Serial Data Communications

This lecture note was adapted from the following materials:

1) **Fast and Effective Embedded System Design Applying the ARM mbed** by Rob Toulson and Tim Wilmshurt
2) AN10216-01 **I²C manual**, Philips Semiconductor
3) Lecture note: **Design of Microprocessor-Based Systems**
   by Dr. Prabal Dutta
4) http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

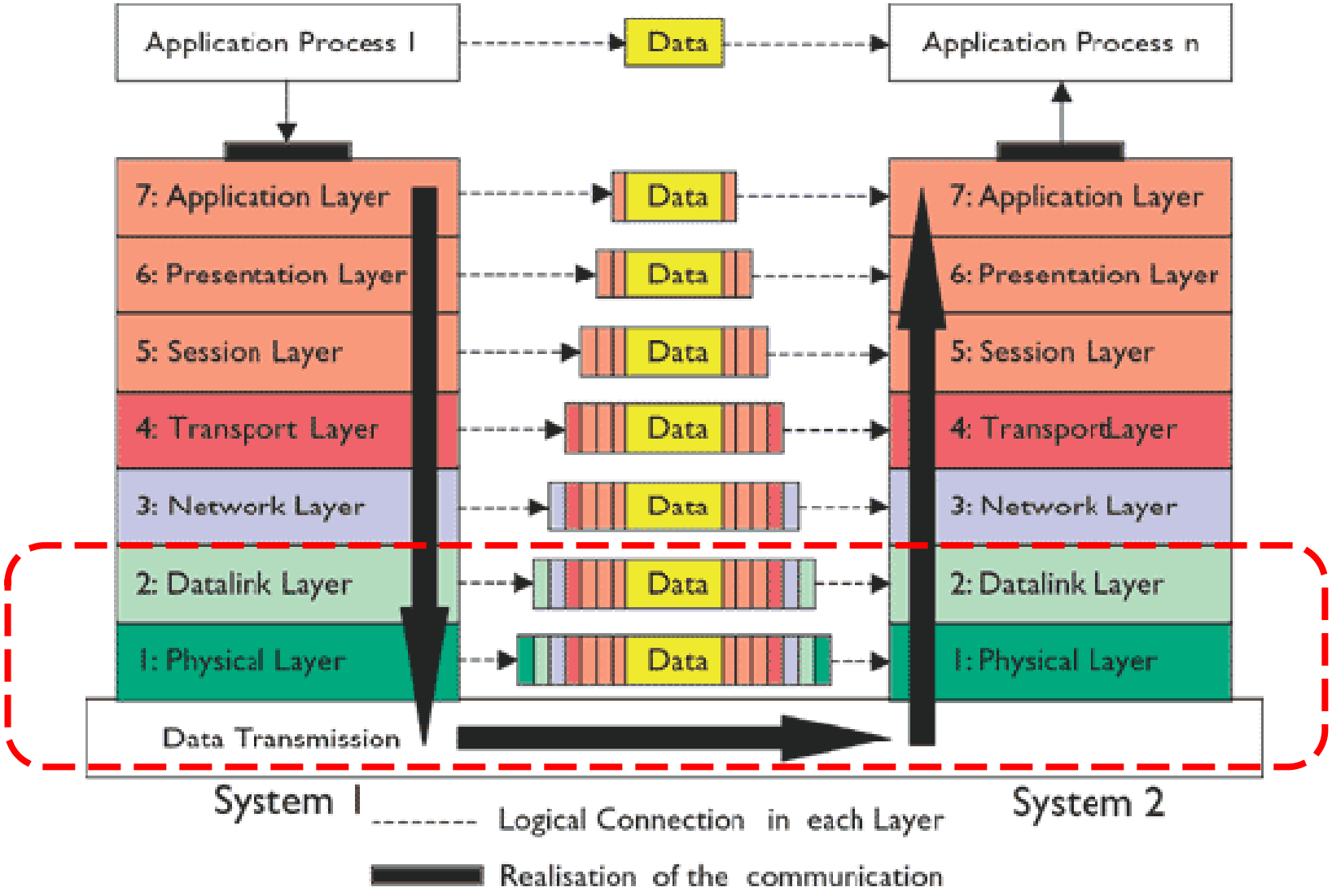
Outline

- Serial vs Parallel
- Synchronous/Asynchronous
- Commonly used Serial: SPI, I2C, UART, USB, CAN
  - SPI
  - I2C
  - UART

Data Communication

- Why ?
- Endless need in computer systems to move data around
  - Inside the Chip
    - Data bus and address bus
  - Inter-chip (in system)
    - MCU to MCU
    - MCU to peripherals
  - Between computer systems
    - Computer network

OSI Network Model



Serial vs Parallel

- Parallel
  - Transfer a whole ***word*** at a time, one wire for each bit
  - Requires a lot of wires : 8, 16, 32
  - Ex: Internal buses: data bus, address bus
- Serial
  - A single wire is effectively used for data transfer.
  - Transfer each ***bit*** at a time
  - EX: External buses between MCU-MCU and MCU-peripheral

