Wired Communication Standards

Kumar Yelamarthi

Central Michigan University

Mt Pleasant, MI

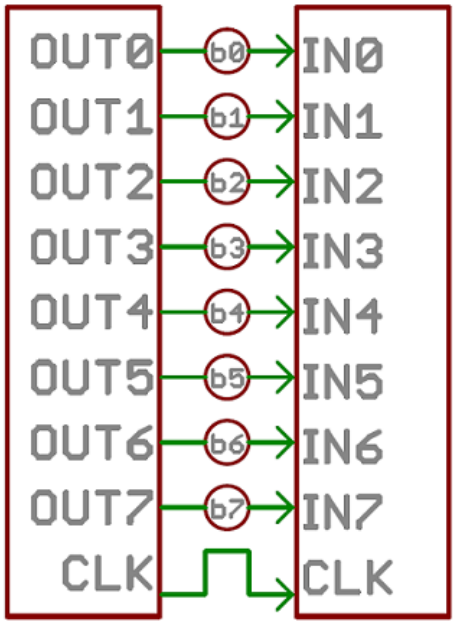
Motivation

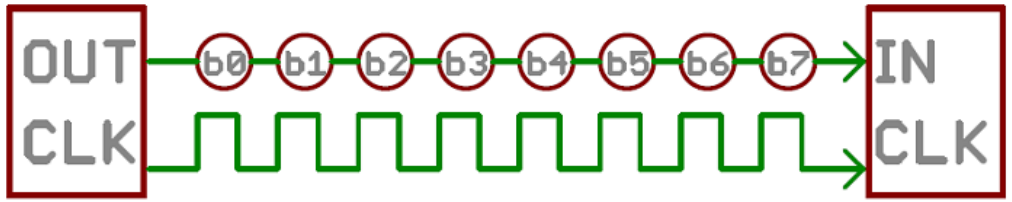
- Connect different systems together
  - Two embedded systems
  - A desktop and an embedded system
- Connect different chips together in the same embedded system
  - MCU to peripheral
  - MCU to MCU
- Without using a lot of I/O lines
  - I/O lines require I/O pads which cost $$$ and size
  - I/O lines require PCB area which costs $$$ and size
- Options?
  - Serial
  - SPI
  - I^2C

Questions to ask in selection process

- 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?
- How do we support flow control?
- How does it handle errors/noise?

Parallel vs. Serial





Source: Sparkfun

- Parallel
  - Transfer multiple bits at the same time.
  - Requires buses of data transmission wires (8, 16, 32,…)
  - Fast, straightforward, relatively easy to implement
  - Requires more input/output lines
- Serial
  - Transfer data one single bit at a time.
  - Requires only two wires (data and clock)
  - Relatively slow
- Which is better suited for microcontrollers?

Synchronous and Asynchronous Serial

- Known serial interface in embedded systems
  - Universal Serial Bus (USB)
  - Ethernet
  - Serial standard
  - SPI
  - I^2C

- Synchronous Serial
  - Pairs its data line with a clock signal, and all devices on serial bus share a common clock
  - Faster serial transfer, but requires at least one extra wire between communicating devices
  - Example: SPI, I^2C

- Asynchronous Serial
  - Data is transferred without support from an external clock signal
  - Ideal for minimizing the required wires and input-output pins.
  - Examples: UART

Rules of Asynchronous Serial, aka Serial

- Number of built-in rules help ensure robust and error free data transfer
  - Baud rate
  - Data bits
  - Synchronization bits
  - Parity bits

- Baud rate
  - Specifies how fast data is sent over a serial line.
  - Usually expressed in units of bits-per-second (bps)
  - Determines how long the transmitter holds a serial line high/low or at what period the receiving device samples its line
  - Can be of any value, but both transmitter and receiver should have the same baud rate
  - Standard baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

Data Packets in Serial transfer

- Each block of data transmitted is actually sent in a packet/frame of bits.

Source: Sparkfun

- Data bits
  - Amount of data in each bit can be from 5 to 9 bits
  - It is important to agree on endianness (is data sent MSB to LSB or vice-versa?)
  - If order not stated, default is LSB sent first

- Synchronization bits
  - Two or three special bits are transferred along with data bits
  - **Start bit** marks the beginning of a data packet. Usually 1 bit
    - Indicated by an idle data line going from 1 to 0
  - **Stop bit(s)** marks the end of a data packet. Configurable to 1 or 2 bit, usually 1 bit.
    - Indicated by stop bit(s) transition back to the idle state by holding the line at 1.

Data Packets in Serial transfer

- Parity bits *(Optional)*
  - A simple and low-level error checking method
  - Options: Odd or Even parity
  - Odd parity bit = 1 if the number of 1's in data is odd
  - Even parity bit = 1 if the number of 1's in data is even

Source: Sparkfun

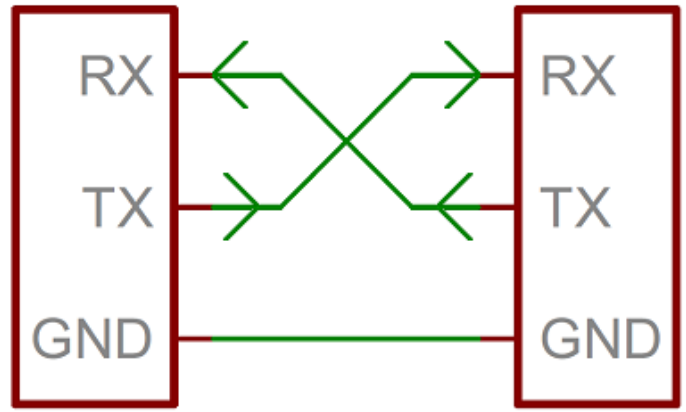
- Example: 9600 8N1 OK
  - Baud rate = 9600
  - Data bits = 8
  - Parity bits = No
  - Stop bits = 1
  - Data = OK

