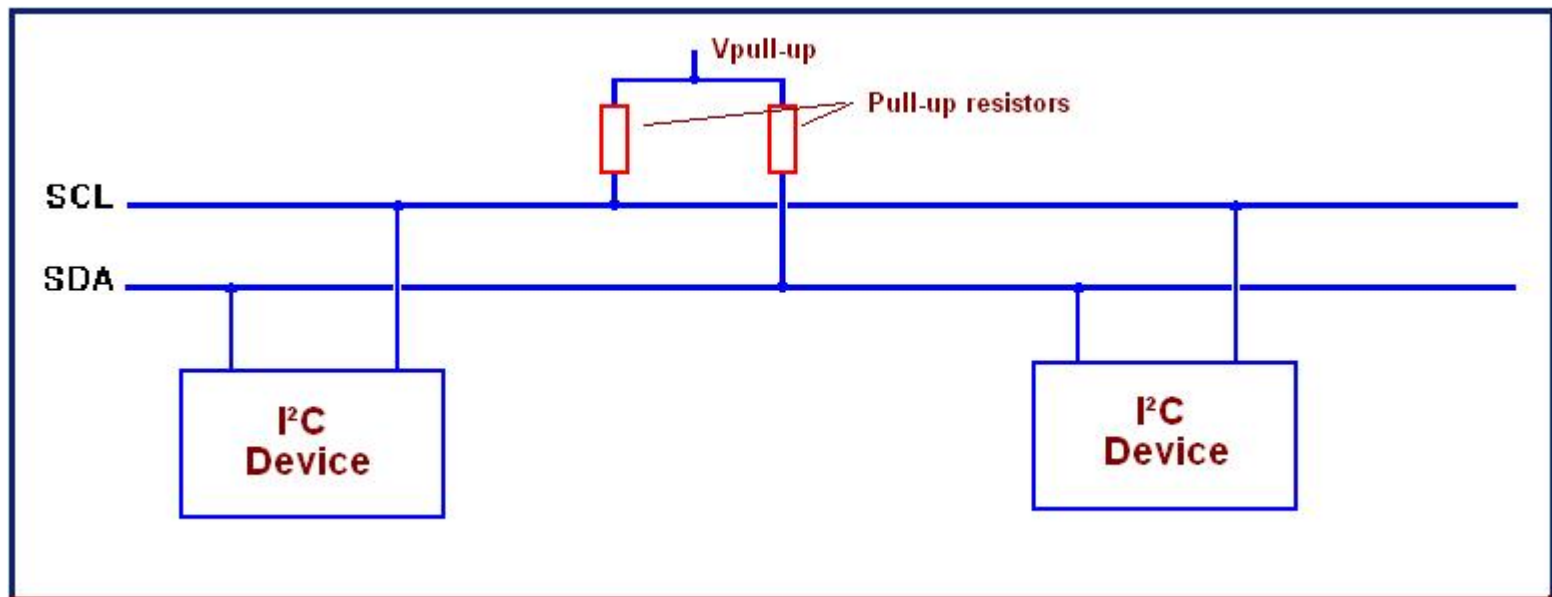


I²C Hardware architecture



I²C Pull-up Resistors

How do you calculate pull up resistor values ?



How to calculate I²C Pull-up Resistors?

Minimum value

There is a minimum resistor value determined by the I²C spec limit of 3 mA.

$$R = (V_{dd_{max}} - V_{ol_{max}}) / 0.003A$$

Example: using a 5±0.5 V bus: $R = (5.5V - 0.4V) / 0.003A = 1.7 \text{ k}\Omega$

Maximum value

Determined by the I²C-bus rise time requirements:

$$V(t_1) = 0.3 \cdot V_{dd} = V_{dd} (1 - 1/e^{t_1/RC}); \text{ then } t_1 = 0.3566749 \cdot RC$$

$$V(t_2) = 0.7 \cdot V_{dd} = V_{dd} (1 - 1/e^{t_2/RC}); \text{ then } t_2 = 1.2039729 \cdot RC$$

$$t = t_2 - t_1 = 0.8472979 \cdot RC$$

For standard-mode I²C-bus: $t = \text{rise time} = 1000\text{ns} (1 \mu\text{s})$