Serial Data Communication

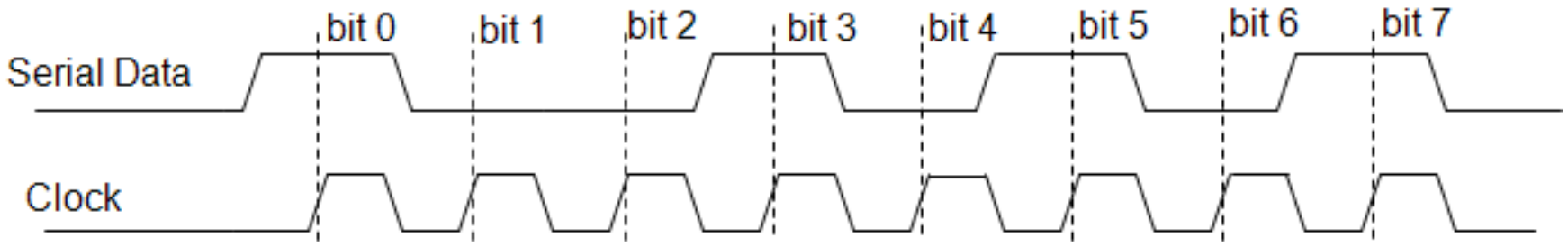
- Challenges
  - How does the receiver know when each bit begins and ends?
  - How does it know when each word begins and ends?
- Solutions
  - 1. Send a clock signal alongside the data, with one clock pulse per data bit. The data is synchronized to the clock.
    - Called Synchronous communication
  - 2. No explicit clock wire but receiver can get timing information from received data
    - Called Asynchronous communication

Synchronous vs Asynchronous

- The term **synchronous** is used to describe any direct communication where the data transmission is time synchronized.
  - Transmitter (Tx) and receiver (Rx) are synchronized by a common timing signal
  - In most cases an explicit clock signal will be shared between Tx and Rx.
  - Tx and Rx are **always connected.**
- The term **asynchronous** means NOT synchronous;
  - The transmission of data without the use of an **external** clock signal.
  - Any timing required to recover the data is encoded within the data itself.
  - Data flow between Tx and Rx can be intermittent (i.e. no need to maintain connection for purpose of synchronization).

