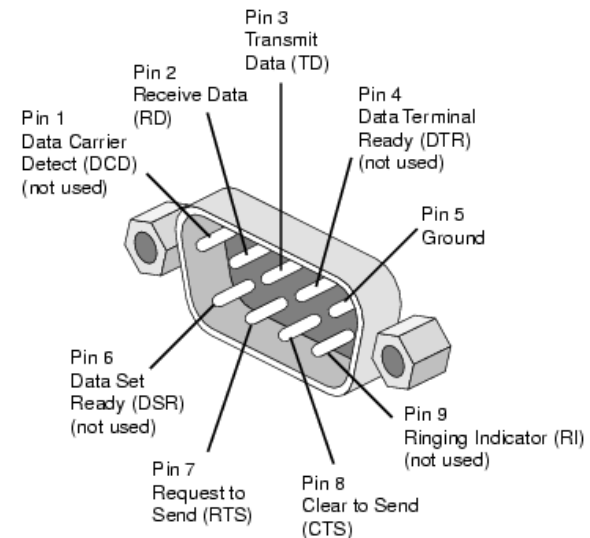


# EECS 373

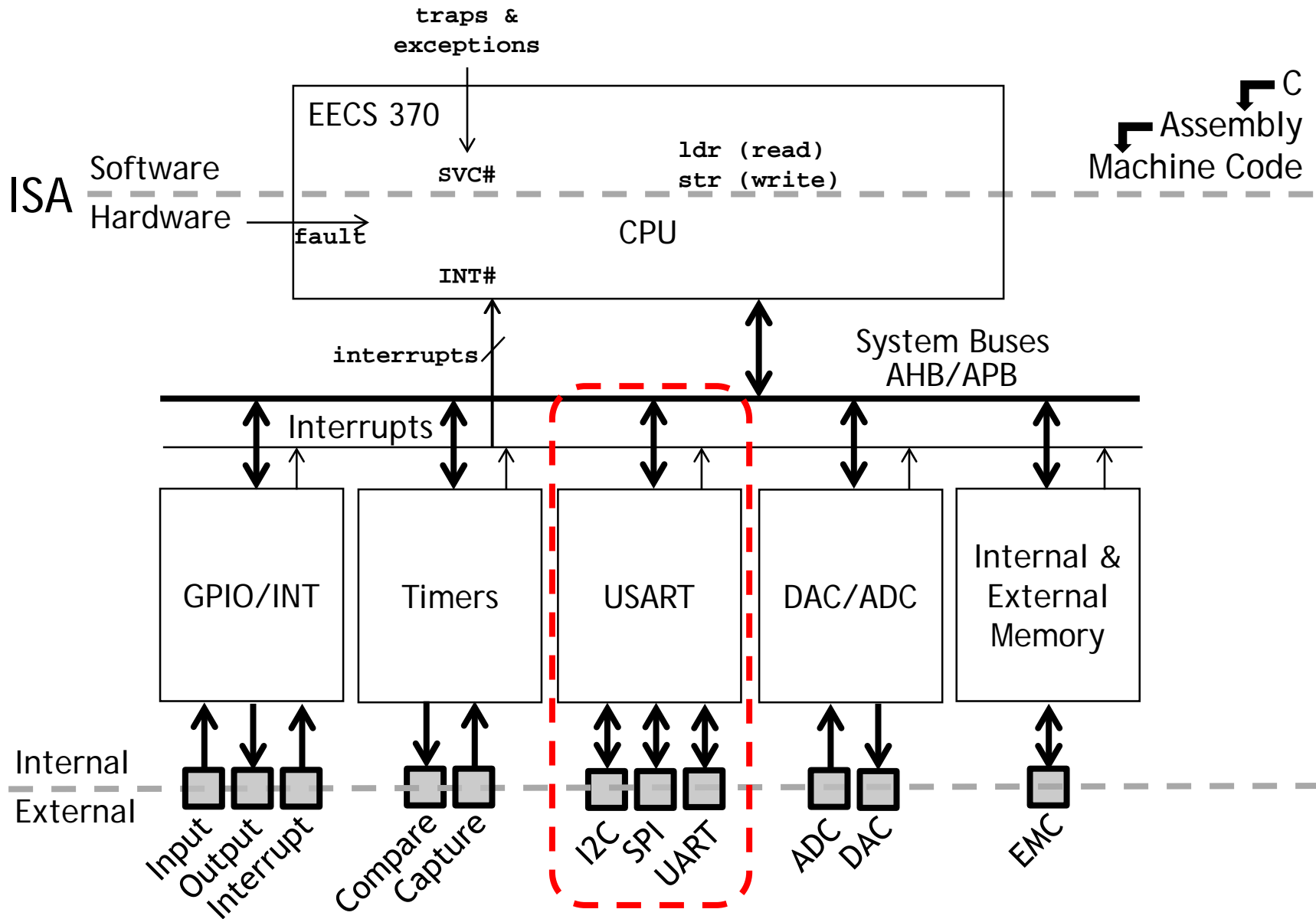
## Design of Microprocessor-Based Systems

Prabal Dutta  
University of Michigan

### Serial buses



# This Week: Serial interfaces



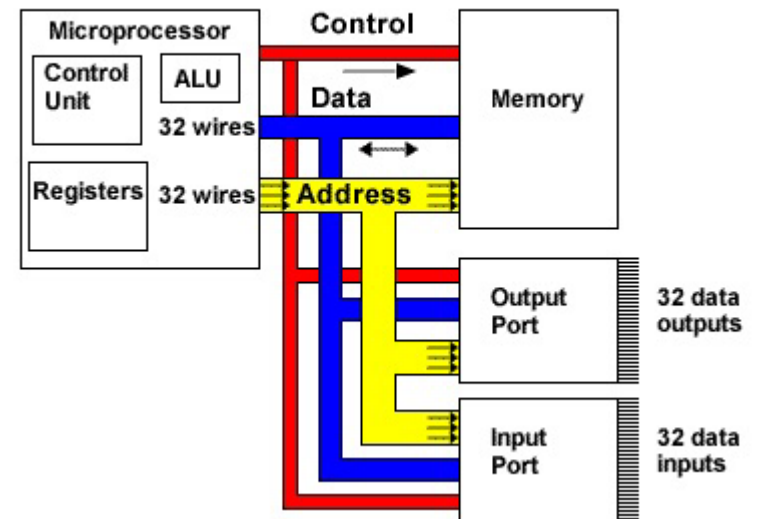
- Introduction to Serial Buses
- UART
- SPI
- I2C