Source: Sparkfun

  - O – ASCII Value = 79 = 01001111
  - K – ASCII Value = 75 = 01001011

Serial – Wiring and Hardware

- Serial bus consists of two wires
  - TX – sends data
  - RX – receives data



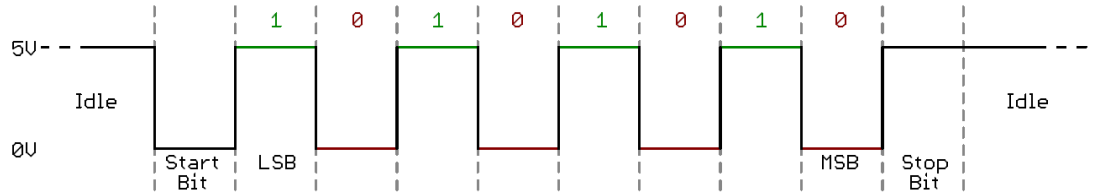
- Interface can operate as **full-duplex** or **half-duplex**
  - Full duplex – both devices can send and receive simultaneously
  - Half duplex – devices must take turns sending and receiving

- Some devices need only one way communi
  - Accordingly, only wire is visible

Source: Sparkfun

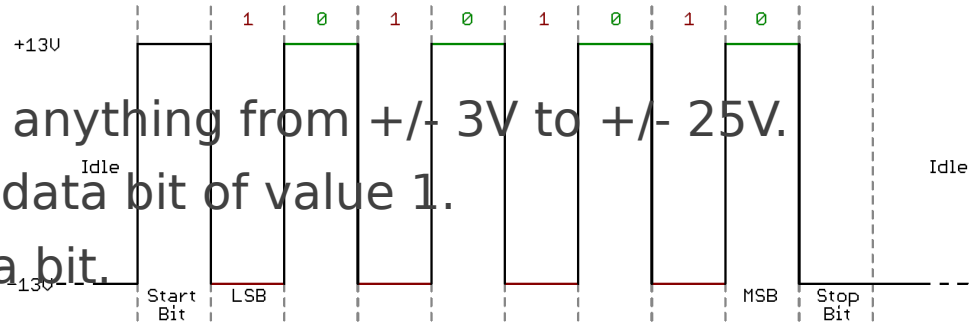
Serial – Hardware Implementation

- Transistor-Transistor Logic (TTL)
  - TTL serial signals exist between a microcontrollers' voltage supply range (usually 0V to 3.3V or 5V)
  - A signal at the VCC level indicates either an idle line, a bit of value 1, or a stop bit.
  - A 0V (GND) signal represents either a start bit or a data bit of value 0.
  - Easier to implement in embedded systems, but susce lines.



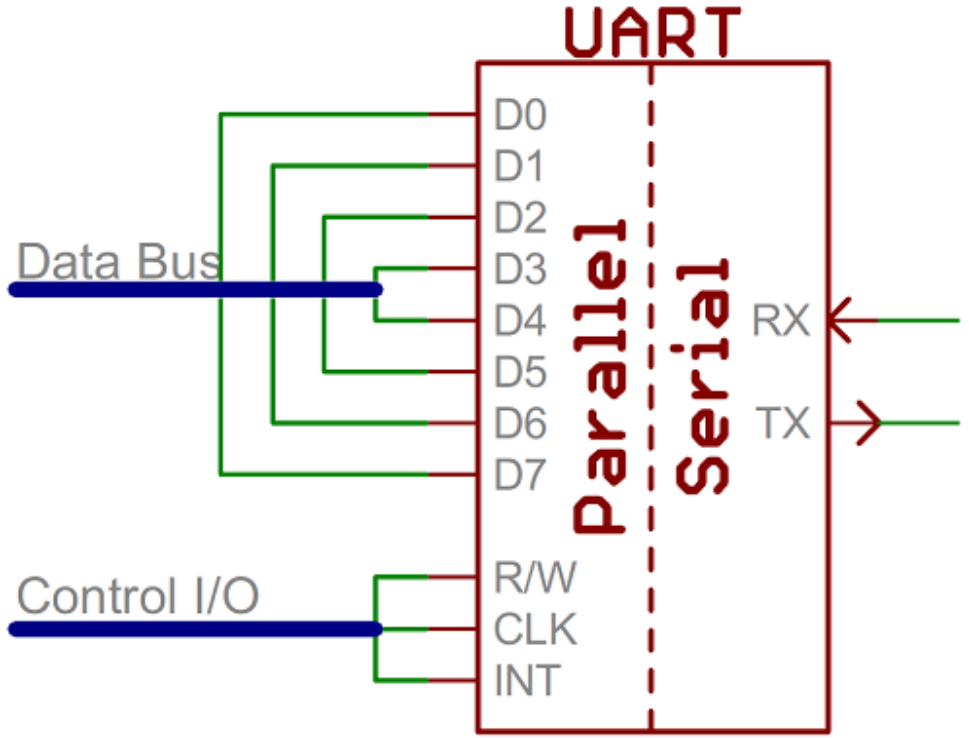
Source: Sparkfun

- RS-232
  - Usually found in old computers and legacy peripherals
  - Usually range between -5V to +5V, though specs allow for anything from +/- 3V to +/- 25V.
  - A low voltage indicates either the idle line, a stop bit, or a data bit of value 1.
  - A high voltage indicates either a start bit, or a 0-value data bit.
  - RS-485 is better suited to long range serial transmissions.