Serial Communication

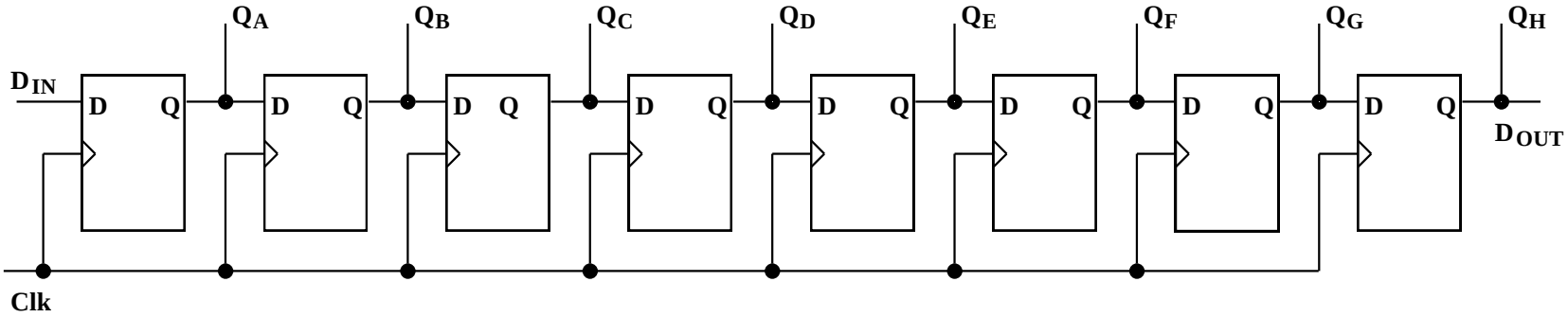
- Synchronous serial data



- Receiver synchronizes its reading of the data with one edges of the clock

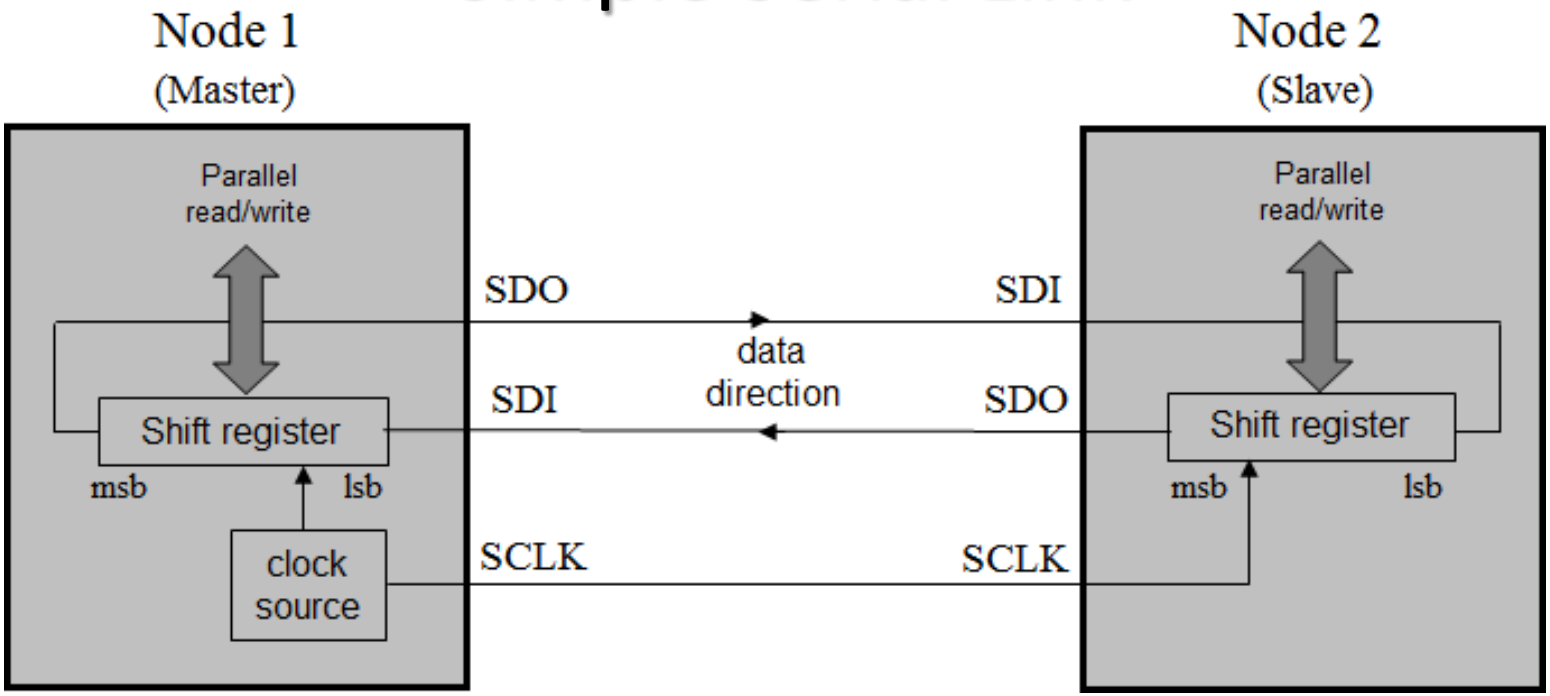
Serialize/De-serialize

- Convert parallel data to serial data and vice versa
- Can be done using ***shift register***



- As the clock pulses the shift register can be both feeding in external data and outputting data.
- The data held by all the flip-flops in the shift register can, moreover, be read all at the same time, as a parallel word, or a new value can be loaded in.

Simple Serial Link

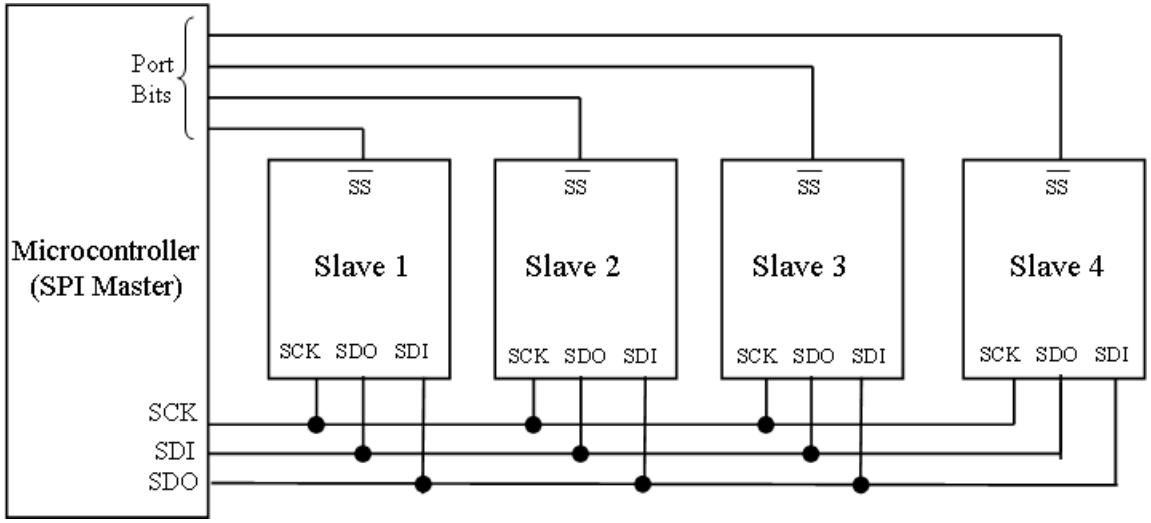


- Node 1 is designated ***Master***; it controls what's going on, as it controls the clock.
- The ***Slave*** is similar to the Master, but receives the clock signal from the Master.
- Suppose both shift registers in are 8-bit, after eight clock cycles, the word that was in the master is now in the slave, and the word that was in the slave is now held in the master

Serial Peripheral Interface (SPI)

- A simple protocol of serial link started by National Semiconductor (called Microwire) and Motorola (called SPI)in early 1970s
- Were adopted by others to the point where they acted as a formal standard
- Link between MCU and MCU, MCU and peripheral
- One MCU is designated as the Master
  - Control all activities
  - Communicate with one or multiple slaves

