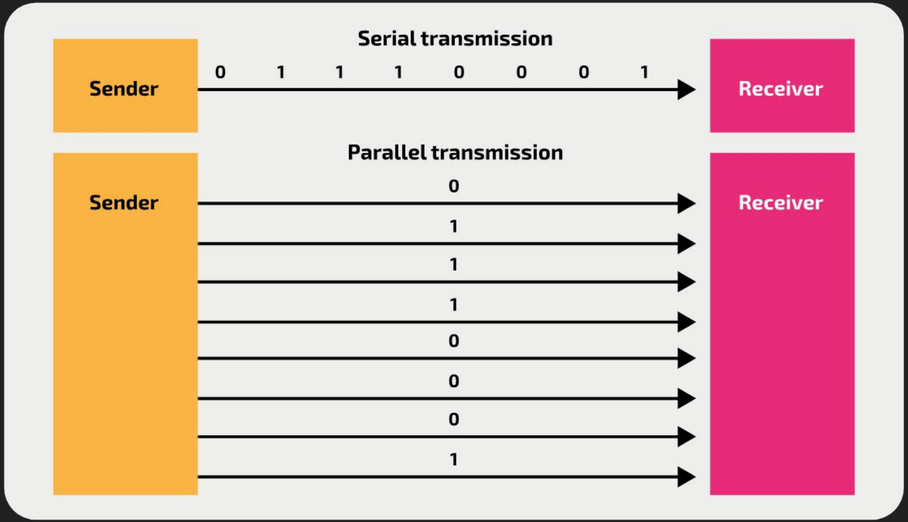
07 - I/O Port Synchronization

CEG 4330/6330 - Microprocessor-Based Embedded Systems

Max Gilson

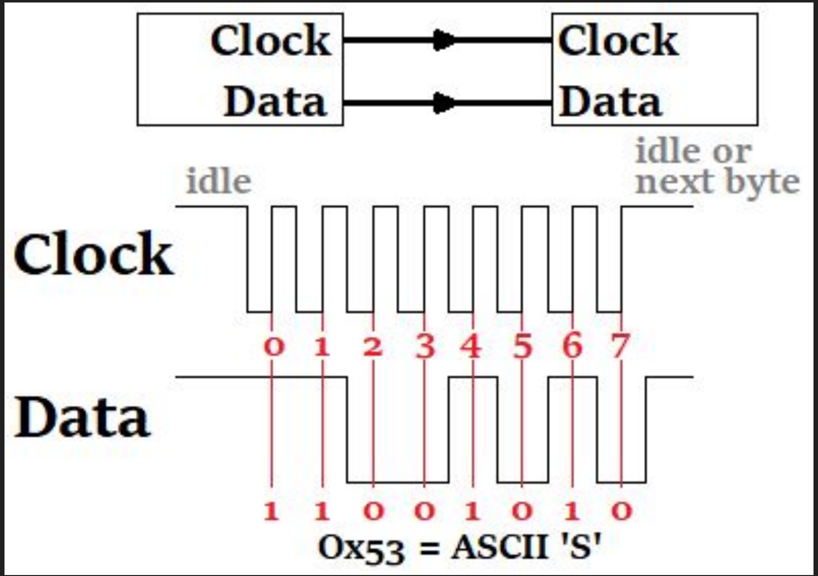
Serial vs Parallel

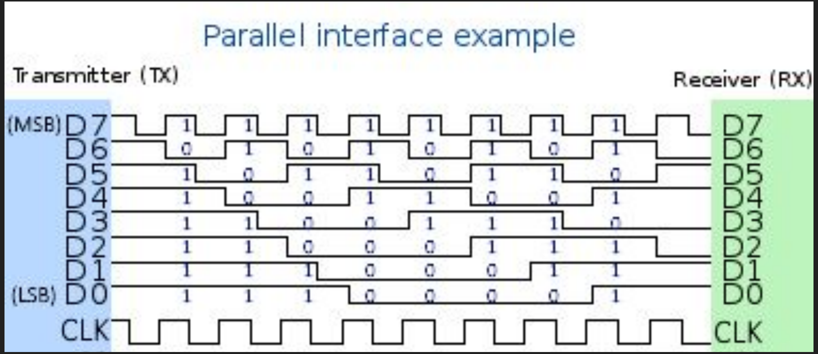
- To communicate between devices or computers, we send bits along wires
- Serial - one bit at a time
- Parallel - multiple bits at the same time



