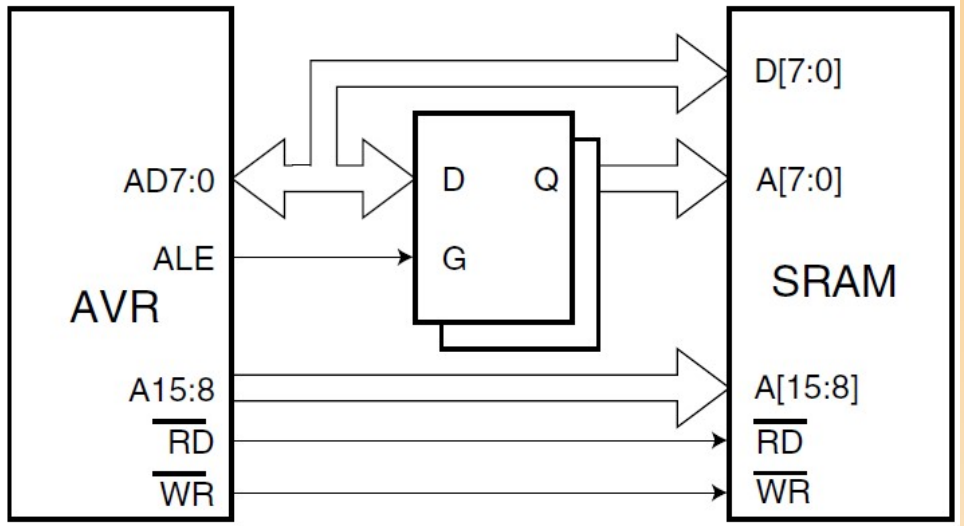
Wired Communications

- Wired Communications allow discrete devices (processors, PCs and other devices) to communicate with each other through hardware connections, for communicating data sets or for synchronization. For this discussion consider wired connections will be covered. Wireless connections are becoming more popular, though, for convenience and should be considered for use when it makes sense.
- Common Types of signals/wires for wired connections:
  - Data signals
  - Clock
  - Control Signals
  - Address signals
- Tri-state signals are common where lines will be driven by multiple devices (e.g. for bi-directional data lines or multiple slaves or masters)
- First fundamental distinction in communications busses is the difference between parallel and serial busses:
  - Serial approach has fewer wires, but may still achieve parallel speeds because parallel alignment is not required.
    - bits interpreted by order in time.
  - Parallel achieves higher throughput for same frequencies by using more wires, but typically cannot run at the same clock speed as a serial bus.
    - bits are interpreted by wire position.
- Parallel busses can be easier to debug by hand if system can be paused and there are a few states to step through. But, dealing with multiple wires can be cumbersome, requires a tester with multiple inputs, and can increase size and cost of system.

RAM parallel interface example

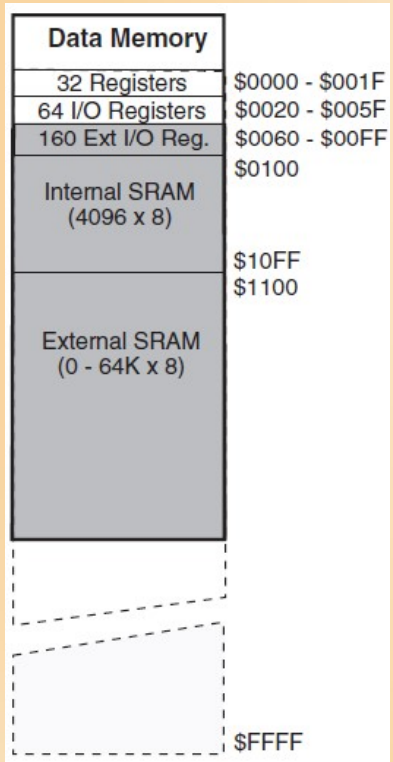
- Memory ICs are one of the most common parallel-interface devices. Some microcontrollers or embedded processes require external RAM, some others support it. Most RAM interfaces are similar.
- If the external RAM interface is enabled, it uses up some of the I/O pins of the microprocessor and enables the use of more memory. The interface uses some of the I/O pins and consists of:
- The interface uses some of the I/O pins and consists of:
  - AD7:0: Multiplexed low-order address bus and data bus.
  - A15:8: High-order address bus (configurable number of bits).
  - ALE: Address latch enable.
  - RD: Read strobe.
  - WR: Write strobe.
- Figure 12; an example from the ATMega128 datasheet:

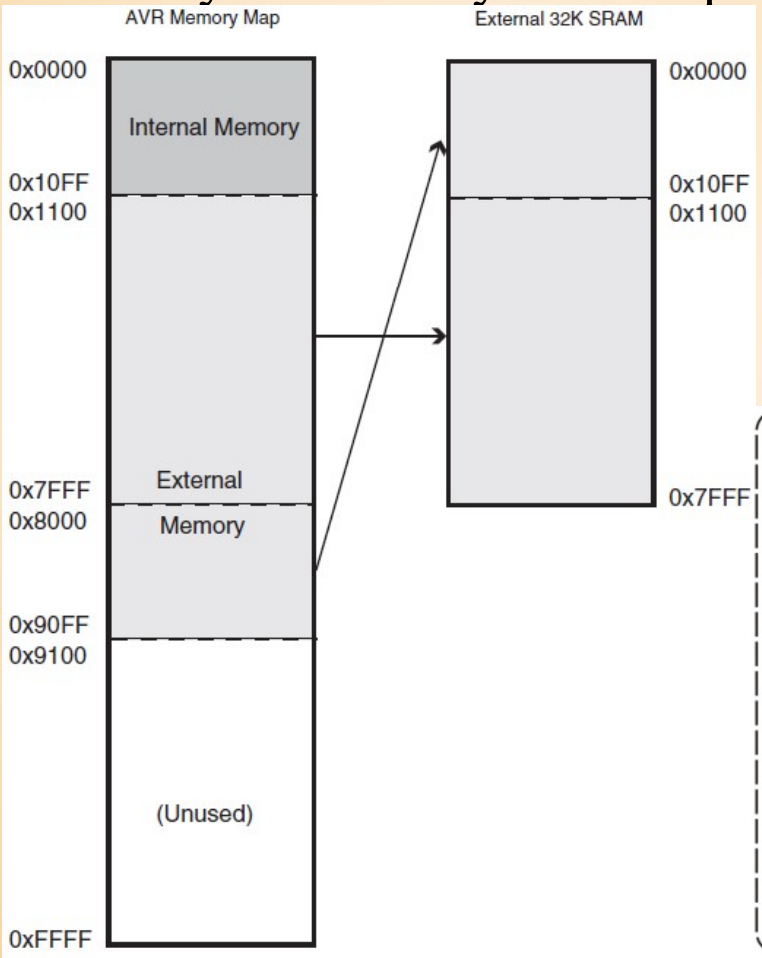
Figure 12. External SRAM Connected to the AVR