SPI Basic and Capabilities

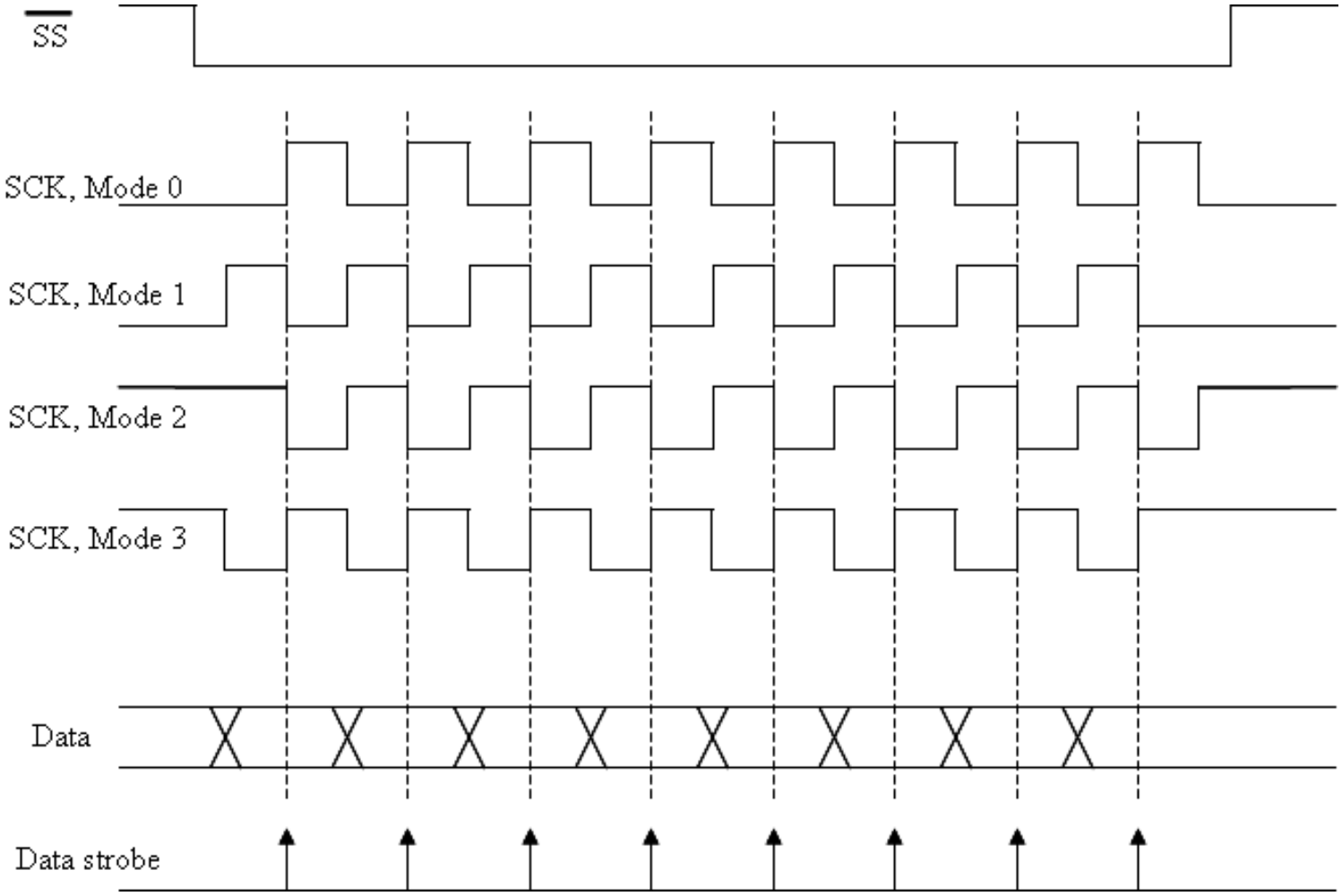


- Master generate and controls clock and data transfer
- SDO of one device is connected to the SDI of the other, and vice versa.
- Data is transferred in both directions : ***full duplex***
- If data transfer in only one direction is wanted, then the data transfer line that is not needed can be omitted.

SPI Basic and Capabilities

- Only one slave active at a time, determined by which **Slave Select (SS) or Chip Select (CS)** line the master activates.
- Slaves can not directly communicate to each other
- For *n* slave, it needs (*n*+3) lines
- Multiple Mbps transmission speed
- Transfer data in 8 to 16 bit
- 4 clocking modes determined from
  - The choices of which clock edge is used to clock data
  - Whether the clock idles high or low
  - Default is mode 0

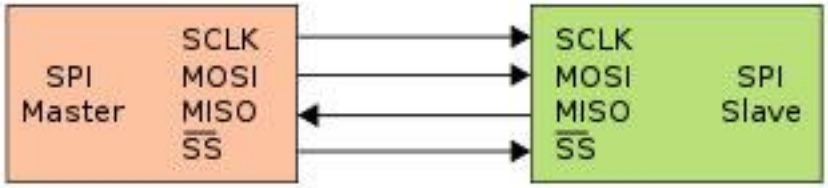
Clocking Modes



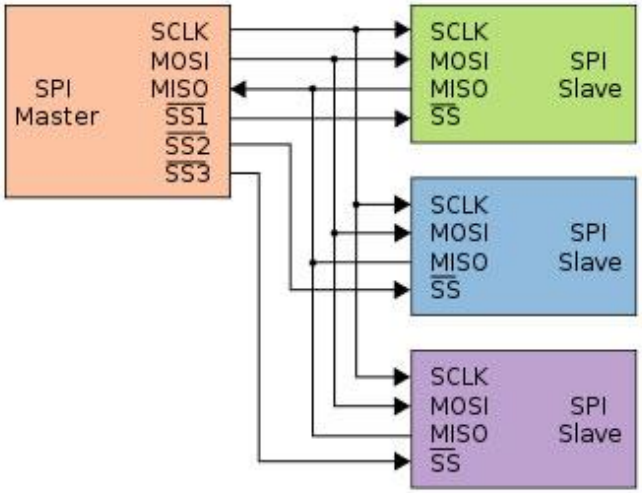
SPI Clocking with Multiple Slaves

- During transfers with slaves A and B, Master must
  - Configure clock to Slave A's clock mode
  - Select Slave A
  - Do transfer
  - Deselect Slave A
  - Configure clock to Slave B's clock mode
  - Select Slave B
  - Do transfer
  - Deselect Slave B
- Master reconfigures clock mode on-the-fly