- Motivation
  - Connect different systems together
    - Two embedded systems
    - A desktop and an embedded system
  - Connect different chips together in the same embedded system
    - MCU to peripheral
    - MCU to MCU
  - Without using a lot of I/O lines
    - I/O lines require I/O pads which cost \$\$\$ and size
    - I/O lines require PCB area which costs \$\$\$ and size
  - Often at relatively low data rates
  - But sometimes at higher data rates
- So, what are our options?
  - Universal Synchronous/Asynchronous Receiver Transmitter
  - Also known as USART (pronounced: "you-zart")

- Number of wires required?
- Asynchronous or synchronous?
- How fast can it transfer data?
- Can it support more than two endpoints?
- Can it support more than one master (i.e. txn initiator)?
- How do we support flow control?
- How does it handle errors/noise?



- A multidrop bus (MDB) is a computer bus in which all components are connected to the same set of electrical wires. (from Wikipedia)
  - In the general case, a bus may have more than one device capable of driving it.
    - That is, it may be a “multi-master” bus as discussed earlier.

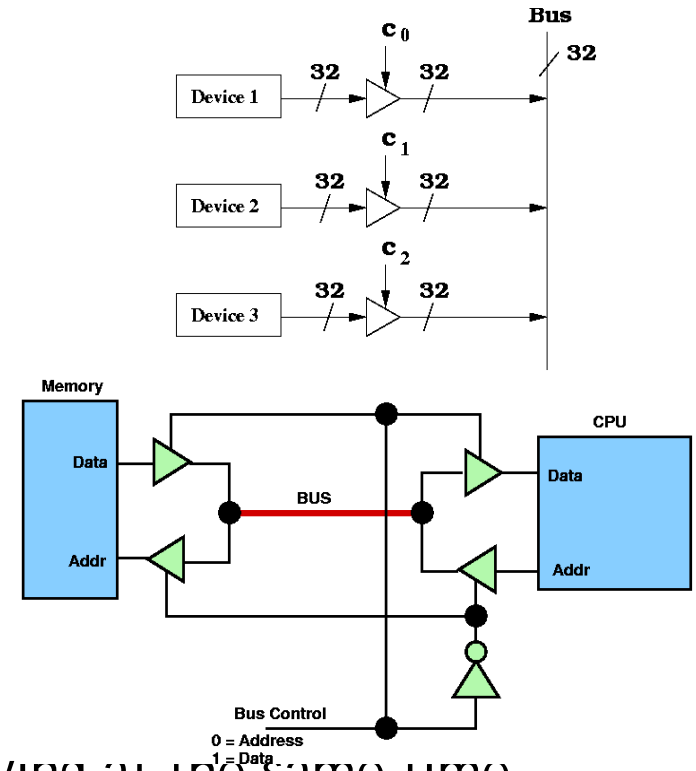




# How can we handle multiple (potential) bus drivers? (1/3)



- Tri-state devices, just have one device drive at a time. Everyone can read though
  - Pros:
    - Very common, fairly fast, pin-efficient.
  - Cons:
    - Tri-state devices can be slow.
      - Especially drive-to-tristate?
    - Need to be sure two folks not driving at the same time
      - Let out the magic smoke.
  - Most common solution (at least historically)
    - Ethernet, PCI, etc.



# How can we handle multiple (potential) bus drivers? (2/3)



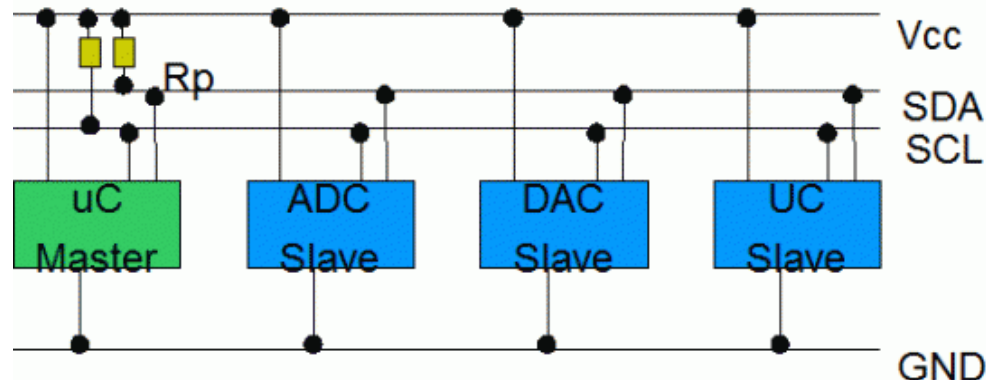
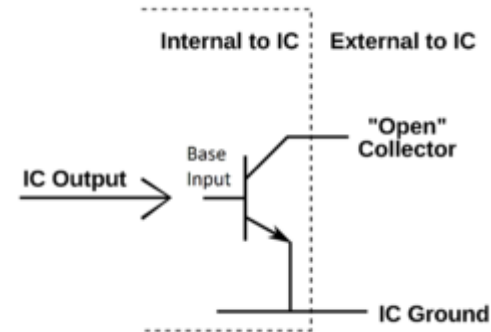
- MUX
  - Just have each device generate its data, and have a MUX select.
    - That's a LOT of pins.
      - Consider a 32-bit bus with 6 potential drivers.
        - » Draw the figure.
        - » How many pins needed for the MUX?
  - Not generally realistic for an “on-PCB” design as we’ll need an extra device (or a lot of pins on one device)
    - But reasonable on-chip
      - In fact AHB, APB do this.