Universal Asynchronous Receiver/Transmitter - UART

- UART is a block of circuitry responsible for implementing serial communication.
- UART acts as an intermediary between parallel and serial interfaces.
- Available as standalone ICs, but generally integrated inside a microcontroller.
- On the transmit side, a UART must create the data packet and send it out with precise timing.
- On the receive side, a UART has to sample RX lines, and extract the data from the packet received

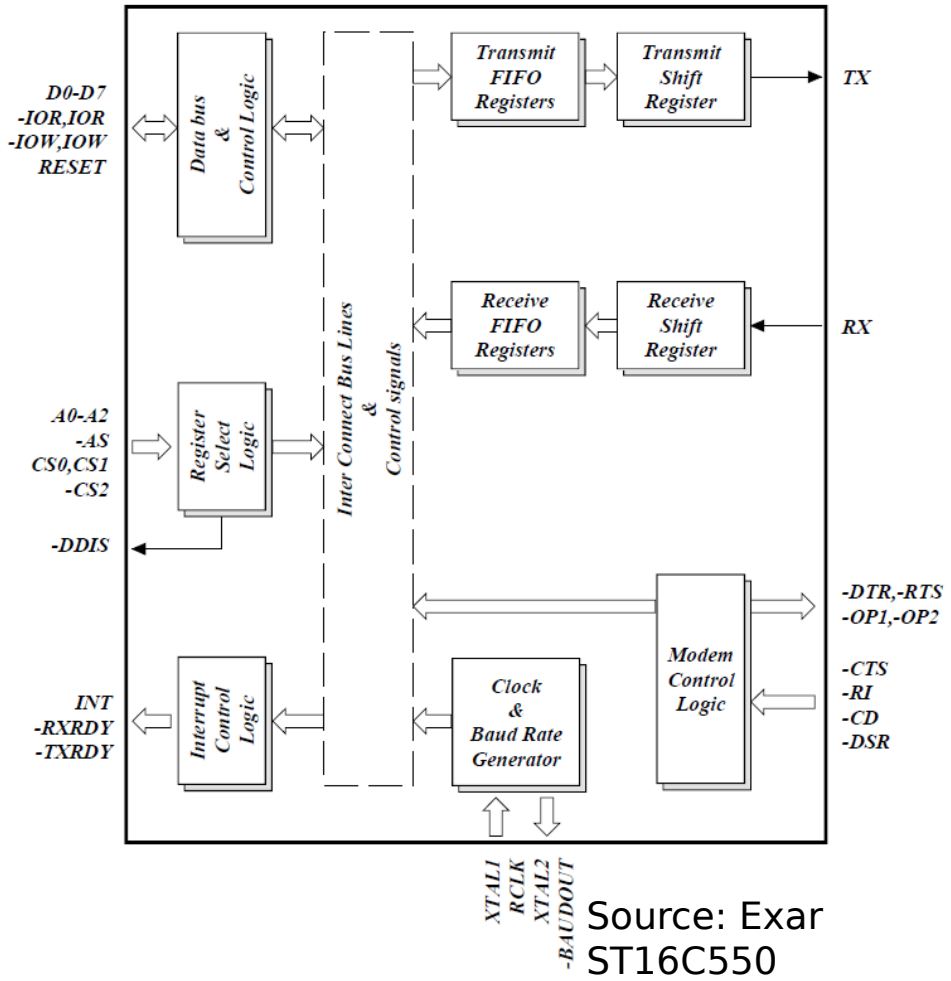


Source: Sparkfun

Universal Asynchronous Receiver/Transmitter - UART

- Advanced UARTs have a built in buffer, where data is stored tentatively for the microcontroller to obtain when timely.
  - Specifically useful in high speed data retrieval

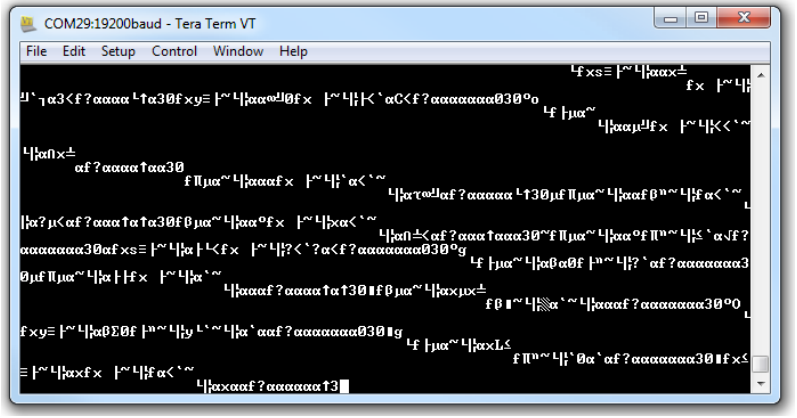
- Software UART
  - Serial interface can be **bit-banged**, by the processor
  - Not an ideal process, but works for simple applications
  - Mostly common known as *SoftwareSerial* in Arduino