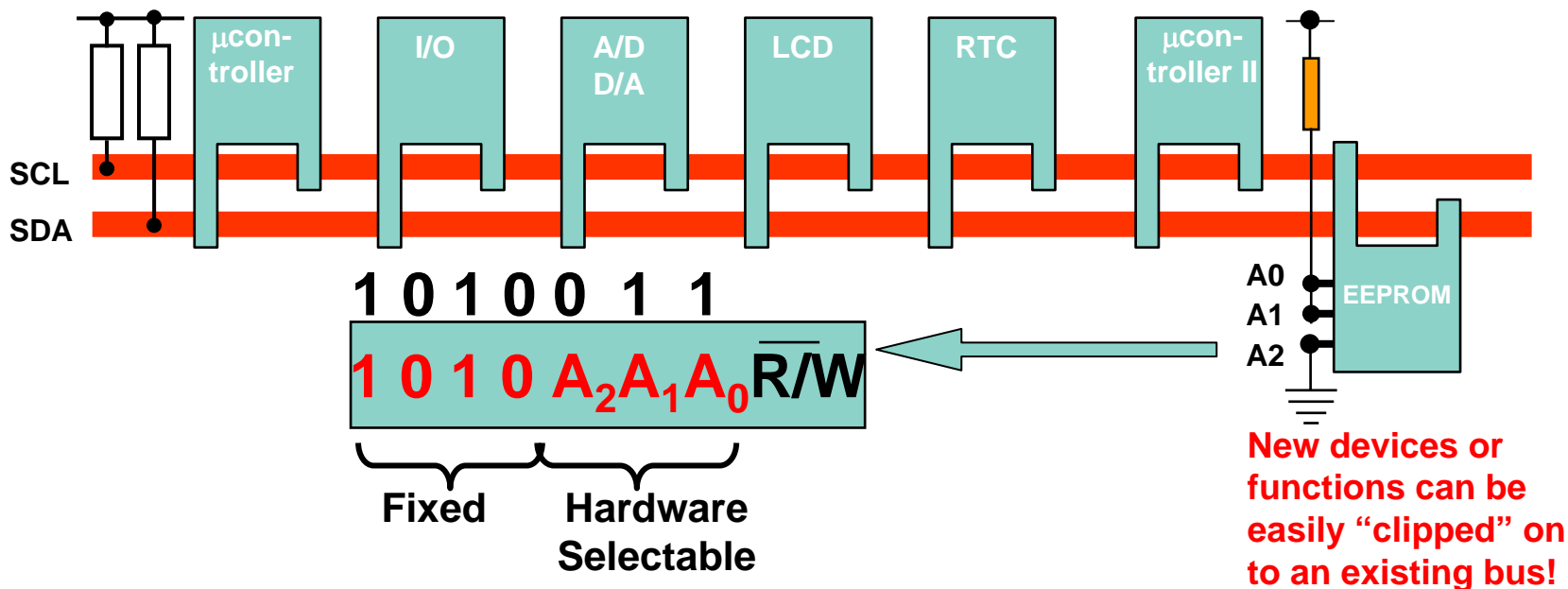
$$\text{so } RC = 1180.2 \text{ ns}$$

Example: at a bus load of 400 pF: $R_{max} = 2.95 \text{ k}\Omega$

For fast-mode:

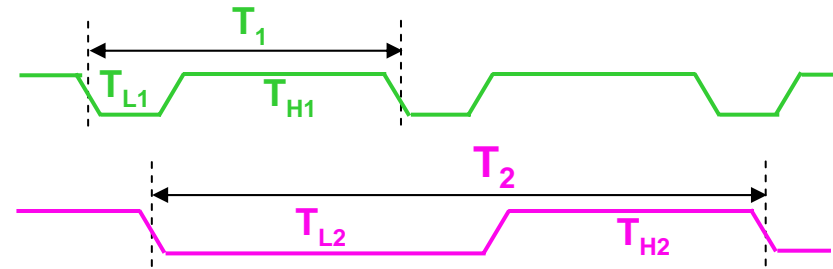
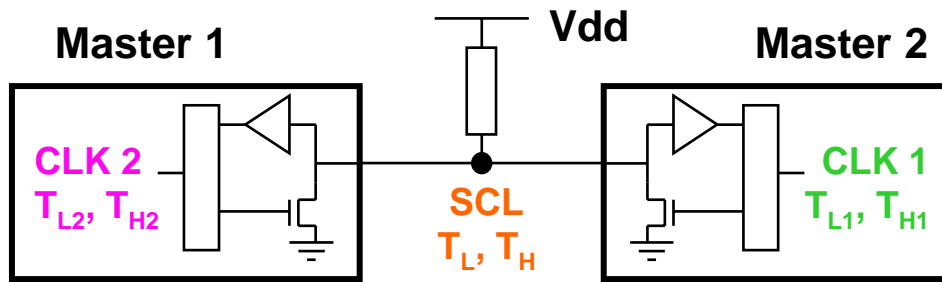
$$\text{I}^2\text{C-bus rise time} = 300 \text{ ns @ } 400 \text{ pF: } R_{max} = 885 \Omega$$