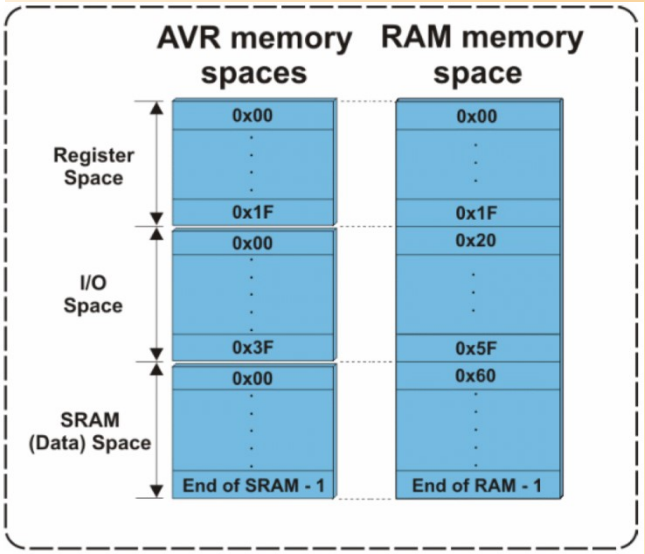


RAM parallel interface example – compiler settings

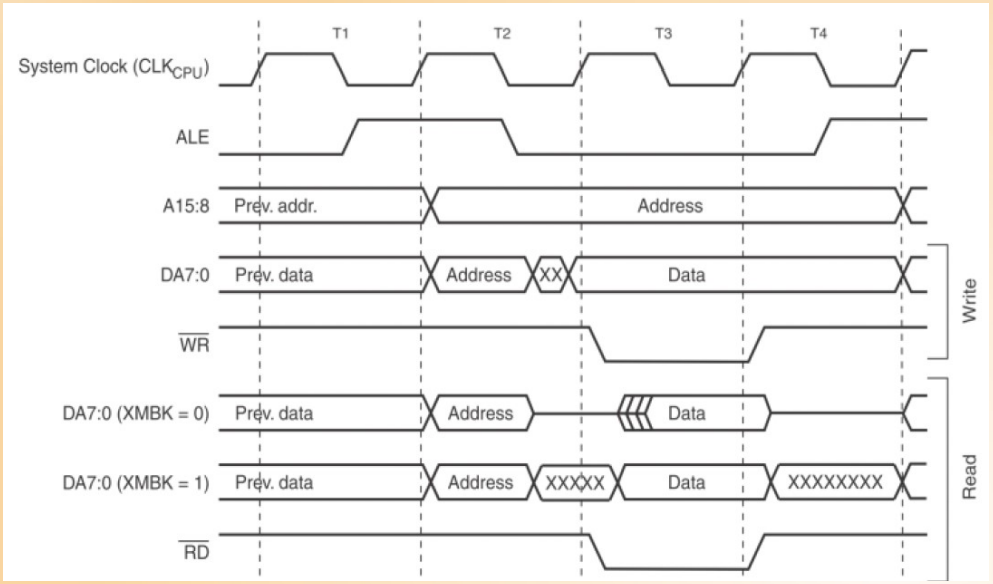
- The compiler settings must be adjusted for it to utilize the RAM.
- Extended RAM becomes available. It is mapped to higher address space and is natively handled by the compiler if it is configured correctly.





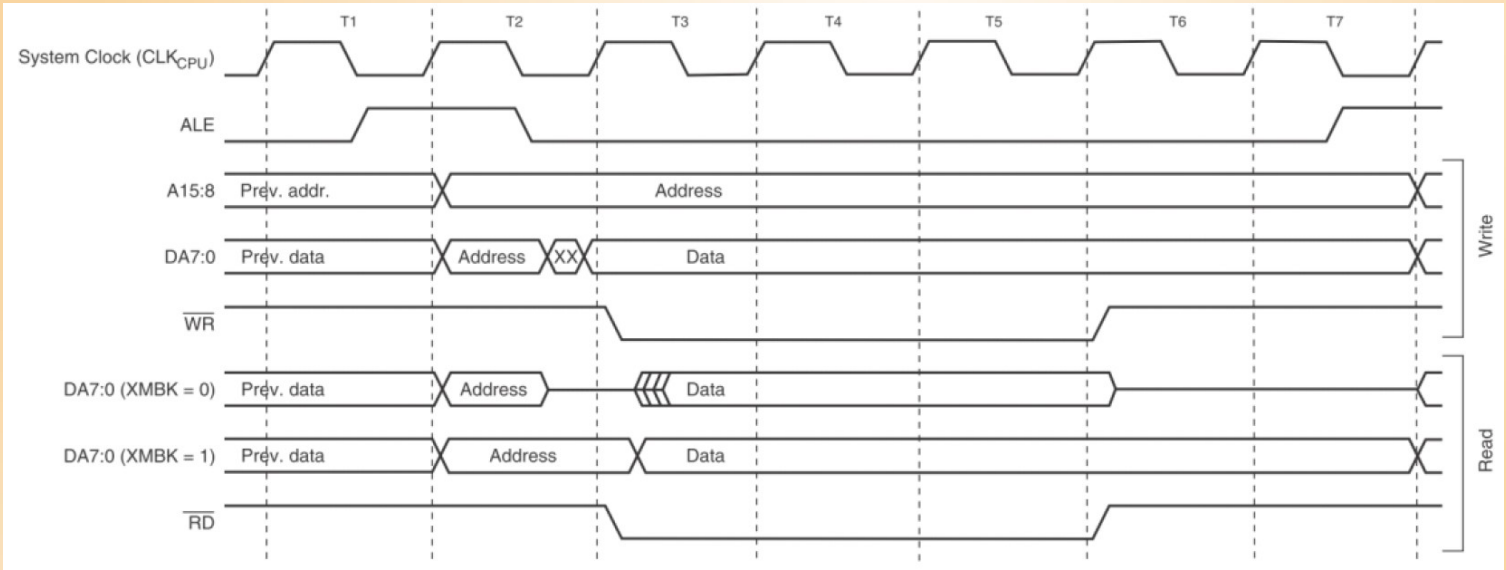


RAM parallel interface example – compiler settings



Before Adding Wait States:

- The memory interface relies on **guaranteed timing** rather than **handshaking signals**. This ATMEL chip supports configuring limited **wait states** to slow the interface to the external RAM; this is a common feature. Other master devices may support much longer wait states.