Synchronous

- Synchronous - uses a shared clock signal to distinguish where bits start and stop
    - Allows for much faster data rates (40 Gbps) but requires accurate and clean clock signal
    - Used in USB, I2C, SPI
    - Note: USB can also be asynchronous in some cases



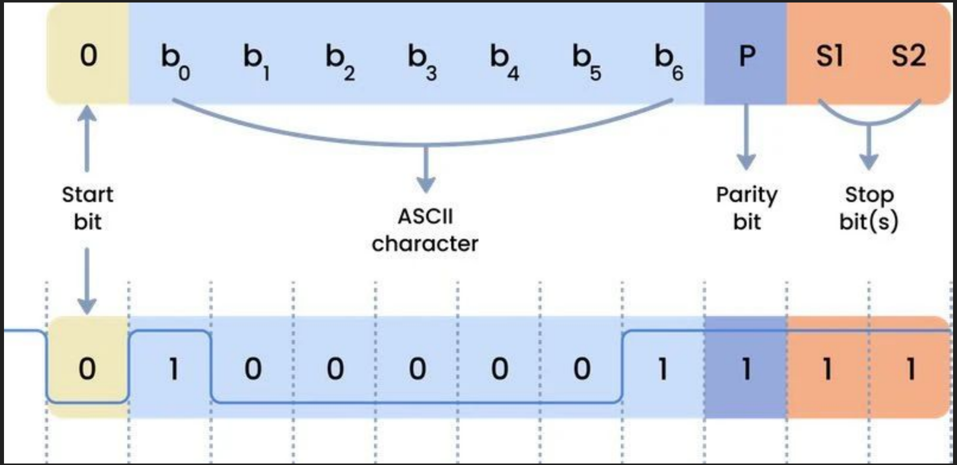


Asynchronous

- Asynchronous - does not use a shared clock signal
  - Baud Rate
    - Devices agree ahead of time to a specific data rate
  - Strobe and Handshaking
    - Receiver waits for a ready signal

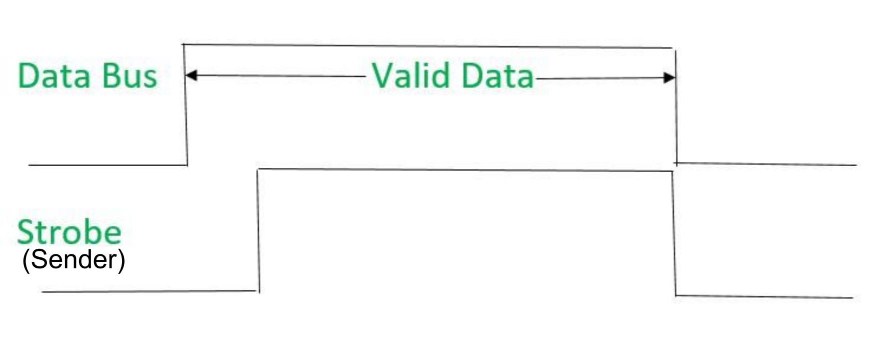
Baud Rate

- Common baud rates: 2400, 9600, 115.2K, 1M, etc.
- For 9600 baud rate, 9600 bits are sent per second
- To send 1 byte must include start bit, stop bit(s), parity bit (optional odd or even)
  - Used for error checking