I²C Address, Basics

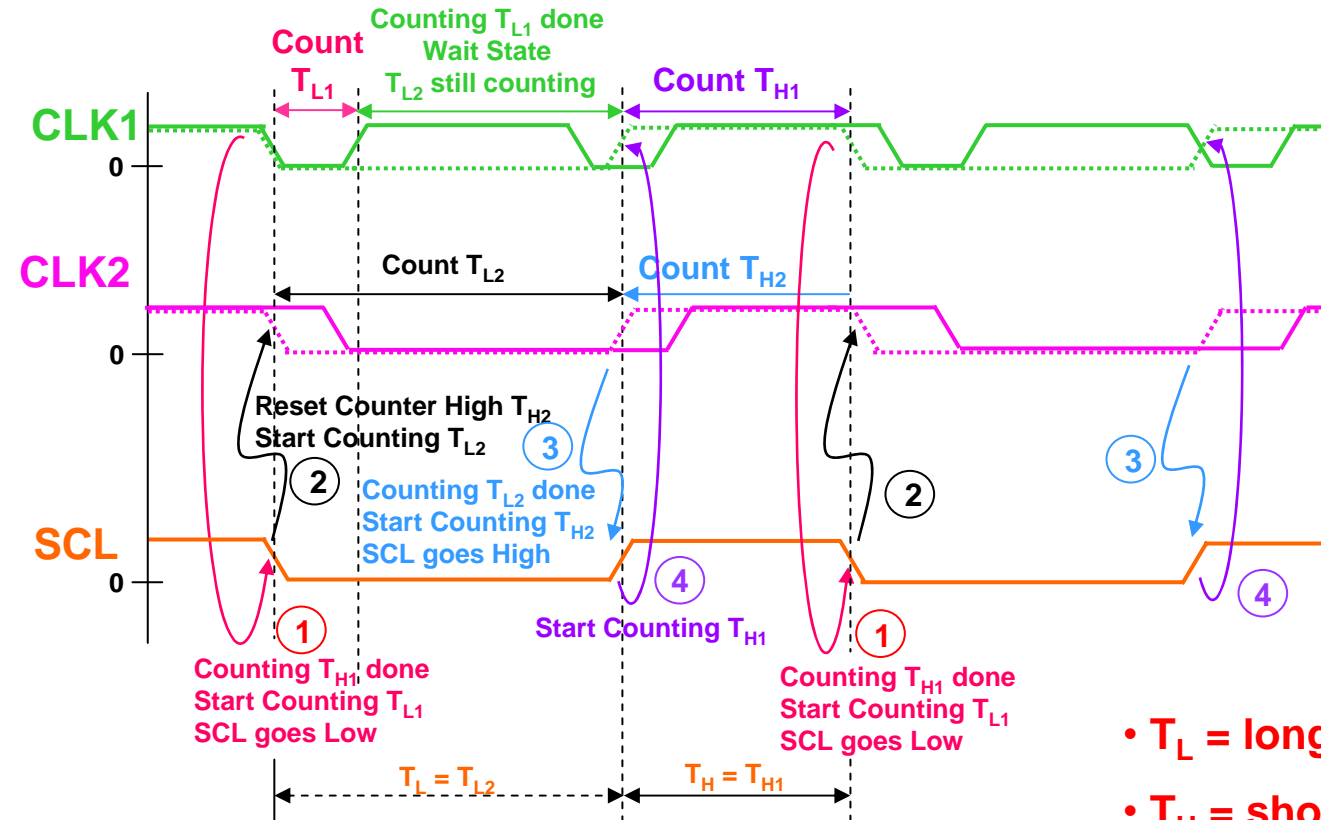


- Each device is addressed individually by software
- Unique address per device: fully fixed or with a programmable part through hardware pin(s)
- Programmable pins mean that several of the same devices can share the same bus
- Address allocation coordinated by the I²C-bus committee
- 112 different addresses max with the 7-bit format (others reserved)
- 10-bit format use a 2 byte message: 1111 0A₉A₈R/W + A₇A₆A₅A₄A₃A₂A₁A₀