# How can we handle multiple (potential) bus drivers? (3/3)



- “pull-up” aka “open collector” aka “wired OR”
  - Wire is pulled high by a resistor
  - If any device pulls the wire low, it goes low.
- Pros:
  - If two devices both drive the bus, it still works!
- Cons:
  - Rise-time is very slow.
  - Constant power drain.
- Used in I2C, CAN



- Introduction to Serial Buses
- UART
- SPI
- I2C

- Universal Asynchronous Receiver/Transmitter
- Hardware that translates between parallel and serial forms
- Commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485
- The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) typically are handled by a special driver circuit external to the UART.

- Each character is sent as
  - a logic *low* start bit
  - a configurable number of data bits (usually 7 or 8, sometimes 5)
  - an optional parity bit
  - *one or more logic high* stop bits
  - with a particular bit timing ("baud")
- Examples
  - "9600-N-8-1" → <baudrate><parity><databits><stopbits>
  - "9600-8-N-1" → <baudrate><databits><parity><stopbits>



- UART is actually a generic term that includes a large number of different devices/standards.
  - RS-232 is a standard that specifies
    - “electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin out of connectors.

# Signals (only most common)

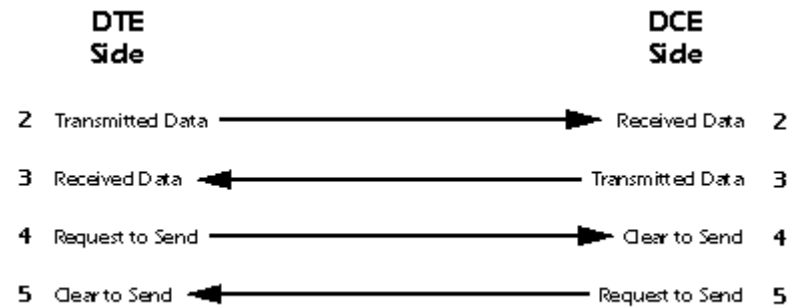
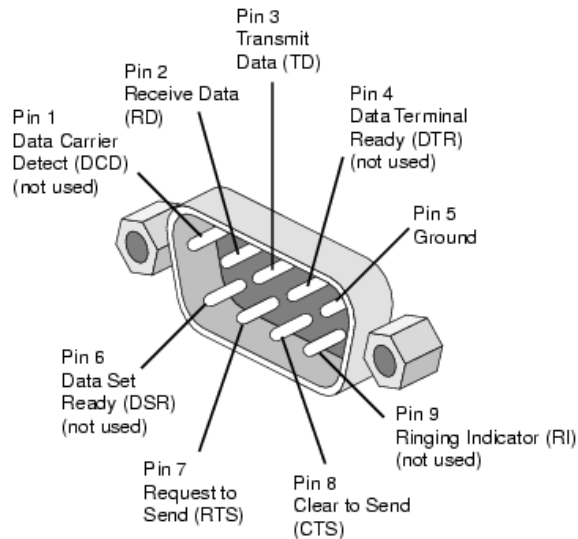


- The RXD signal of a UART is the signal receiving the data. This will be an input and is usually connected to the TXD line of the downstream device.
- The TXD signal of a UART is the signal transmitting the data. This will be an output and is usually connected to the RXD line of the downstream device.
- The RTS# (Ready to Send) signal of a UART is used to indicate to the downstream device that the device is ready to receive data. This will be an output and is usually connected to the CTS# line of the downstream device.
- The CTS# (Clear to Send) signal of a UART is used by the downstream device to identify that it is OK to transmit data to the upstream device. This will be an input and is usually connected to the RTS# line of the upstream device.

# DB9 stuff



- DTE vs DCE
- Pinout of a DCE?
- Common ground?
- Noise effects?



Pin Number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring Indicator

Wiring a DTE device to a DCE device for communication is easy.

The pins are a one-to-one connection, meaning all wires go from pin x to pin x.

A straight through cable is commonly used for this application.