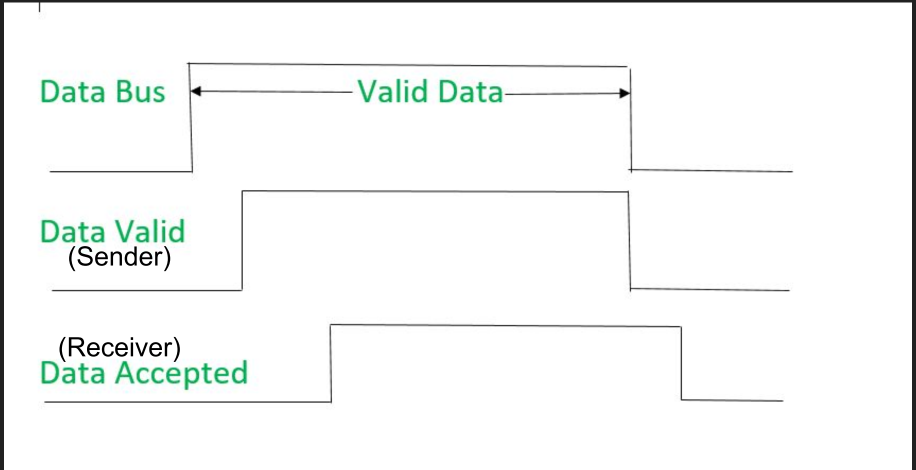
Strobing

- Fast device waits for the slow device to signal “ready”
- Sender says data is “ready” to be received
- Use only when you have a really fast system communicating with a really slow system



Handshaking

- Both devices wait for ready signal from each other
- Sender says data is “ready” to be received
- Receiver says “ready” for next bit



Flow Control

- Assume a printer has a data buffer that gets full quickly when you send a page to print
- Using handshaking you can *control the flow* of data to stop when:
  - The printer’s buffer is full
  - The computer is busy executing something else
- In flow control the data may be sent synchronously or asynchronously, handshaking only controls when large chunks of data are sent and received

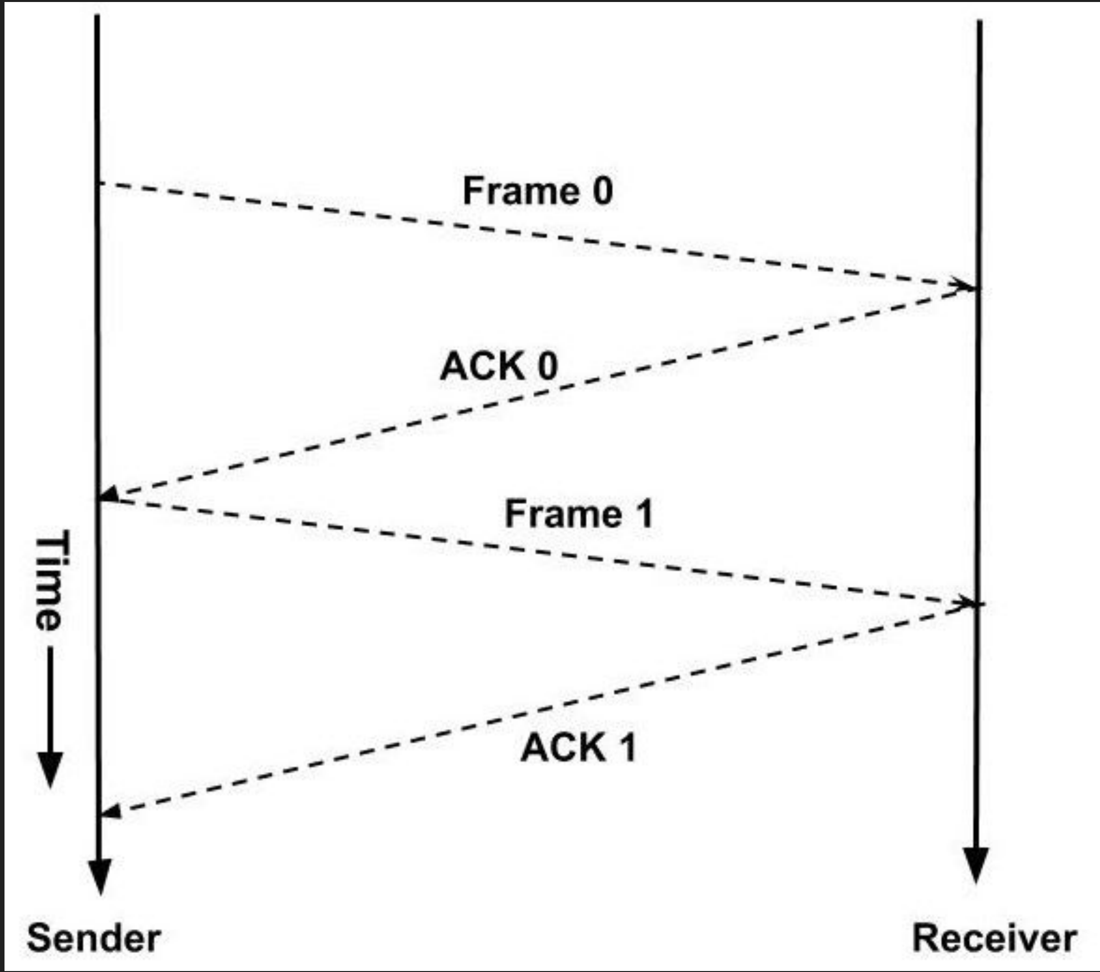
Flow Control (cont.)