Multi Master Mode - Clock Synchronization



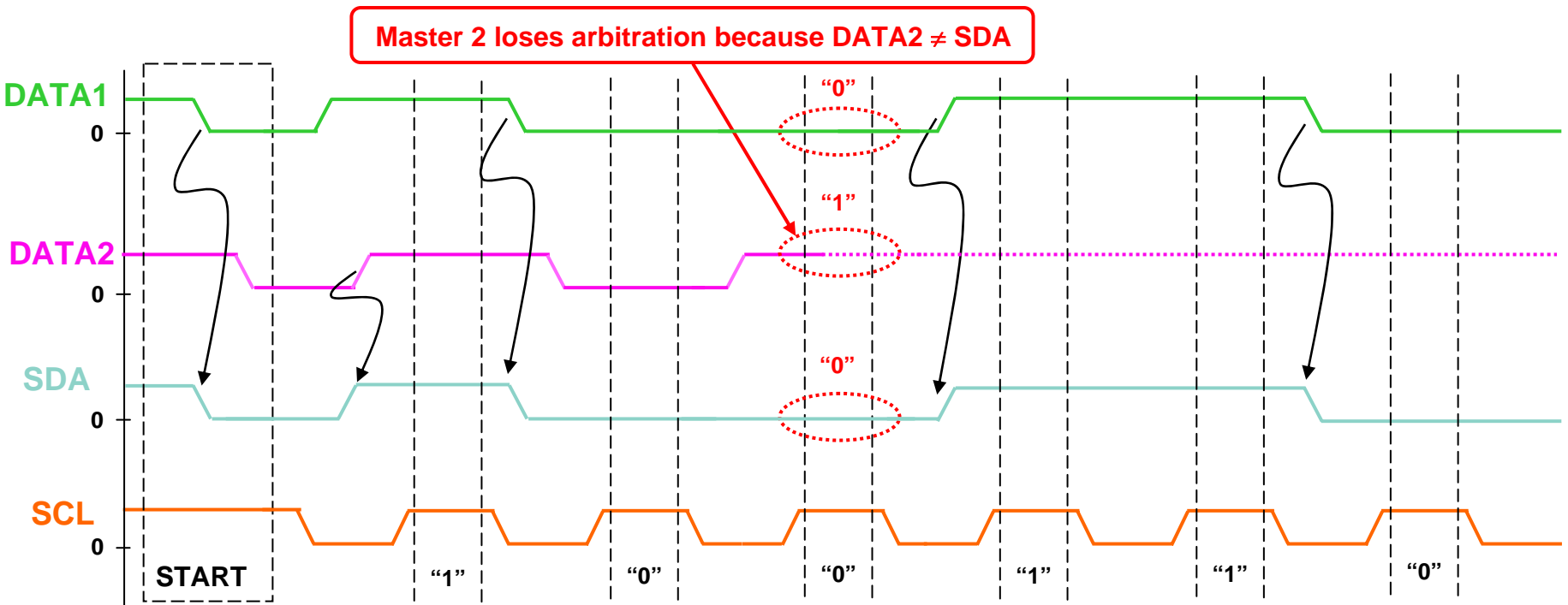
Internal Counters count the Low and High times (T_{L1}, T_{H1}) and (T_{L2}, T_{H2})



- $T_L = \text{longest } T_L = \max(T_{L1}, T_{L2}, T_{Ln})$
- $T_H = \text{shortest } T_H = \min(T_{H1}, T_{H2}, T_{Hn})$

Multi Master Mode - Arbitration

- Two or more masters may generate a START condition at the same time
- Arbitration is done on SDA while SCL is HIGH - Slaves are not involved