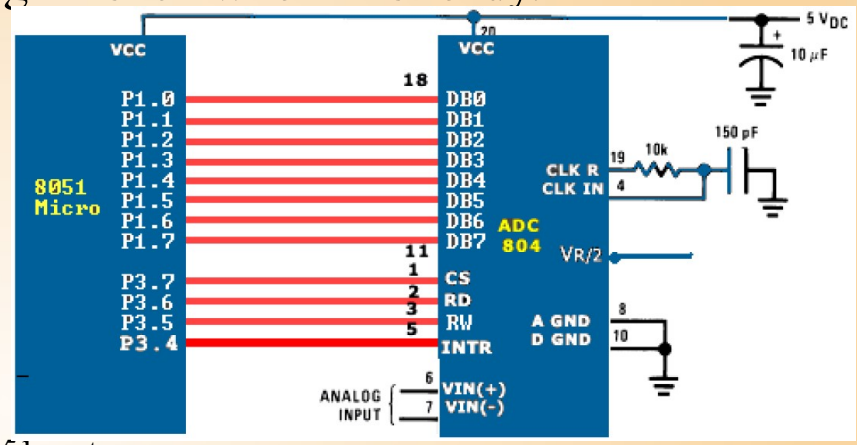


After Adding Wait States:

Handshaking and HardwareControl vs SoftwareControl

- In the previous interface, the microcontroller was the master, the slave device had to operate fast enough to keep up. Other interfaces rely on **handshaking** allowing the slave device to send a signal back when it is ready.

- This author explains how to connect a ADC804 part to a 8051 microcontroller. He describes one option to tap into the memory interface buses, and another to just interface the part with software:

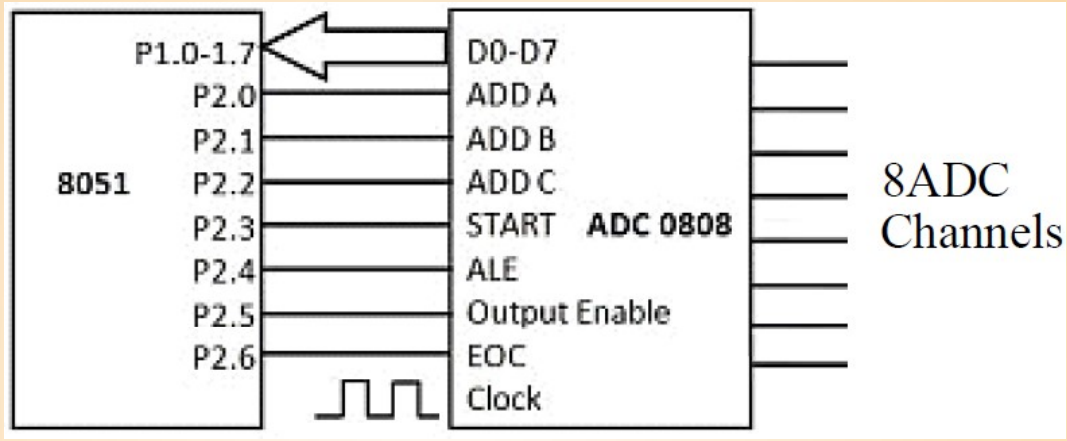


- Memory Mapping (timing is critical)
  - Connect D0-D7 of ADC804 to the data bus of the 8051 system
  - Connect RD, WR of the ADC804 to the 8051 system (ensure polarity)
  - Connect CS of ADC804 to an appropriate address decoder output
  - Connect INTR of ADC804 to an external interrupt Pin on the 8051 (INT0 or INT1)
- IO Mapping (easiest - professor preference)
  - Connect D0-D7, RD, WR, CS, INTR to some port bits on the 8051 (12 in all).
- Algorithm
  - Make CS=0 and send a low-to-high to pin WR to start the conversion.
  - Keep monitoring INTR
    - If INTR =0, the conversion is finished and we can go to the next step.
    - If INTR=1, keep polling until it goes low.
- After INTR=0, we make CS=0 and send a high-to-low pulse to RD to get the data out of the ADC804 chip.
- Pasted from http://www.freewebs.com/maheshwankhede/adcdac.html

- Hardware interfacing uses less CPU resources and is often faster but it is less flexible, so it may not work for the part you have chosen. Software interfacing is very flexible, but it may be slower and uses more CPU resources and may use up extra general I/O pins.