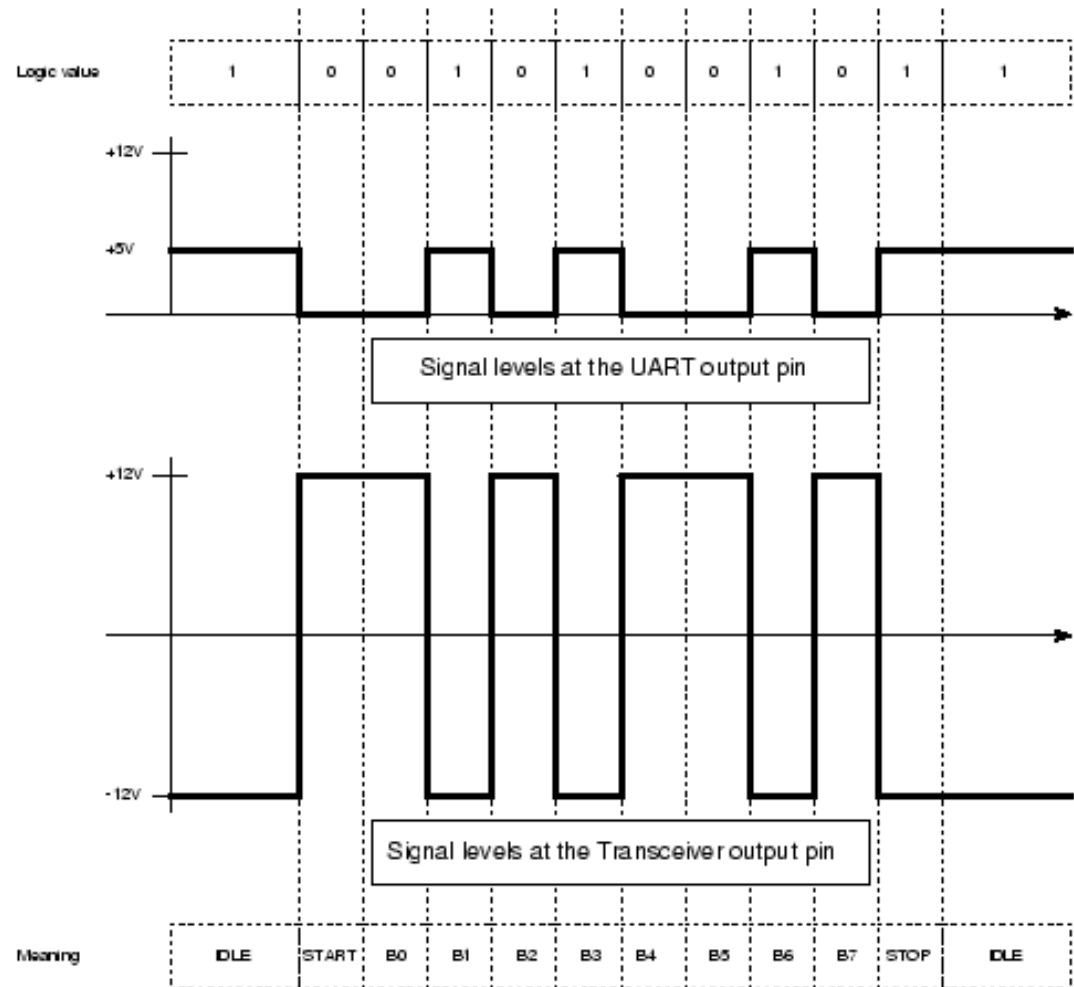
In contrast, wiring two DTE devices together requires crossing the transmit and receive wires.

This cable is known as a null modem or crossover cable.

# RS-232 transmission example



RS232 Transmission of the letter 'J'



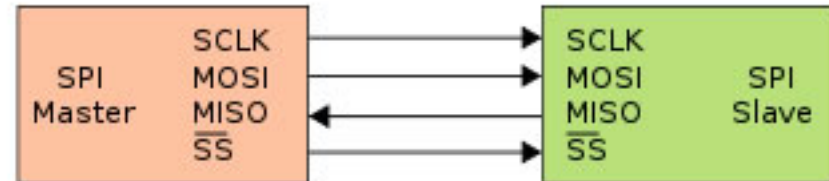


- How fast can we run a UART?
- What are the limitations?
- Why do we need start/stop bits?
- How many data bits can be sent?
  - 9600-8-N-1 is ok. Is 9600-8192-N-1 ok too?

- Introduction to Serial Buses
- UART
- SPI
- I2C

# Introduction

- What is it?
- Basic Serial Peripheral Interface (SPI)
- Capabilities
- Protocol
- Pro / Cons and Competitor
- Uses
- Conclusion

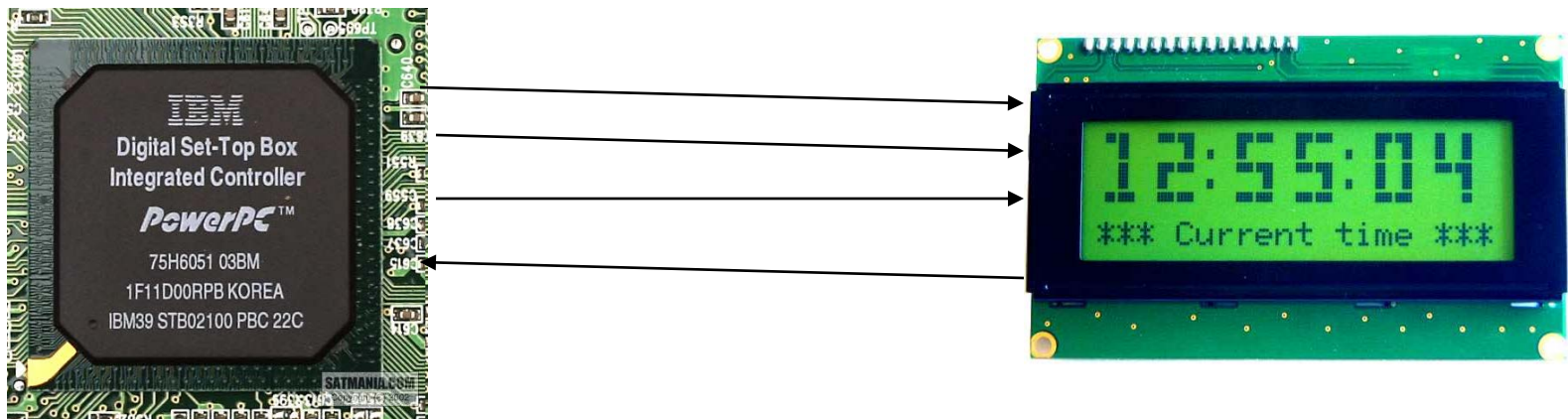


Serial Peripheral Interface

[http://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/SPI\\_single\\_slave.svg/350px-SPI\\_single\\_slave.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/SPI_single_slave.svg/350px-SPI_single_slave.svg.png)