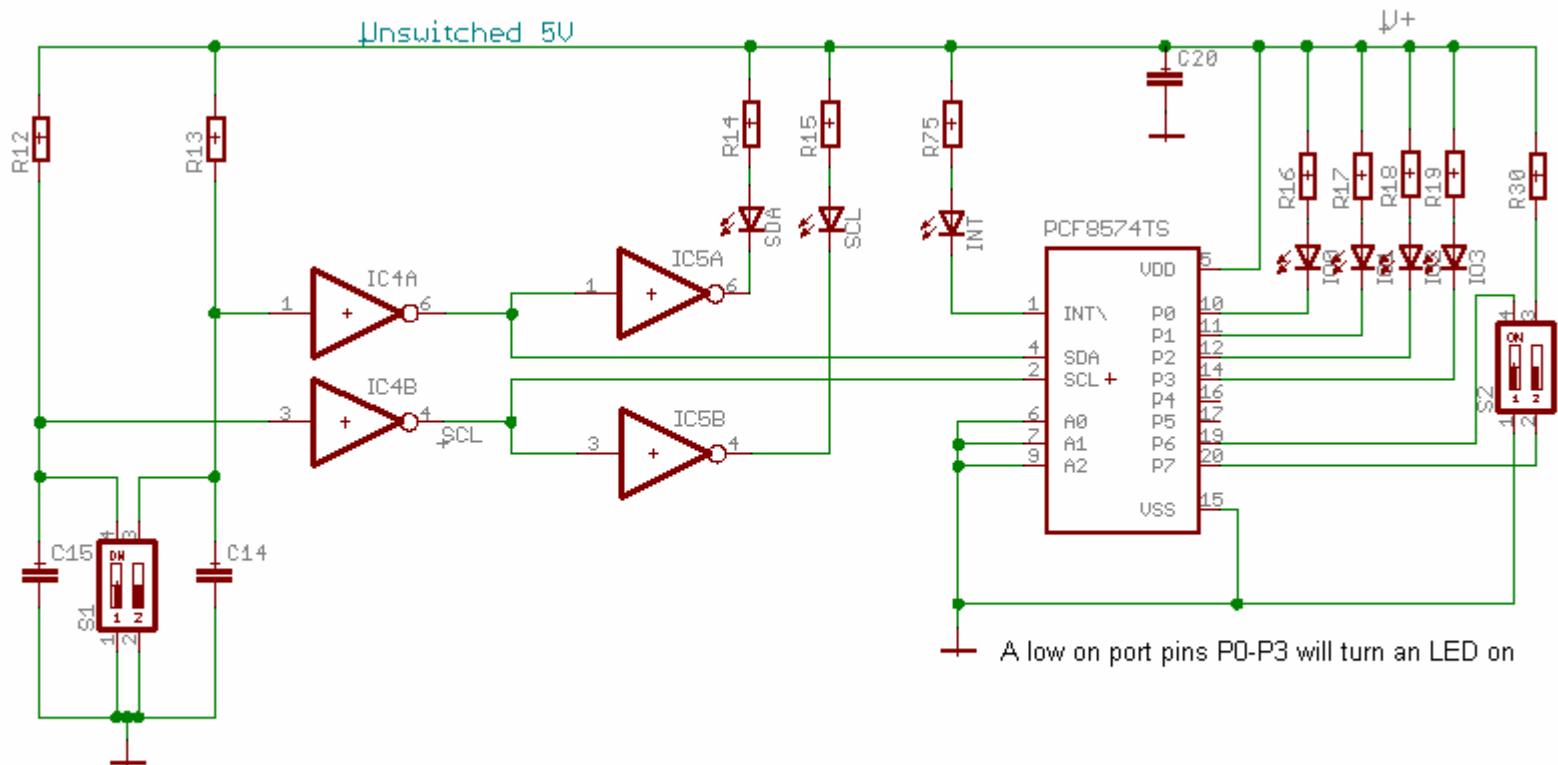
SUMMARY: the master that sends a "1" while the other sends a "0" loses the arbitration

Test: I²C Protocol Summary

START	<u>HIGH</u> to <u>LOW</u> transition on SDA while SCL is HIGH
STOP	<u>LOW</u> to <u>HIGH</u> transition on SDA while SCL is HIGH
DATA	8-bit word, MSB first (Address, Control, Data) <ul style="list-style-type: none">- must be stable when SCL is <u>HIGH</u>- can change only when SCL is <u>LOW</u>- number of bytes transmitted is <u>unrestricted</u>
ACKNOWLEDGE	<ul style="list-style-type: none">- done on each <u>9</u> th clock pulse during the <u>HIGH</u> period- the transmitter releases the bus - SDA goes <u>HIGH</u>- the receiver pulls DOWN the bus line - SDA goes <u>LOW</u>
CLOCK	<ul style="list-style-type: none">- Generated by the <u>master</u>(s)- Maximum speed: (<u>100</u>, <u>400</u>, <u>1000</u>, <u>3400</u> kHz) but NO min.- A receiver can hold SCL <u>LOW</u> when performing another function (transmitter in a Wait state)- A master can <u>slow down</u> the clock for slow devices
ARBITRATION	<ul style="list-style-type: none">- Master can start a transfer only if the bus is <u>free</u>- Several masters can start a transfer <u>at the same time</u>- Arbitration is done on <u>SDA</u> line- Master that <u>lost</u> the arbitration must stop sending data