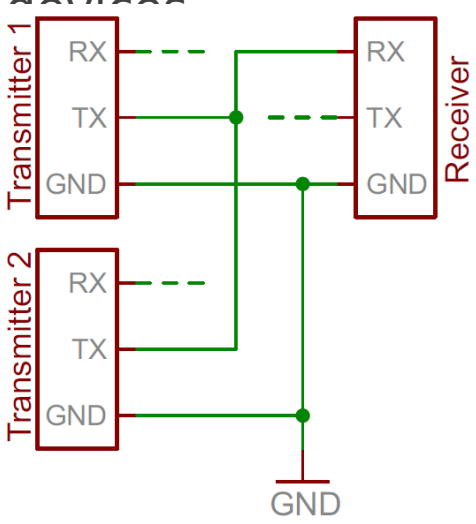
Source: Exar ST16C550

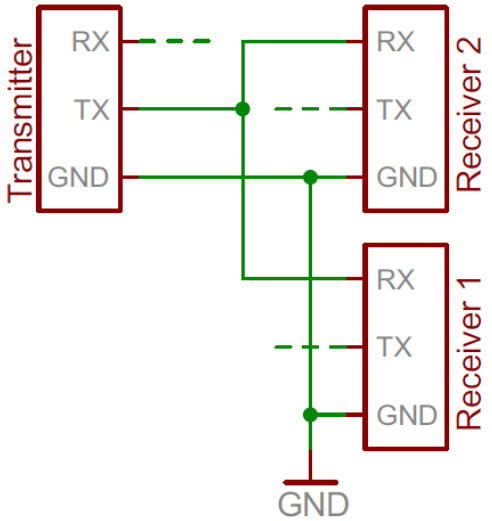
Common Mistakes in Serial Communication

- Baud Rate Mismatch
  - If two devices are not communicating at same speed, data can be misinterpreted or
  - Make sure that baud rate is same on both devices



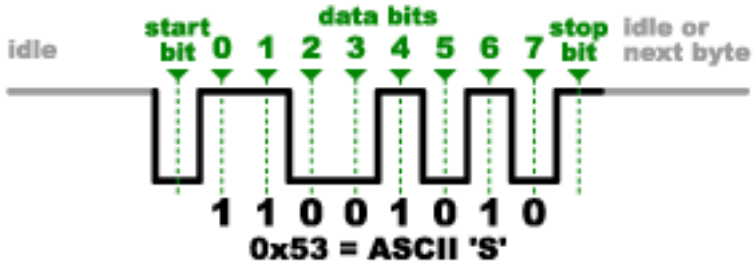
- Bus Contention
  - Serial communication is designed to allow just two devices to communicate across one serial bus.





Limitations of Serial Communication

- Lack of clock signal – no control over when data is sent or any guarantee that both sides are running at precisely the same rate.
- This can be a problem when two systems with slightly different clocks try to communicate with each other.
- Asynchronous serial connections add extra start and stop bits to each byte help the receiver sync up to data as it arrives.
- Both sides must be at the same transmission speed.



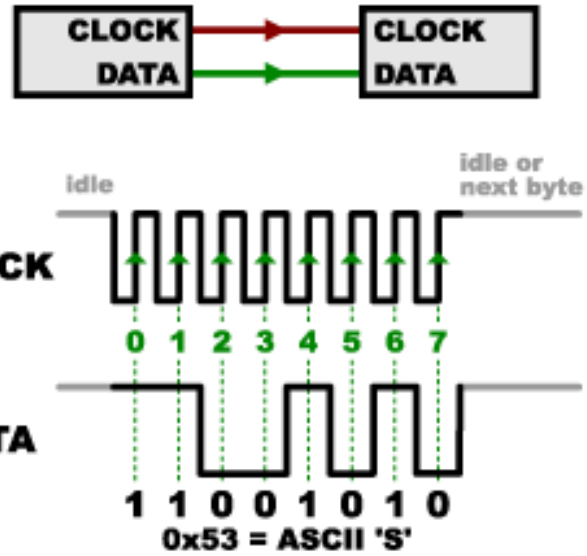
Source: Mike Grusin

- Notice anything unusual with the data in the figure?

- Serial protocols will often send the least significant bits first, so the smallest bit is on the far left.
  - The lower nibble is actually 0011 = 0x3
  - The upper nibble is 0101 = 0x5.

Serial Peripheral Interface (SPI)

- SPI includes a clock signal and provides a synchronous solution
- Uses separate lines for data and clock.
- Clock signal keeps both devices in perfect sync.
- Clock signal informs the receiver when to sample the bits on the data line.
- When the receiver clock line detects the edge, it will look at the data line to read the next bit.



Source: Mike Grusin

- Advantage – a simple shift register can implement this function.

- Limitation in current design?