# What is SPI?

- Serial Bus protocol
- Fast, Easy to use, Simple
- Everyone supports it



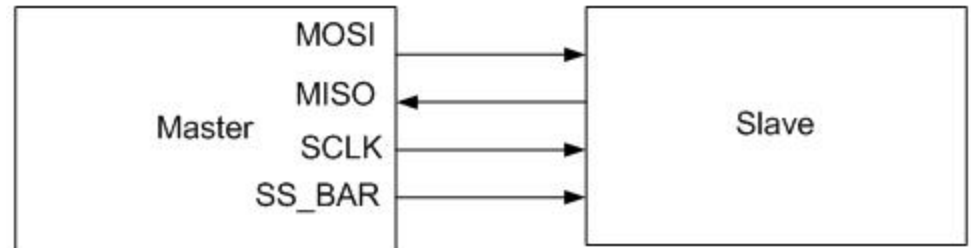
# SPI Basics

- A communication protocol using 4 wires
  - Also known as a 4 wire bus
- Used to communicate across small distances
- Multiple Slaves, Single Master
- Synchronized

# Capabilities of SPI

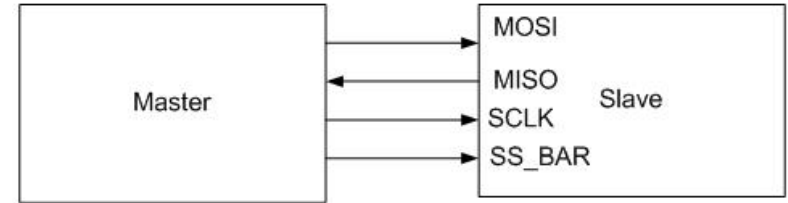
- Always Full Duplex
  - Communicating in two directions at the same time
  - Transmission need not be meaningful
- Multiple Mbps transmission speed
- Transfers data in 4 to 16 bit characters
- Multiple slaves
  - Daisy-chaining possible

# Protocol



- Wires:
  - Master Out Slave In (MOSI)
  - Master In Slave Out (MISO)
  - System Clock (SCLK)
  - Slave Select 1...N
- Master Set Slave Select low
- Master Generates Clock
- Shift registers shift in and out data

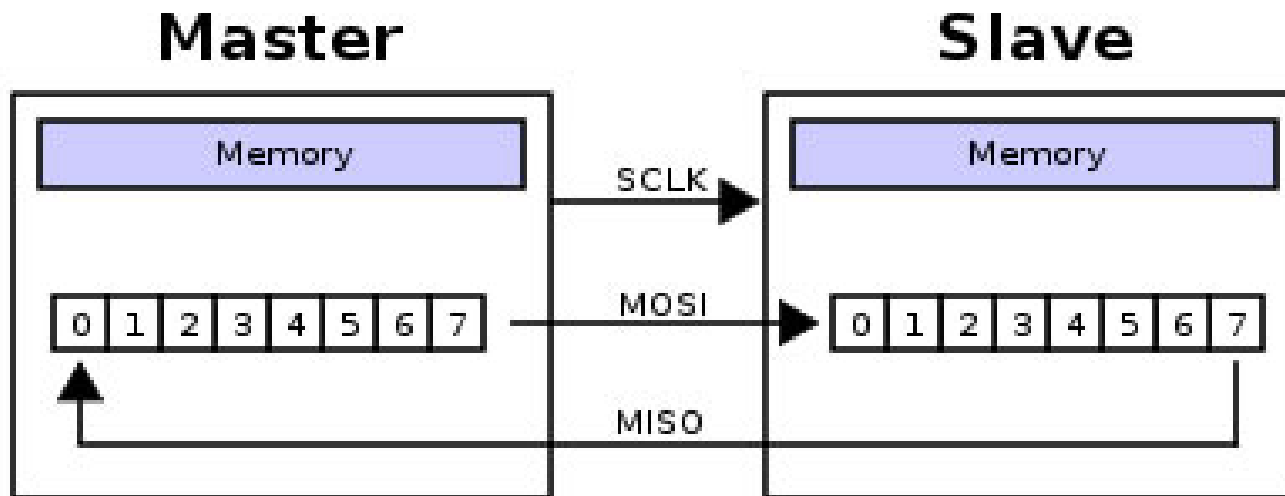
# Wires in Detail



- MOSI – Carries data out of Master to Slave
- MISO – Carries data from Slave to Master
  - Both signals happen for every transmission
- SS\_BAR – Unique line to select a slave
- SCLK – Master produced clock to synchronize data transfer