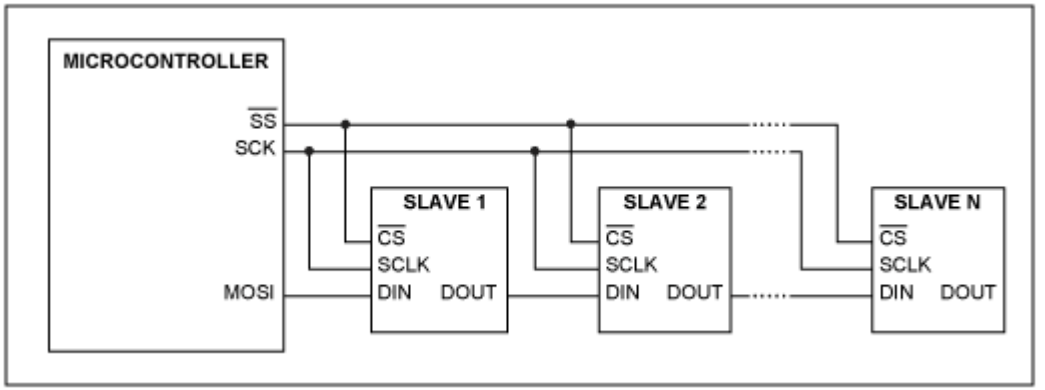
SPI Bus Configuration Models



A master and a slave



A master and multiple slaves



A master and multiple daisy chained slaves

Evaluating SPI

- Pros : Simple,convenient and low cost
  - Fast for point-to-point connections
  - Easily allows streaming/constant data inflow
  - No addressing in protocol, so it’s simple to implement
  - Broadly supported

- Cons : Not appropriate for complex or high-reliably system
  - Slave select/chip select makes multiple slaves more complex
  - No acknowledgement from receiver (if data has been received)
  - No error checking (no way to detect or correct error)

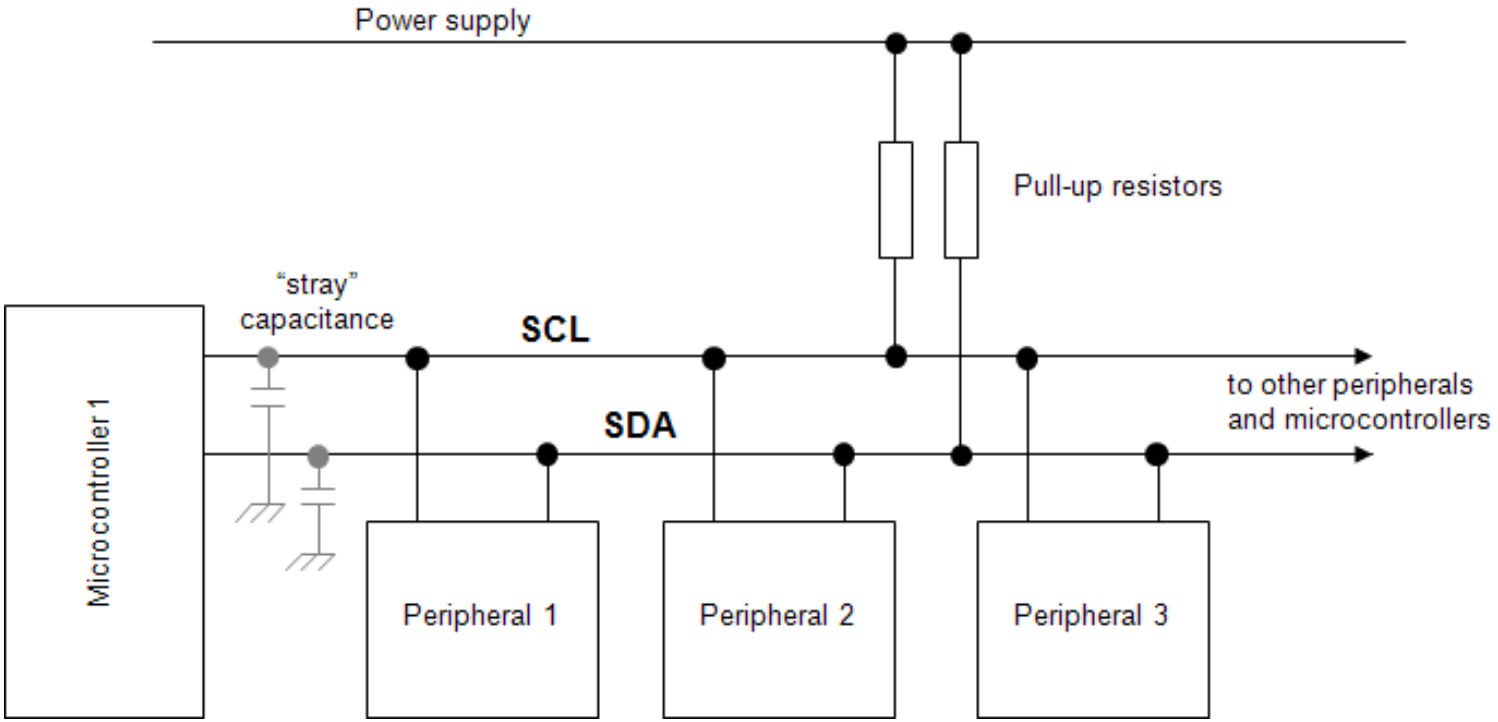
SPI Applications

- Peripherals
  - LCDs
  - Sensors
  - Memory (EEPROM)
  - Lots of other chips
- Microcontrollers
  - Almost all MCUs have SPI masters
  - Some have SPI slaves

Inter-Integrated Circuit Bus (I2C)

- Two wire synchronous serial bus
  - Invented by Philips (now NXP) in the early 1980s
  - Intended for interconnection over short distances and generally within a piece of equipment such as TV
  - Now a very common peripheral bus standard use in embedded systems
  - Also use in PCs
    - RTC
    - Temp sensors
    - System management bus (on motherboard)