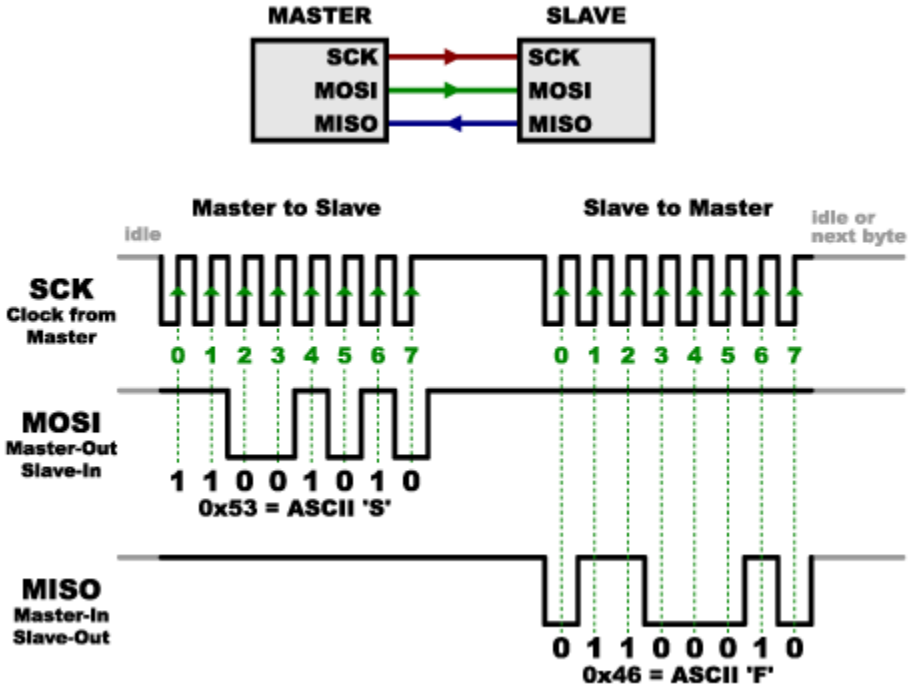
SPI – Receiving Data

- Master – the side that generates the clock
- Slave – the side that receives the clock

- When data is sent from the master to a slave, it's sent on a data line called MOSI -Master Out / Slave In
-
- If the slave needs to respond, the master will continue to generate a prearranged number of clock cycles, and the slave will put the data onto MISO -Master In / Slave Out

- Prearranged - Master must know in advance when a slave needs to return data and how much data will be returned

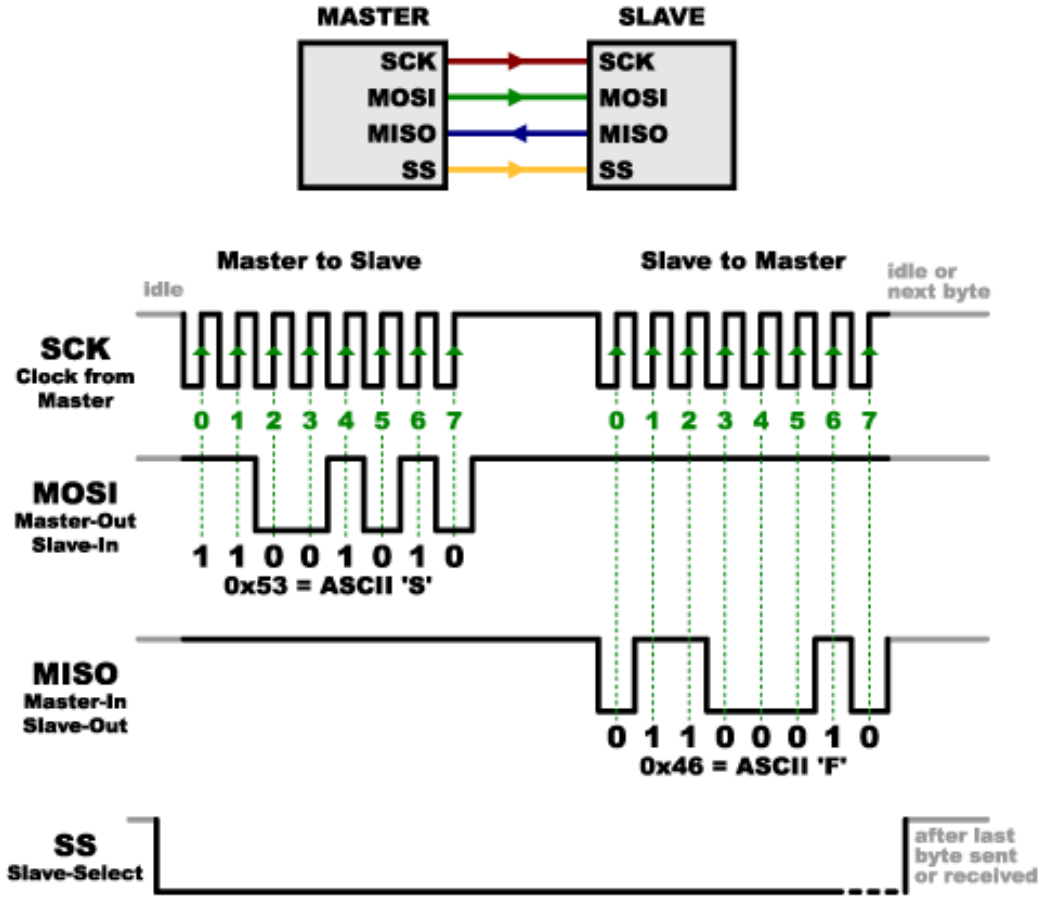
- Full-duplex or half-duplex ?