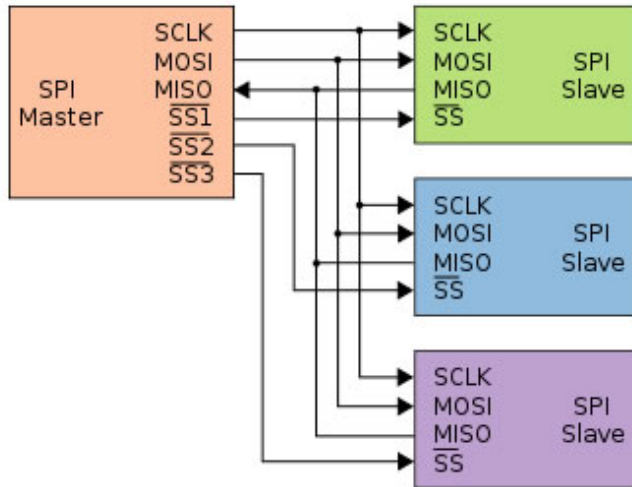
# Shifting Protocol



Master shifts out data to Slave, and shift in data from Slave

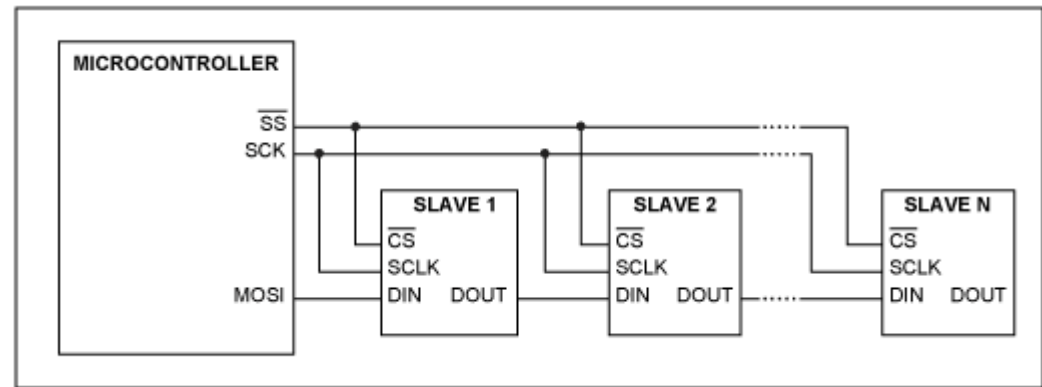
[http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/SPI\\_8-bit\\_circular\\_transfer.svg/400px-SPI\\_8-bit\\_circular\\_transfer.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/SPI_8-bit_circular_transfer.svg/400px-SPI_8-bit_circular_transfer.svg.png)

# Diagram



## Master and multiple independent slaves

[http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI\\_three\\_slaves.svg/350px-SPI\\_three\\_slaves.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/SPI_three_slaves.svg/350px-SPI_three_slaves.svg.png)



Some wires have been renamed

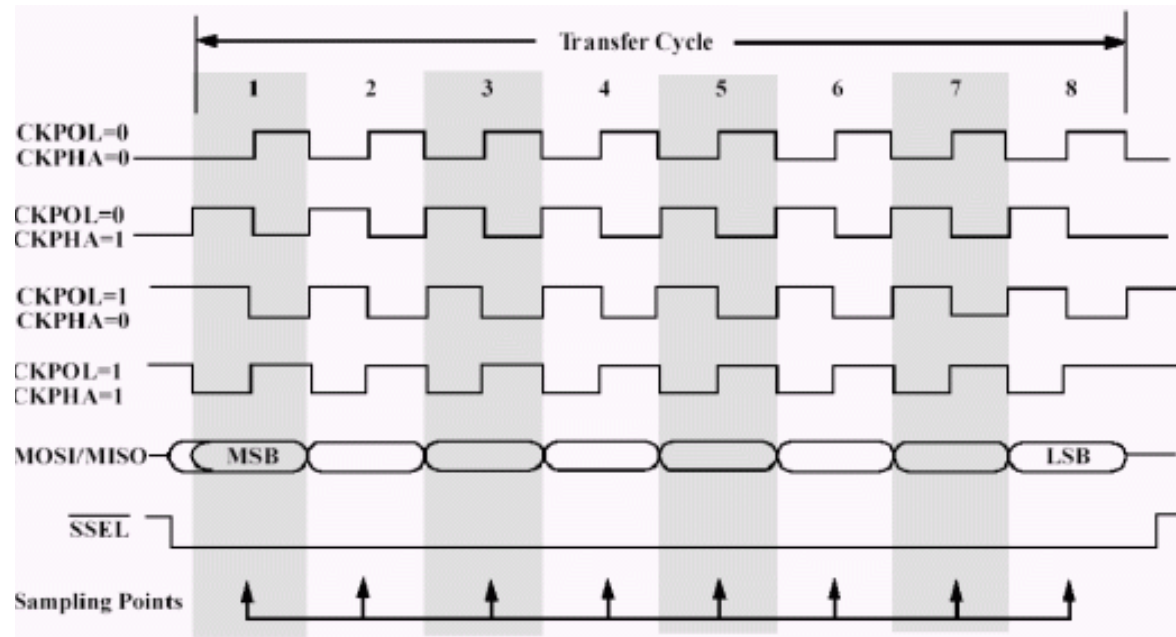
## Master and multiple daisy-chained slaves

[http://www.maxim-ic.com/appnotes.cfm/an\\_pk/3947](http://www.maxim-ic.com/appnotes.cfm/an_pk/3947)

# Clock Phase (Advanced)

- Two phases and two polarities of clock
- Four modes
- Master and selected slave must be in same mode
- Master must change polarity and phase to communicate with slaves of different numbers

# Timing Diagram



Timing Diagram – Showing Clock polarities and phases

<http://www.maxim-ic.com.cn/images/appnotes/3078/3078Fig02.gif>

# Pros and Cons

## Pros:

- Fast and easy
  - Fast for point-to-point connections
  - Easily allows streaming/Constant data inflow
  - No addressing/Simple to implement
- Everyone supports it

## Cons:

- SS makes multiple slaves very complicated
- No acknowledgement ability
- No inherent arbitration
- No flow control

# Uses

- Some Serial Encoders/Decoders, Converters, Serial LCDs, Sensors, etc.
- Pre-SPI serial devices

# Summary

- SPI – 4 wire serial bus protocol
  - MOSI MISO SS SCLK wires
- Full duplex
- Multiple slaves, One master
- Best for point-to-point streaming data
- Easily Supported

- Introduction to Serial Buses
- UART
- SPI
- I2C



# What is I<sup>2</sup>C (or I2C)?



- Inter-Integrated Circuit
- Pronounced “eye-squared-see”
- Two-wire serial bus protocol
- Invented by Philips in the early 1980's
  - That division now spun-off into NXP

- Originally used by Philips inside television sets
- Now very common in peripheral devices intended for embedded systems use
  - Philips, National Semiconductor, Xicor, and Siemens , ...
- Also used in the PC world
  - Real time clock
  - Temperature sensors

- Two-wire serial protocol with addressing capability
- Speeds up to 3.4 Mbit/s
- Multi-master/Multi-slave