Exercise

Use switch S1 to generate an I²C message to the PCF8574



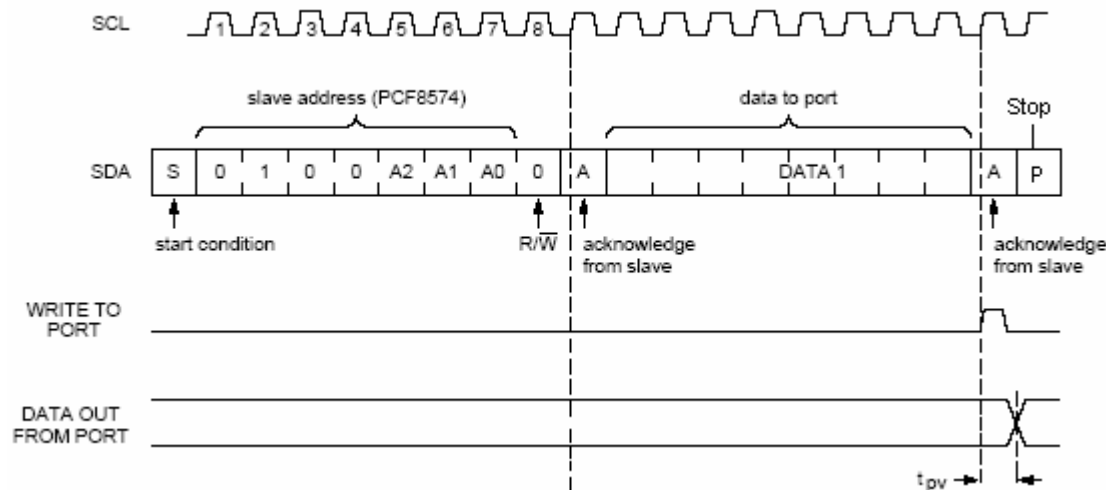
Low - switch open. Bus low. LED off.
High - switch closed. Bus high. LED on.

Exercise

Objective: verify understanding of the I²C bus protocol

Use switch S1 to generate an I²C message to the PCF8574

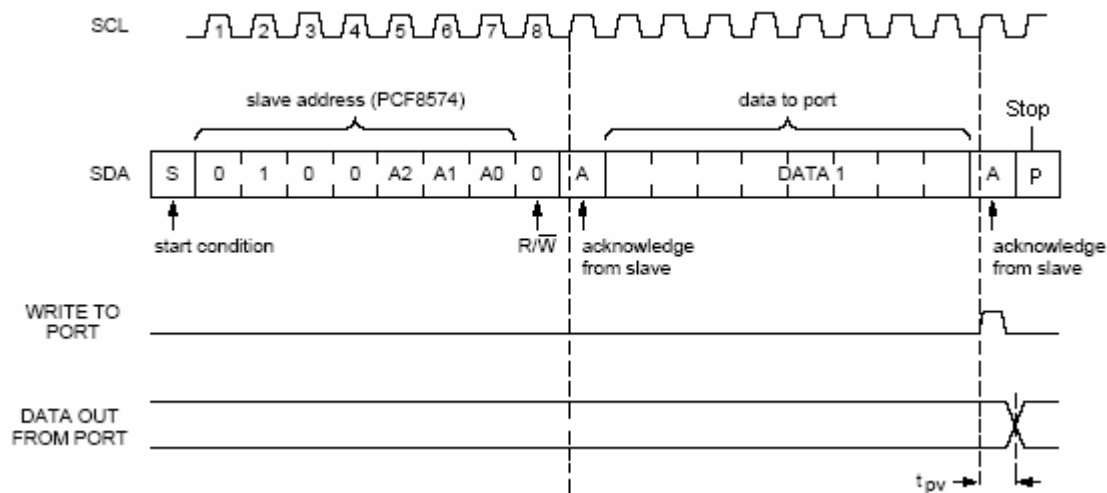
- The PCF8574 address is 0x40
- The PCF8574 requires one data byte
- The data byte should be 0xFA to turn on two of the LEDs