Another Parallel Example: ADC interface:

- When we look at another parallel ADC interface, even with the different names and functions for the signals, we can see similarities emerge. Most parallel interfaces have a lot in common.
- From: http://knol.google.com/k/interfacing-adc-0808-to-8051-microcontroller#



- I/O Pins
  - **ADDRESS LINE A, B, C**: The device contains 8-channels. A particular channel is selected by using the address decoder line. The TABLE 1 shows the input states for address lines to select any channel. (3 bits encode which channel)
  - **Address Latch Enable ALE**: The address is latched on the Low –High transition of ALE.
  - **START**: The ADC's Successive Approximation Register (SAR) is reset on the positive edge i.e. Low-High of the Start Conversion pulse. Whereas the conversion is begun on the falling edge i.e. High – Low of the pulse.
  - **Output Enable**: Whenever data has to be read from the ADC, Output Enable pin has to be pulled high thus enabling the TRI-STATE outputs, allowing data to be read from the data pins D0-D7.
  - **End of Conversion (EOC) (handshaking signal)**: This Pin becomes High when the conversion has ended, so the controller comes to know that the data can now be read from the data pins.
  - **Clock**: External clock pulses are to be given to the ADC; this can be given either from LM 555 in Astable mode or the controller can also be used to give the pulses.
- Reference: http://knol.google.com/k/interfacing-adc-0808-to-8051-microcontroller

Software Algorithm for Interfacing 0808

- ALGORITHM
  1. Start.
  2. Select the channel.
  3. A Low -> High transition on ALE to latch in the address.
  4. A Low -> High transition on Start to reset the ADC's SAR.
  5. A High -> Low transition on ALE.
  6. A High -> Low transition on start to start the conversion.
  7. Wait for End of cycle (EOC) pin to become high.
  8. Make Output Enable pin High.
  9. Take Data from the ADC's output
  10. Make Output Enable pin Low.
  11. Stop
- The total numbers of lines required are:
  - 8 - data lines
  - 1 - ALE
  - 1 - START
  - 1 - EOC
  - 1 - Output Enable
  - 12 --- Total
- The OE pin can be directly connected to Vcc. Moreover instead of polling for EOC just put some delay so instead of 12 lines you will require 10 lines.
- Further, the clock can be provided through the controller thus eliminating the need of external circuit for clock.
- Pasted from http://knol.google.com/k/interfacing-adc-0808-to-8051-microcontroller

Things to look for in the parallel hardware interface

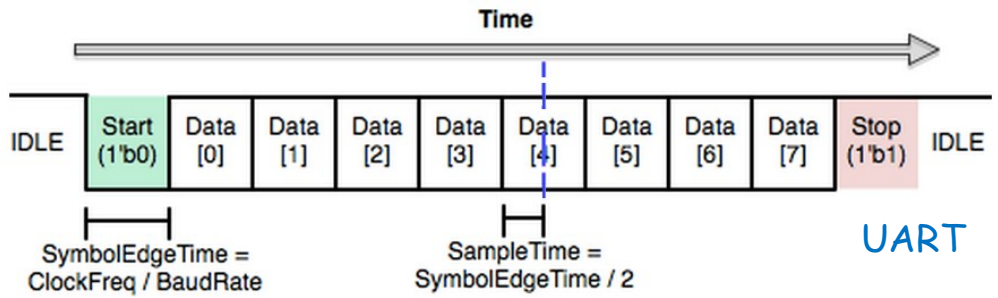
- Does the part have a tri-state or bi-directional bus interface? How is the tri-stating controlled?
  - Tristate interfaces are less trivial to design for
- Does the part have an enable signal or chip select (CS), so that it can share a bus and control lines with other parts?
  - If not, the (micro)controller must generate dedicated control and data signals for each part unless extra hardware for logic-gating can be provided on-board.