Source: Mike Grusin

SPI – Slave Select

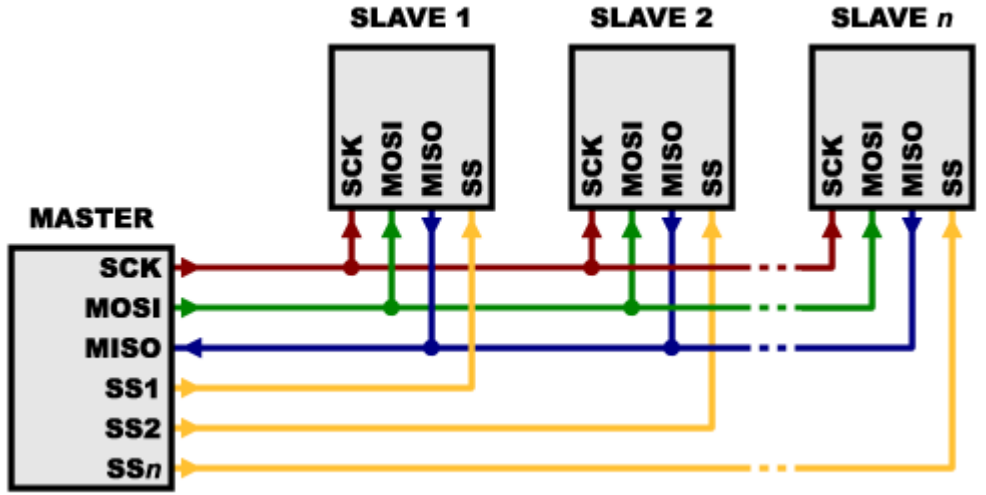
- Slave Select informs the slave that is should wake up and receive/send data.
- Useful when multiple slaves are present to identify the device of interest.
- The SS line is normally held high, which disconnects the slave from the SPI bus.
- Just before data is sent to the slave, the SS line is brought low, which activates the slave.



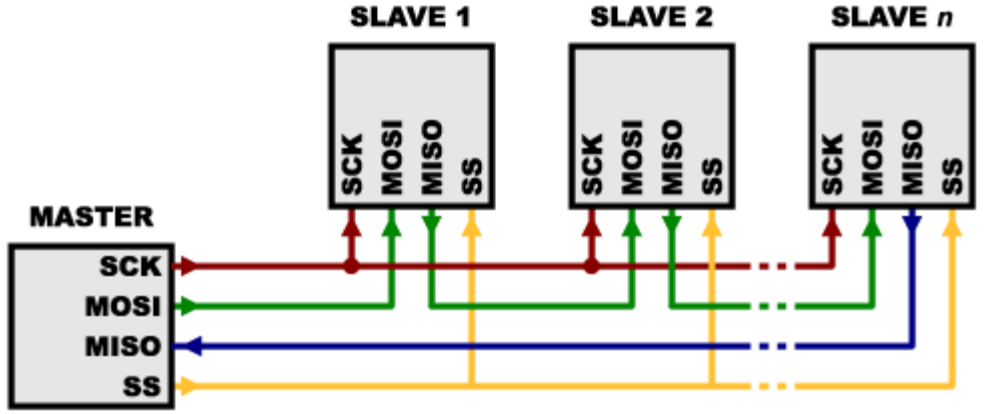
Source: Mike Grusin

SPI – Multiple Slaves

- Option-01
  - Each slave will need a separate SS line
  - Make the SS line of interest to *low*, and retain the other lines *high*
  - Limitation – lots of slaves will require lots of SS lines
    - Alternative: binary decoder chips



- Option-02
  - Daisy chain the MISO of one to MOSI of the other
  - Limitation – data overflows from one slave to the next. You need to transmit enough data to reach all of the devices.
  - *First* piece of data you transmit will end up in the *last* slave



Source: Mike Grusin

SPI - Summary

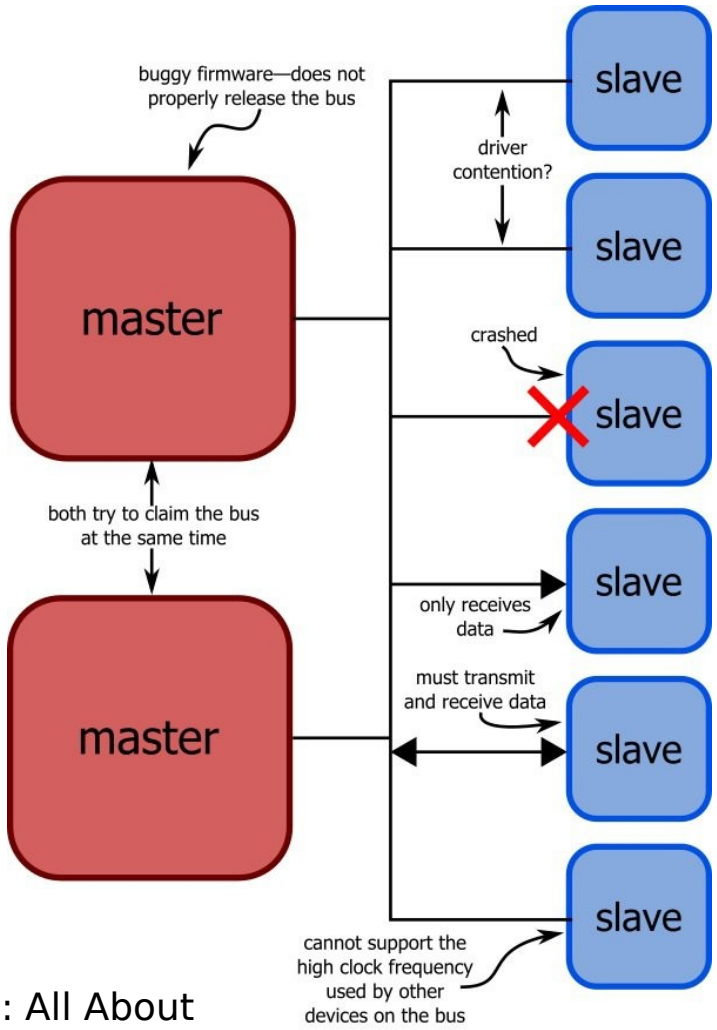
Advantages

- Receiver hardware can be a simple shift register
- Full duplex communication
- Not limited to any maximum clock speed
- Faster than asynchronous serial & high throughput
- Supports multiple slaves