Exercise

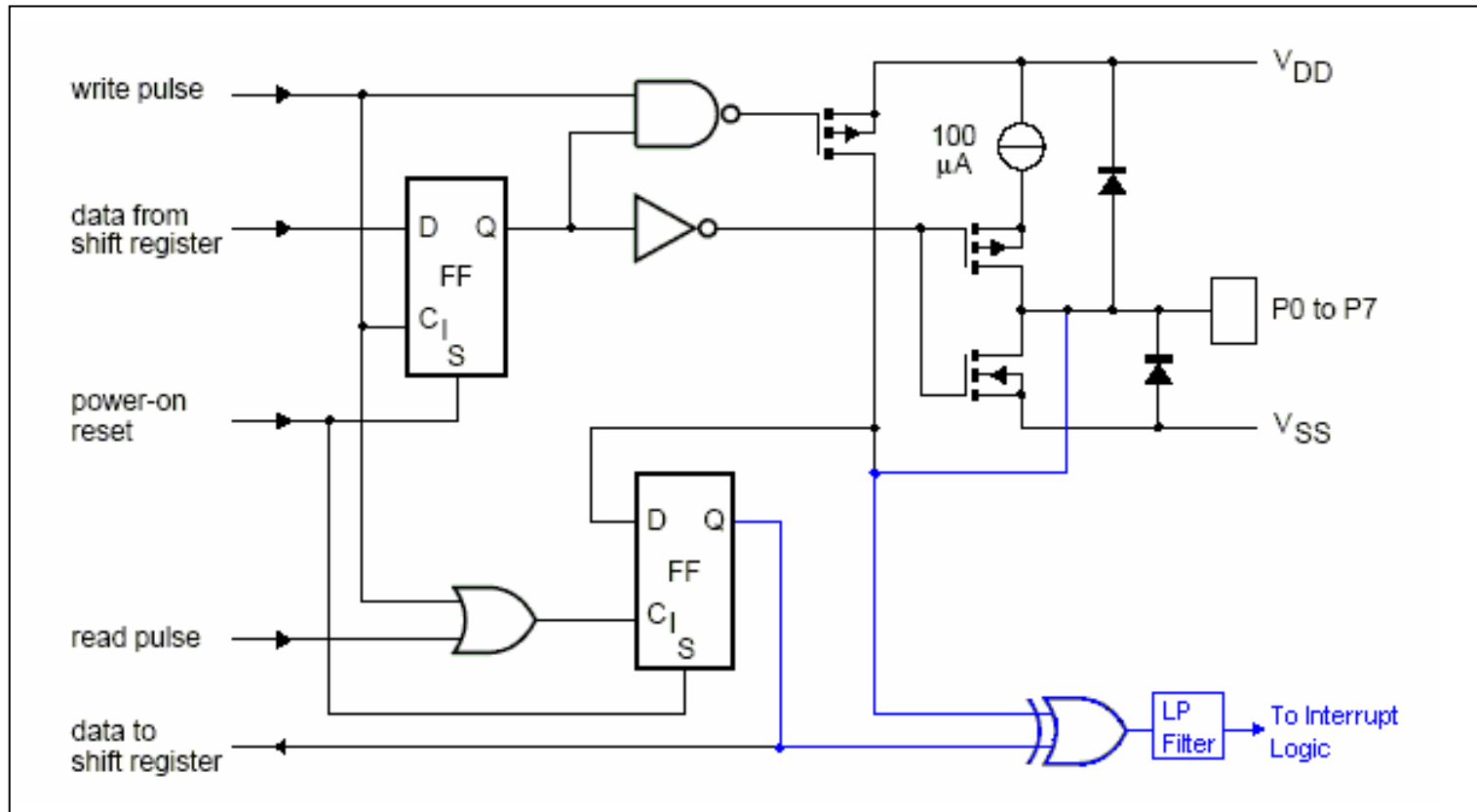
The following sequence should be followed:

1. Send Start condition
2. Send PCF8574 address with LSB='0' for Write
3. Send data byte 0xFA
4. Send Stop condition
5. Don't forget the acknowledge clock pulses