Basic I2C

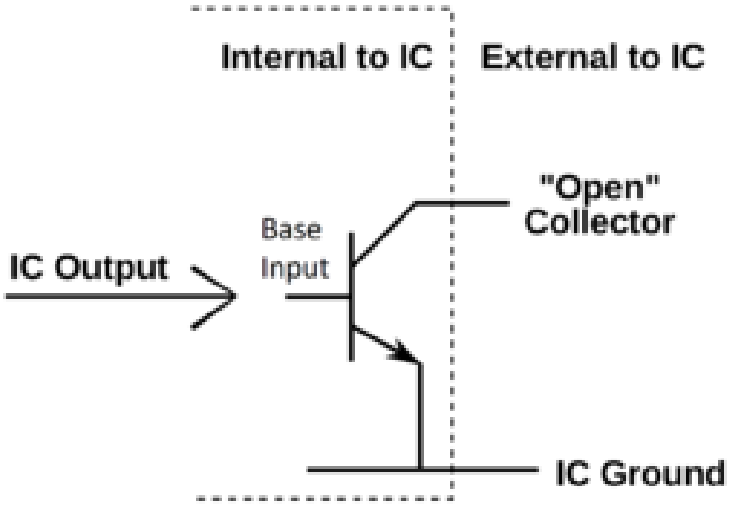


- All devices on the bus are connected to these two lines
  - SDA (serial data)
  - SCL (serial clock)

Basic I2C

- The SDA line is bidirectional, so data can travel in either direction, but only one direction at any one time. This is called ***half duplex***.
- Using open collector (or open drain) design
  - Use a single pull-up resistor (2.2k - 4.7k depending on amount of stray capacitor) connected to each line.
  - Any node connected to it can only pull down the SCL or SDA line to logic 0
  - Simplifies interfacing for multi-voltage level
  - Supports bus arbitration

Open Collector/Drain



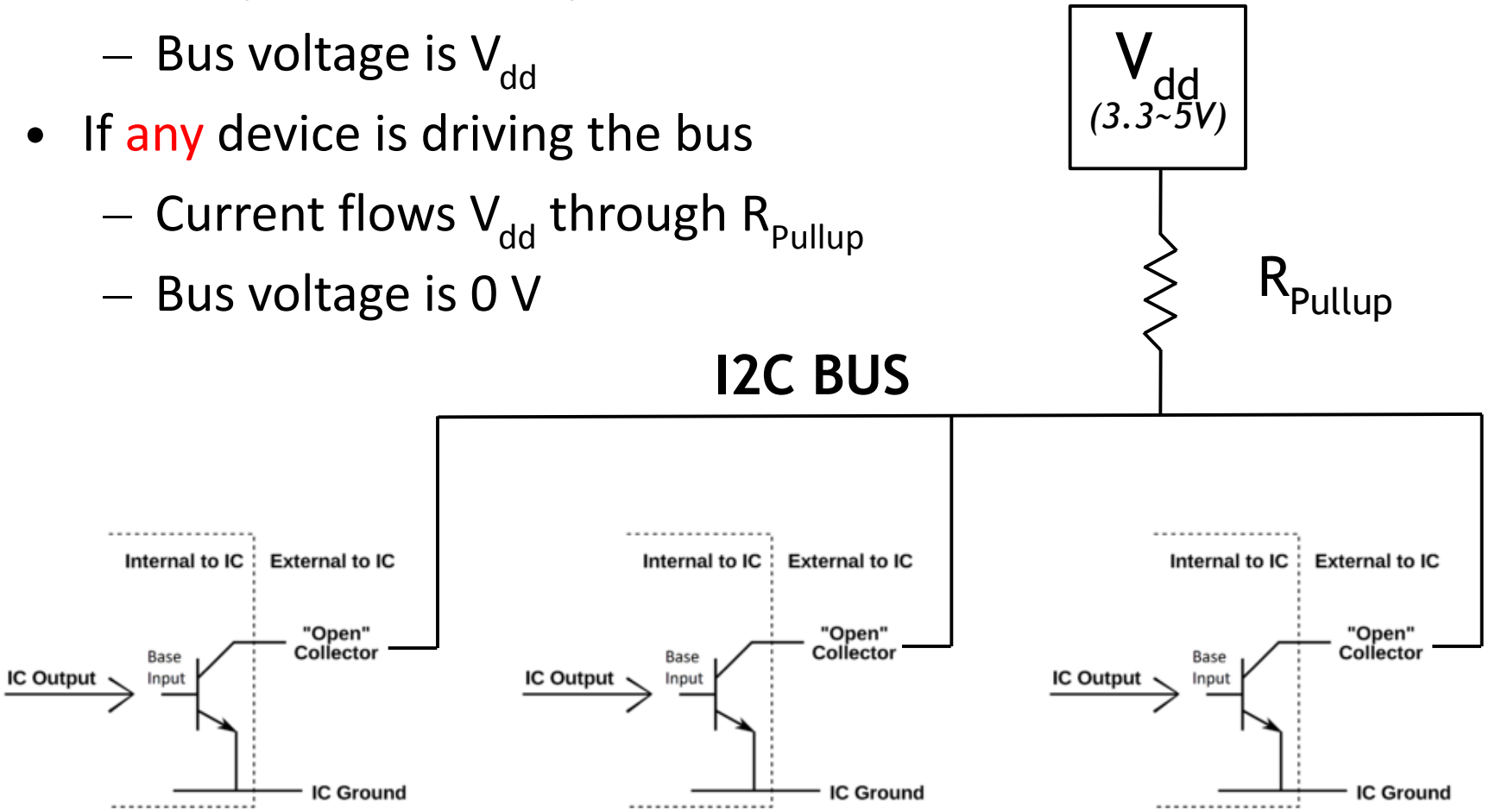
- If IC Output is

  Not Driven (OFF): The transistor is off and looks like a open circuit

  Driven (ON): The transistor is on, and the bus line is tied (pulled) to ground

I2C Bus

- If no device is driving the bus
  - They all look like open circuits
  - Bus voltage is V_{dd}
- If any device is driving the bus
  - Current flows V_{dd} through R_{Pullup}
  - Bus voltage is 0 V