- Sometimes hardware has specialized pins for flow control handshaking
  - RTS - Request To Send
    - Printer says “I’m ready to send”
  - CTS - Clear To Send
    - Computer says “I’m ready to read”
- This is hardware flow control

Flow Control (cont.)

- Software flow control
  - Instead of using pins to request or send ready signal, use a data to indicate readiness
- Example:
  - On a parallel interface:
  - The printer sends the ASCII character ‘a’ (x61) to indicate readiness to receive data
  - The computer sends the ASCII character ‘b’ (x62) to indicate readiness to send data

Flow Control (cont.)



Flow Control (cont.)

- Software flow control
  - Instead of using pins to request or send ready signal, use a data to indicate readiness
- Example:
  - On a parallel interface:
  - The printer sends the ASCII character ‘a’ (x61) to indicate readiness to receive data
  - The computer sends the ASCII character ‘b’ (x62) to indicate readiness to send data

Flag Bit

- Sometimes there exists a flag bit inside of a device that gets set HIGH when a “ready” signal is received
- By reading this bit in software, you can then access the data register associated with the flag bit to read the incoming data
- Setting the flag bit can be used in either polling or interrupt

Serial Communications Interface (SCI)

- Sometimes there exists a flag bit inside of a device that gets set HIGH when a “ready” signal is received
- By reading this bit in software, you can then access the data register associated with the flag bit to read the incoming data
- Setting the flag bit can be used in either polling or interrupt

Serial Communications Interface (SCI)

- Asynchronous - no shared clock signal
- Start bit, data bits, parity bit (even/odd optional), stop bit(s)
- Devices must agree to a baud rate
- Used for low data rates and longer transmission distances
- Tx - Transmit Data
- Rx - Receive Data

Serial Communications Interface (SCI) (cont.)

- DTE vs DCE
- DTE - Data Terminal Equipment
  - Tx - Transmit (Output)
  - Rx - Receive (Input)
- DCE - Data Communication Equipment
  - Tx - Transmit (Input)
  - Rx - Receive (Output)

Serial Peripheral Interface (SPI)

- Synchronous - uses shared clock signal
- Uses main and subnode notation
  - Older documentation uses master and slave notation
- Main is the controller and subnode is dependent on main
- Used for high data rate and short transmission distance
- MOSI - Main Out / Subnode In (Data)
- MISO - Main In / Subnode Out (Data)
- CLK - Clock
- CS - Chip Select
  - Used when communicating to multiple devices

I2C (I^2C)

- Synchronous - uses shared clock signal
- SDA - Serial Data
- SCL - Serial Clock
- When using multiple devices, SDA sends address information to communicate only with the desired device