Disadvantages

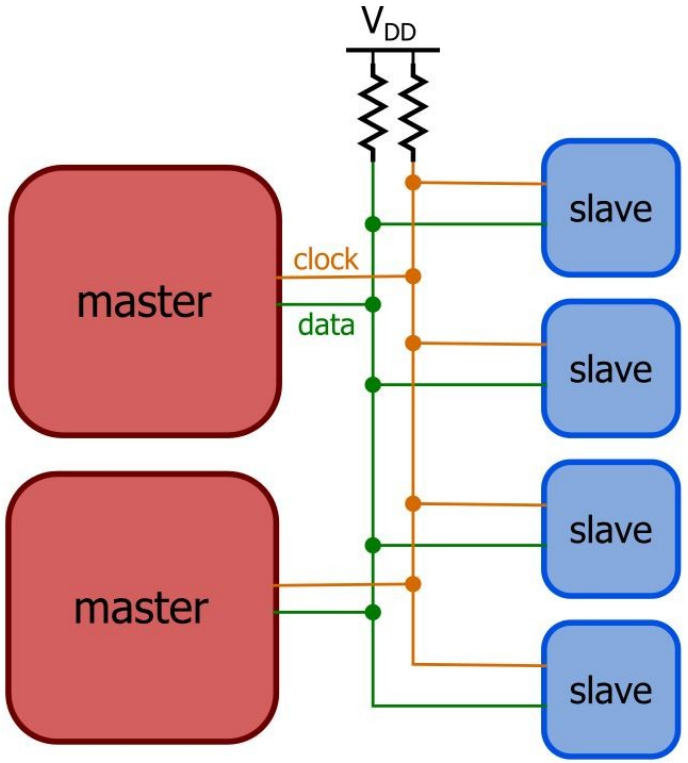
- It requires more signal lines
- The communications must be well-defined in advance
- The master must control all communications
- Requires separate SS lines to each slave
- No hardware slave acknowledgment
- No error-checking protocol defined
- Only handles short distances

Challenges and Potential Solution



Source: All About Circuits

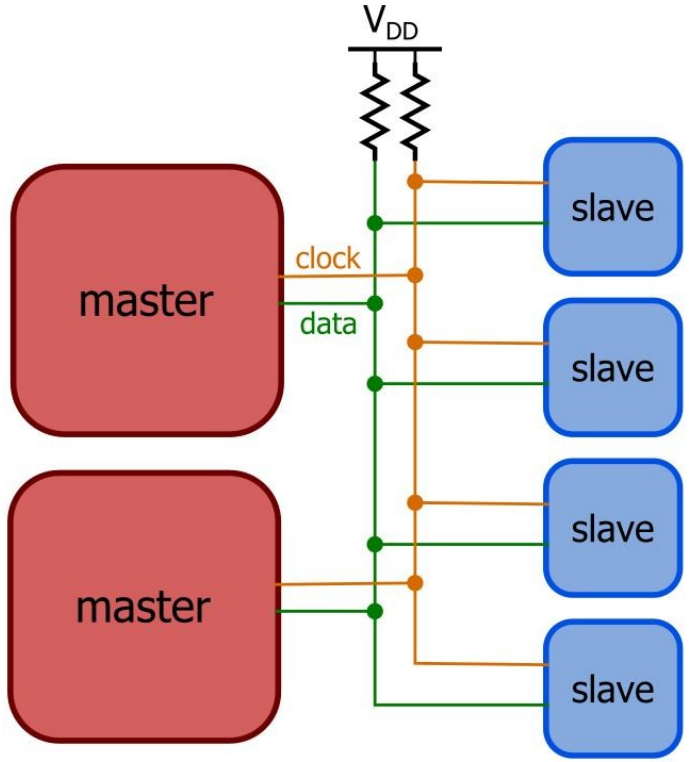
- What if you have multiple slaves?
- What if slaves does not know who the master is?
- What if there are multiple masters?
- What happens if a master requests data from a slave that for some reason has become nonfunctional?
- What if the slave becomes nonfunctional in the middle of a transmission?
- What if a master claims the bus to make a transmission then crashes before releasing



Source: All About Circuits

Inter-Integrated Circuit (I^2C or I2C)

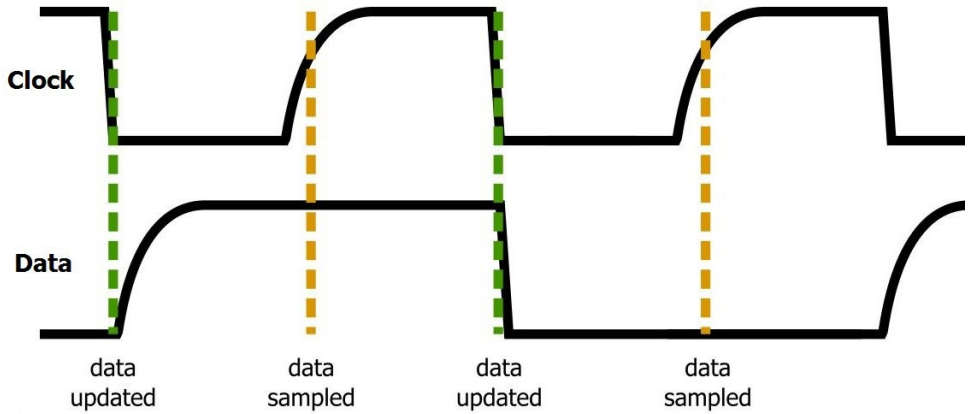
- Only two signals (clock and data) are used, regardless of how many devices are on the bus.
- Both signals are pulled up to a positive power supply voltage through pull-up resistors.
- Each slave device is identified by means of a 7-bit address; the master must know these addresses of each slave.
- All transmissions are initiated and terminated by a master.
- The labels “master” and “slave” are inherently non-permanent: any device can function as a master or slave if it incorporates the necessary hardware and/or firmware.