Basic I2C

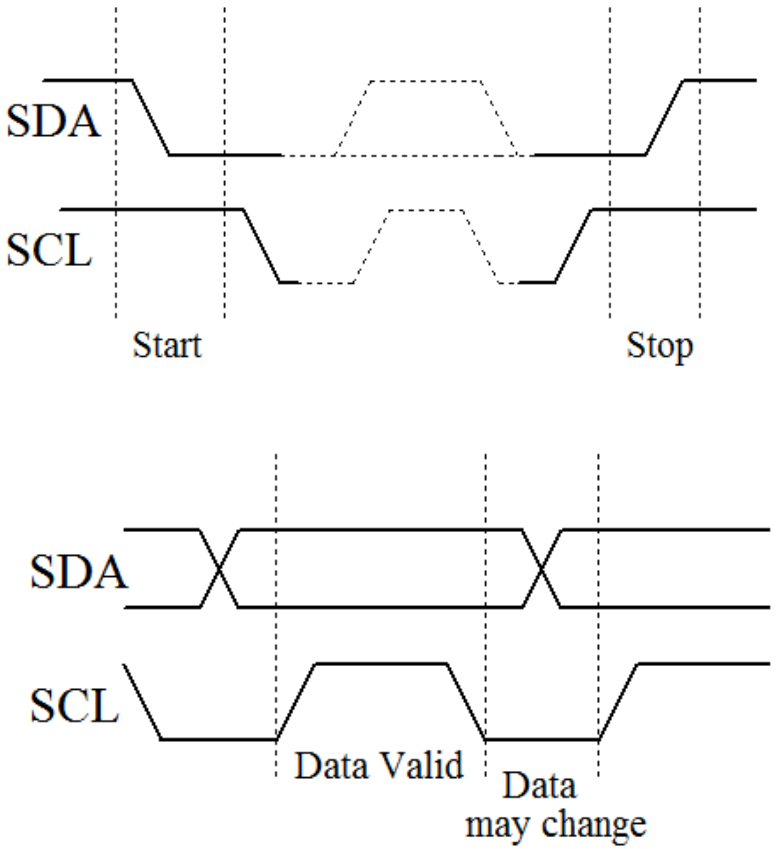
- Nodes on an I2C bus can act as master or slave.
- The master initiates and terminates transfers, and generates the clock.
- The slave is any device addressed by the master.
- A system may have more than one master, although only one may be active at any time.
- An arbitration process is defined if more than one master attempts to control the bus.
- Bit rate upto 3.4 Mbits/s

I2C Clock Characteristics

- Not a “traditional” clock
- Normally is kept “high” using a pull-up
- Pulsed by the master (pull down and then release) during data transmission
  - Master could be either the transmitter or receiver
- Slave device can hold clock low if needs more time
  - Allows for flow control
  - Called “clock stretching”

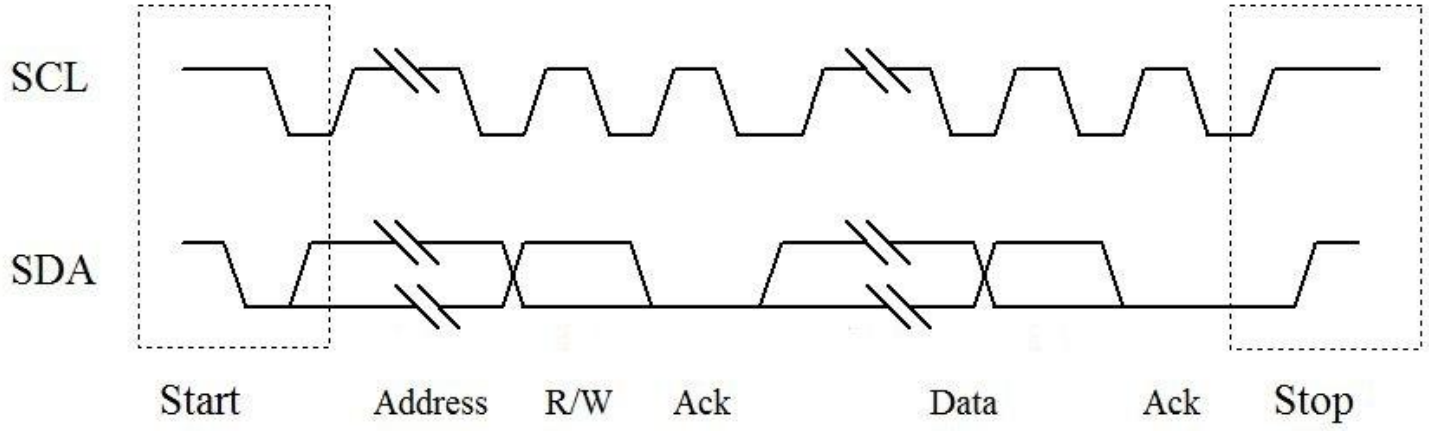
I2C Start/Stop Condition

- A data transfer is made up of the Master signalling a ***Start Condition***, followed by one or two bytes containing **address** and control information.
- The Start condition is defined by a **high to low** transition of SDA when SCL is high.
- A low to high transition of SDA while SCL is high defines a ***Stop condition***
- One SCL clock pulse is generated for each SDA data bit, and **data may only change when the clock is low**.