USB

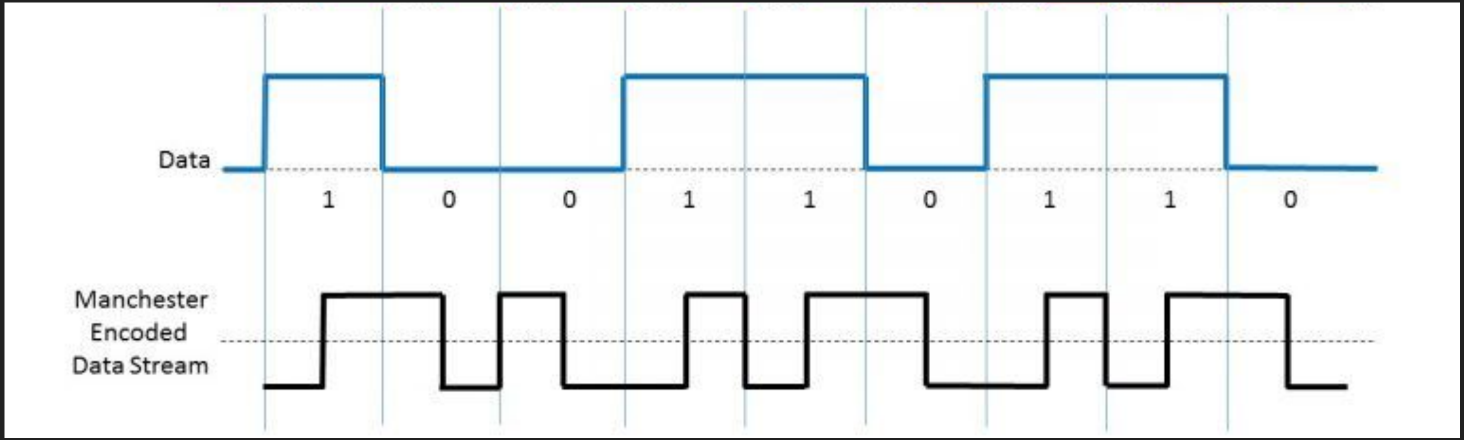
- Synchronous - doesn’t use clock (how?)
- VCC - 5V
- GND - 0V
- Data+ - Positive Data Pin
- Data- - Negative Data Pin
- Differential signaling
  - Data+ - Data-
  - Helps cancel out noise compared to Data and GND
  - Used to achieve fastest transmission

Manchester Coding (Self Clocking)

- USB is synchronous - doesn’t use clock (how?)
- Instead of:

  0 = LOW and 1 = HIGH, use:

  0 = FALLING and 1 = RISING



Manchester Coding (Self Clocking) (cont.)

- Used widely:
  - USB, PCI Express, SATA, Ethernet
- Before PCI-E, PCI was much slower and is a parallel interface and a clock signal
- PCI-E uses multiple serial interfaces all in parallel
  - All the serial interfaces use Manchester Coding
  - More noise resistance = faster speeds
  - Self clocking = multiple devices on bus can operate at different frequencies
    - have a fast graphics card and a slow Bluetooth card on same bus, no slowdowns

Software Serial

- Notice: There is dedicated hardware for communication
  - On the Arduino SCI is dedicated to pins 0 and 1
  - This is why if you communicate to the serial monitor, pins 0 and 1 are not reliable for other devices
- Serial communication can be emulated in software without the need for specialized hardware or using specific pins
- Hardware:
  - Faster: 2M max baud rate (theoretical max)
  - Specific pins and dedicated hardware required
- Software:
  - Slower: 115200 max baud rate (under special circumstances)
  - Any pins, no dedicated hardware
  - Takes up more memory

Software Serial (cont.)

- Sometimes using software serial communication is required
- If you are using hardware serial (for serial monitor), there is no other option
- If you want to use a bluetooth transceiver and serial monitor, you need hardware serial and software serial

Hardware Serial Library Code

- https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/HardwareSerial.cpp
- Serial.write(*byte*) uses a buffer, why?
  - I/O is really slow compared to processor
  - Processor maintains a buffer to prevent data loss
  - Uses polling and interrupts
    - When buffer is full, use polling
    - When buffer is not full, use interrupts

Hardware Serial

- Serial.write(*byte*) uses interrupts if buffer is not full
- When inside an ISR interrupts must wait until ISR finishes
  - Now, Serial.write(*byte*) becomes unreliable *scary*
  - If your design requires hardware serial communication, avoid putting it inside an interrupt

SCI, RS232

- RS232 uses serial communication:
  - Uses a voltage offset: -2.5V to 2.5V
  - This is better for power consumption than: 0V to 5V
  - $P \propto V^2$
  - To carry the same information, TTL consumes 2x more power
    - Avg voltage of 0V to 5V is 2.5V
    - Avg voltage of -2.5V to 2.5V is 0V
  - Before you assume a serial comm is 0V to 5V check the datasheet, it may be -2.5V to 2.5V!