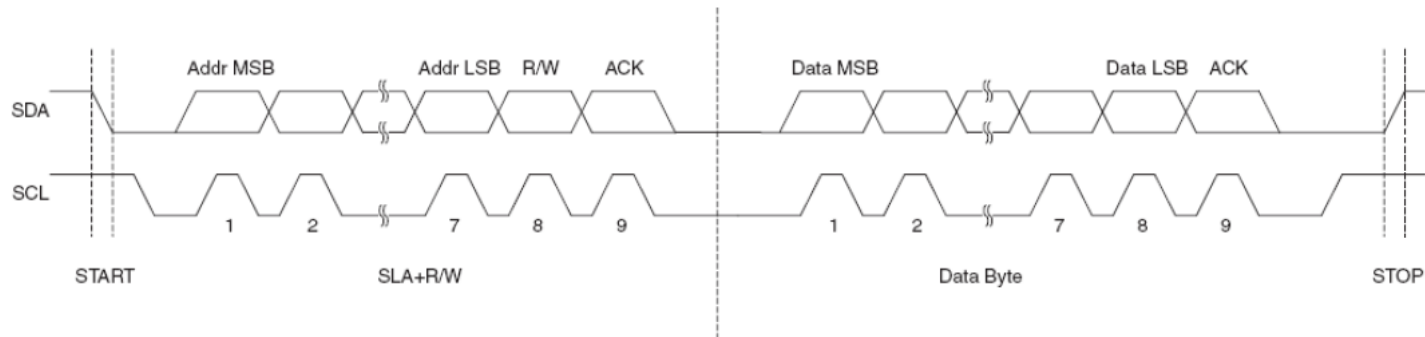
- Two lines
  - SDA (data)
  - SCL (clock)
- Open-collector
  - Very simple interfacing between different voltage levels

- Not a traditional clock
- Normally high (kept high by the pull-up)
- Pulsed by the master during data transmission (whether the master is transmitter or receiver)
- Slave device can hold clock low if it needs more time

# A Basic I2C Transaction



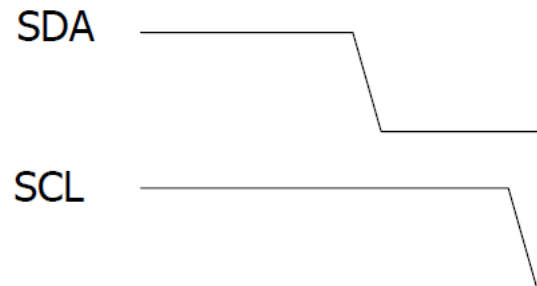
- Master always initiates transactions
- Start Condition
- Address
- Data
- Acknowledgements
- Stop Condition



Source: ATmega8 Handbook

- Transmitter/Receiver differs from Master/Slave
- Master initiates transactions, slave responds
- Transmitter sets data on the SDA line, Receiver acknowledges
  - For a read, slave is transmitter
  - For a write, master is transmitter

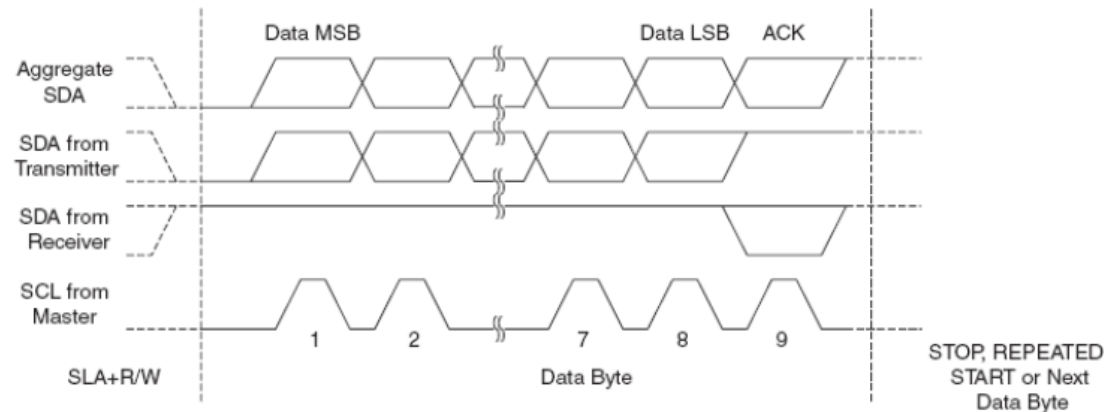
- Master pulls SDA low while SCL is high
  - Normal SDA changes only happen while SCL is low





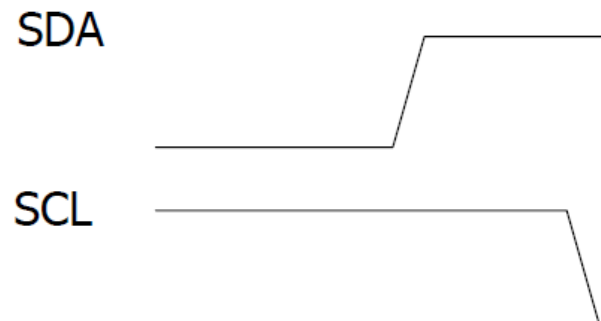
- Data is always sampled on rising edge of clock
- Address is 7 bits
- An 8th bit indicates read or write
  - High for read, low for write
- Addresses assigned by Philips/NXP (for a fee)

- Transmitted just like address (8 bits)
- For a write, master transmits, slave acknowledges
- For a read, slave transmits, master acknowledges
- Transmission continues with subsequent bytes until master creates stop condition