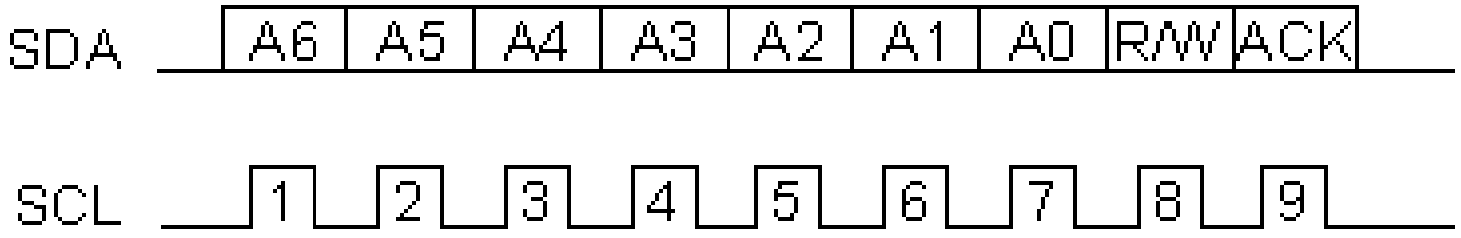
How the data transferred

- The byte following the Start condition is made up of seven address bits, and one data direction bit (Read/Write)
- All data transferred is in units of one byte, with no limit on the number of bytes transferred in one message.
- Each byte must be followed by a ***1-bit acknowledge*** from the ***receiver***, during which time the transmitter relinquishes SDA control.



I2C Device Addressing

- All I2C addresses are either 7 bits (or 10 bits)
- Most are 7-bit; can have 128 devices on the bus
- 8-bit Address byte : 7-bit address + R/W bit on LSB
  - Master read from slave: R/W = 1 ➔ odd address
  - Master write to slave: R/W=0 ➔ even address



The I2C Physical Protocol

- Master initiate the transfer by issuing Start Condition
- Data transfer in 8 bits serially starting with MSB
- For every 8 bit transfer, the transmitter receiving data sends back a ACK bit
  - A **low** ACK meaning it has received the data and ready to accept another byte
  - A **high** ACK meaning it cannot accept any further data and the master should terminate the transfer
- Master terminate the transfer by sending Stop Condition

The I2C Software Protocol

Example: To write to slave device

- 1. Send a Start Condition
  2. Send the I2C address of the slave with the R/W bit low (W=even address)
  3. Send the internal register number you want to write to
  4. Send the data byte
  5. [Optionally, send any further data bytes]
  6. Send the Stop Condition.

Reading from the Slave

To Read a slave device:

- Example: MCU reads a bearing register of the CMPS03 (compass module) at I2C address x60

  1. Send a Start Condition
  2. Send 0xC0 (I2C address of the CMPS03 with the R/W bit low (W=even address)
  3. Send 0x01 (Internal address of the bearing register)
  4. Send a Start Condition again (“Repeated Start”)
  5. Send 0xC1 (I2C address of the CMPS03 with the R/W bit high (R=odd address)
  6. Read data byte from CMPS03
  7. Send the stop sequence.

Clock Stretching

- There is a case that Slave is busy and not ready to communicate such as the slave is another MCU
- When master is reading from the slave and the slave is not ready to place data on the bus
  - Slave will hold SCL low until its ready. This called clock stretching
  - Master check if SCL is high. If SCL is low Master should wait until it goes high (being released)
  - After Slave release SCL, Master continue clocking
- Most of hardware on the Master I2C port are design to handle this. If not, program on the Master must be written to handle instead.

Clock Stretching

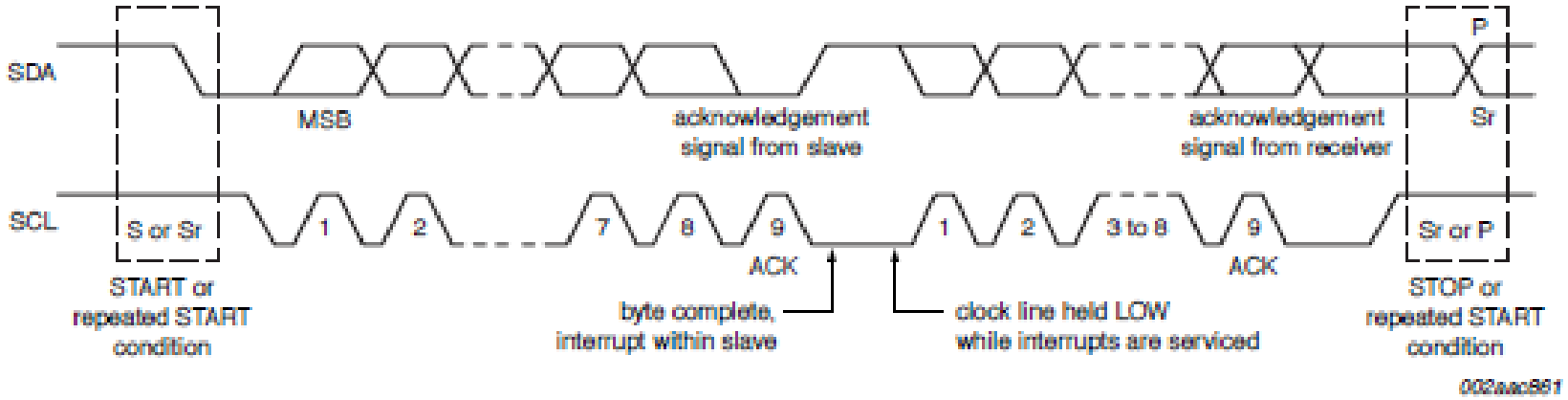


Fig 6. Data transfer on the I^2C-bus

Evaluating I2C

- Better than SPI in
  - Its ability to set up more complex networks, and to add and subtract nodes with comparative ease.
  - Reliability
  - If an addressed device does not send an acknowledgment, the master can act upon that fault.
- Bandwidth is comparatively limited
- Susceptible to interference and does not check for errors.
  - Unlikely to consider using it in a medical, motor vehicle or other high-reliability application