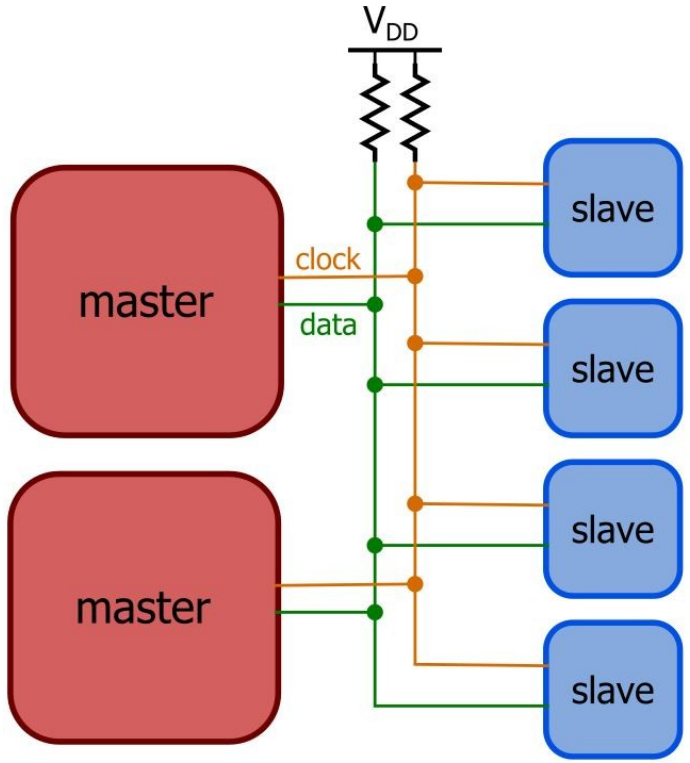
Source: All About Circuits

Inter-Integrated Circuit (I²C or I2C)

- Data signal is updated on the falling edge of the clock signal and sampled on the rising edge.



Source: All About Circuits

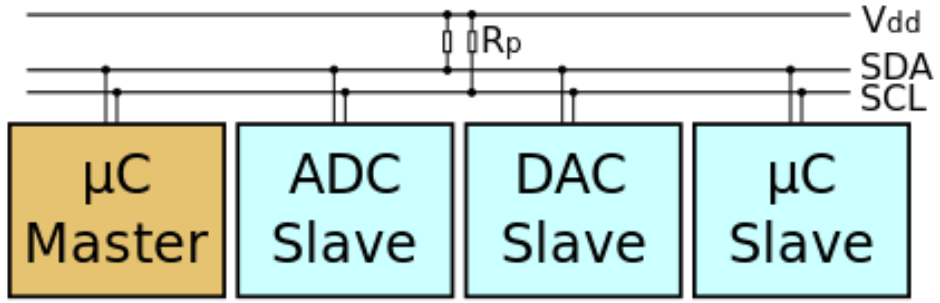


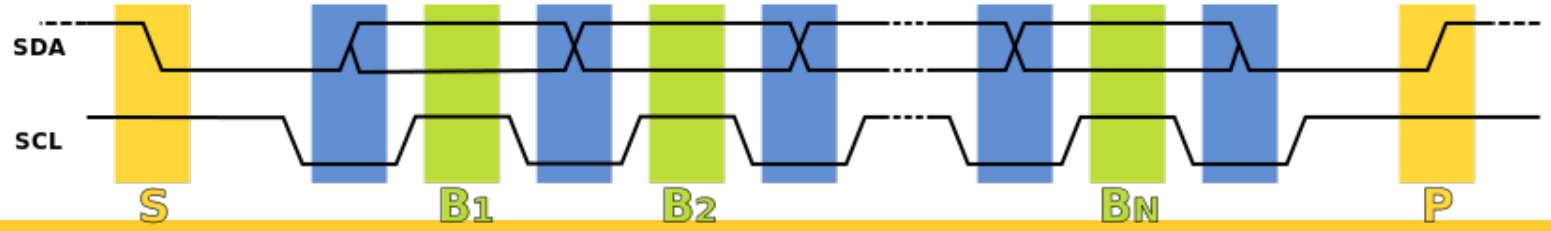
- Data is transferred in one-byte sections, with each byte followed by a one-bit handshaking signal referred to as the ACK/NACK (acknowledge or not-acknowledge) bit.

Source: All About Circuits

I²C Communications

- Master initiates a start condition
- Masters sends the 7-bit unique address of the device
- Master outputs the read or write bit
- Slave sends an acknowledge bit
- Byte of data is exchanged followed by an acknowledgement
- Transaction ends with the master device causing a stop condition





I2C - Summary

Advantages

- Maintains low pin/signal count even with numerous devices on the bus
- Adapts to needs of different slave devices
- Readily supports multiple masters
- Incorporates ACK/NACK functionality for improved error handling

Disadvantages

- Complexity of firmware or low-level hardware
- Imposes protocol overhead and reduces throughput
- Requires pull-up resistors
  - Limit clock speed
  - Consume valuable PCB real estate
  - High power dissipation

SPI vs. I2C

http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/

- Bus topology / routing / resources
- Throughput / Speed
- Elegance