Quasi Bi-directional I/O Port Structure

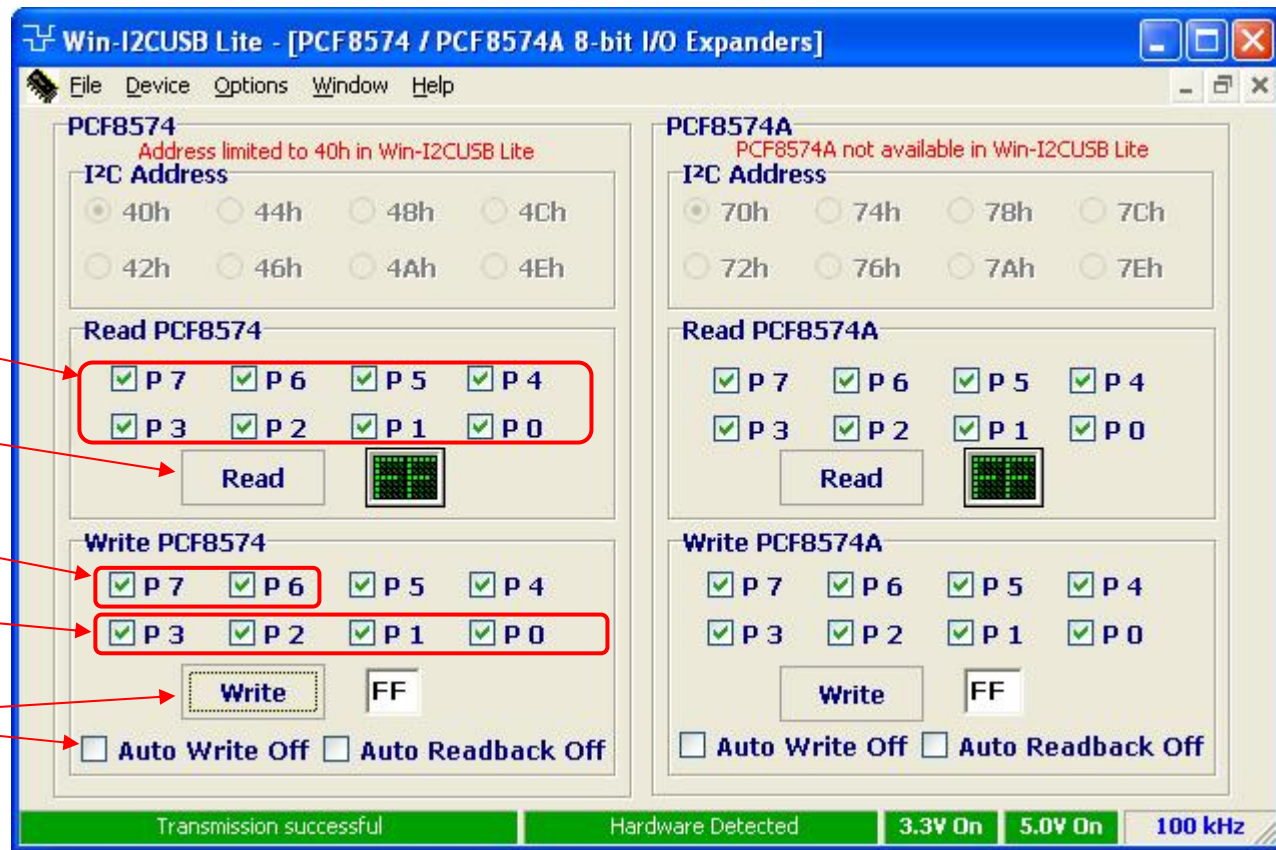
Quasi Bi-directional I/O Port Structure



Exercise

Objective: to understand the operation of the quasi bi-directional port structure

We will use the PCF8574 interface found in Win-I2CUSB Lite



Exercise 2

1. Set all the port pins high using Win-I2CUSB Lite
2. Change the state of switch 1 on S2.
3. The INT LED should illuminate
4. Change the state of the switch. The LED should turn off.
5. Turn the INT LED again by changing the switch.
6. Using Win-I2CUSB Lite. Read the PCF8574. What happened to the Interrupt?
7. Turn the INT LED again by changing the switch. Write to the PCF8574. What happened to the Interrupt?