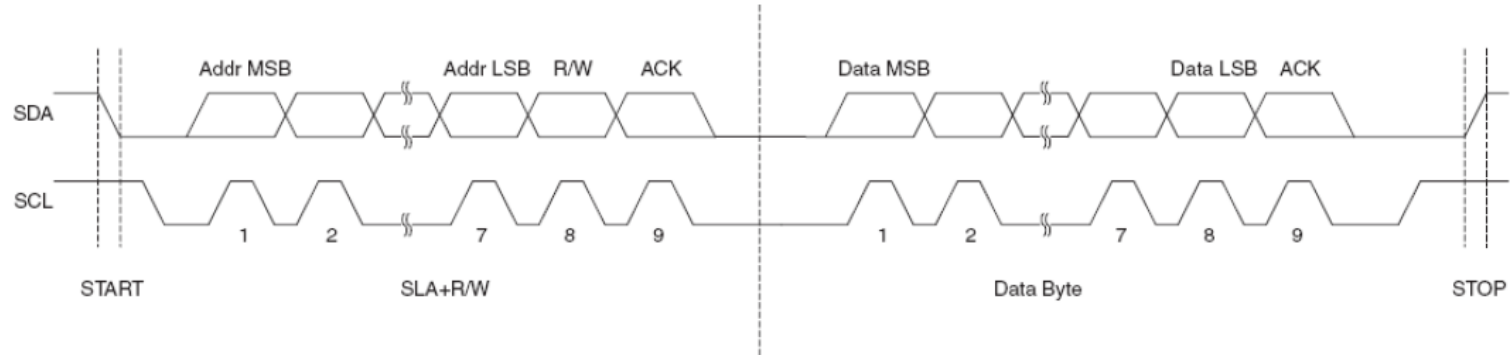


Source: ATmega8 Handbook

- Master pulls SDA high while SCL is high
- Also used to abort transactions

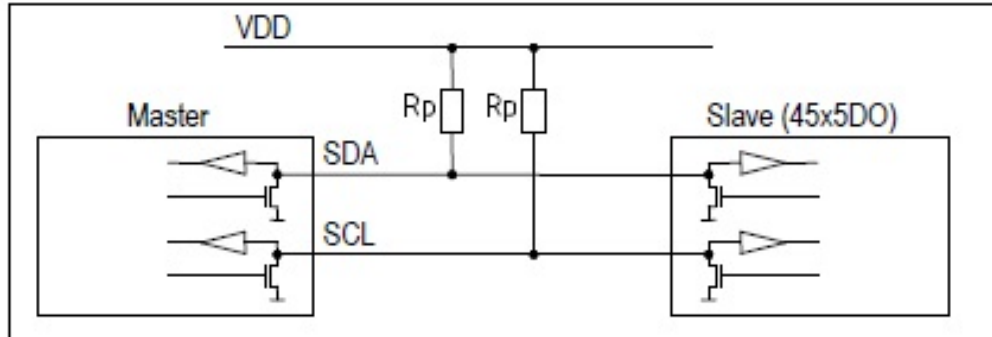


# Another look at I2C

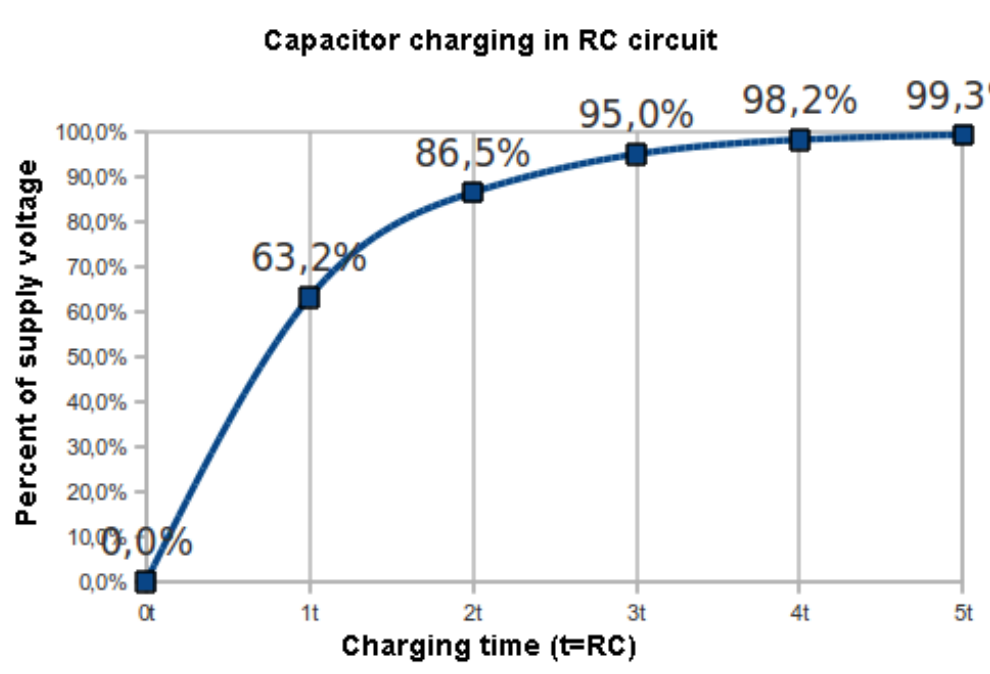


Source: ATmega8 Handbook

# Exercise: How fast can I2C run?



- How fast can you run it?
- Assumptions
  - 0's are driven
  - 1's are "pulled up"
- Some working figures
  - $R_p = 10 \text{ k}\Omega$
  - $C_{\text{cap}} = 100 \text{ pF}$
  - $V_{\text{DD}} = 5 \text{ V}$
  - $V_{\text{in\_high}} = 3.5 \text{ V}$
- Recall for RC circuit
  - $V_{\text{cap}}(t) = V_{\text{DD}}(1 - e^{-t/\tau})$
  - Where  $\tau = RC$



## Exercise: Bus bit rate vs Useful data rate



- An I2C “transactions” involves the following bits
  - $\langle S \rangle \langle A6:A0 \rangle \langle R/W \rangle \langle A \rangle \langle D7:D0 \rangle \langle A \rangle \langle F \rangle$
- Which of these actually carries useful data?
  - $\langle S \rangle \langle A6:A0 \rangle \langle R/W \rangle \langle A \rangle \langle D7:D0 \rangle \langle A \rangle \langle F \rangle$
- So, if a bus runs at 400 kHz
  - What is the clock period?
  - What is the data throughput (i.e. data-bits/second)?
  - What is the bus “efficiency”?