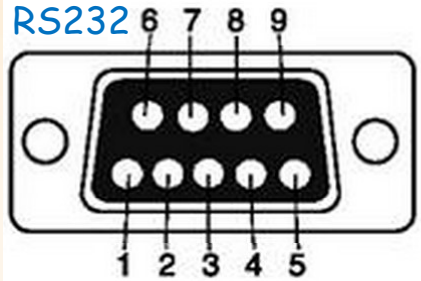
Serial Interface Summary; Common Serial Interfaces

- **RS232:** This is peer-to-peer but typically one device is treated as the master. (You've been using this one.)
- **RS485:** This is supports communications among many devices.
- **USB (Universal Serial Bus):** This is a master-slave interface, it replaced RS232 on desktop computers with much higher speeds.
- **CAN (Controller Area Network):** Multi-point interface that is popular in the automotive industry.
- **SPI (Serial Peripheral Interface):** Relatively simple interface: Single master generates: 1 CS per slave, serial out, serial data in per slave, clock
- **I2C (Inter-Integrated Circuit):** Multi-master communications with only two signal wires
- **Firewire:** -high speed, typically storage or cameras, lower communication overhead than USB
- **Ethernet:** -high speeds and flexible but requires complex special hardware ICs for support and a considerable software level

RS232 UART (infrequently, USART)

- UART (or USART) - Universal (Synchronous) Asynchronous Receiver/Transmitter. This is, essentially, a serial communications interface. The "universal" part means that it can be configured to support many different specific serial protocols. The term is generic, and does not represent a specific standard. At minimum it means that it has a TX and an RX line, which sends a serial data stream and receives a serial data stream.





p#	Pin	Signal	Dir*
1	CD	Carrier Detect	←
2	RXD	Receive Data	←
3	TXD	Transmit Data	→
4	DTR	Data Terminal Read	→
5	GND	System Ground	
6	DSR	Data Set Ready	←
7	RTS	Request To Send	→
8	CTS	Clear To Send	←
9	RI	Ring Indicator	←

* (DTE ←/→ DCE) DTE - Master/Male
DCE - Female/Slave

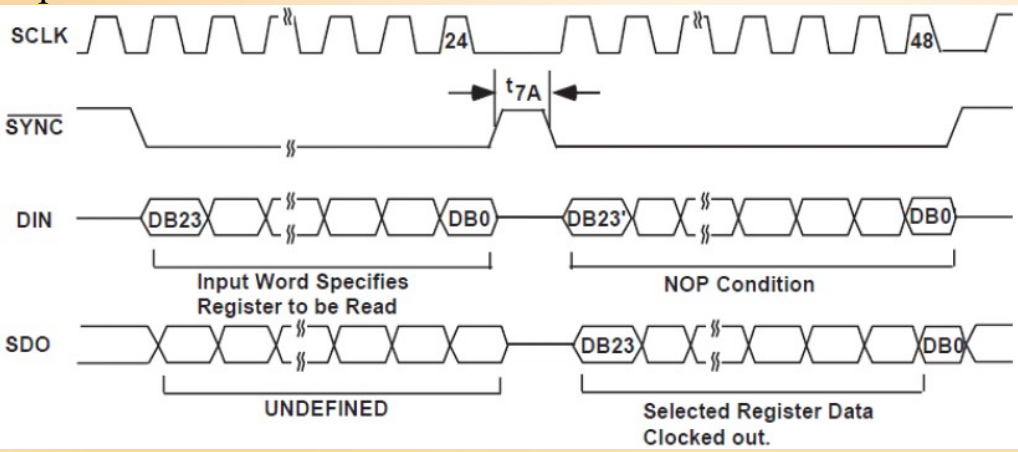
- RS-232 - A standard defining the signals between two devices, defining the signal names, their purpose, voltage levels, connectors and pinouts.
- Very common interface on computers.
- Designed to run tens of feet, next slide compares various voltage levels:
- Often a RS232 level converter IC is required to convert standard logic voltages to these levels.
- RS232 works in a restricted range/set of baud rates: 75, 110, 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 33600, 56000, 115200 and 230400.
- Baud rate, #start and stops bits, parity (and type of parity), #data bits, Flow Control – XON/XOFF is configurable

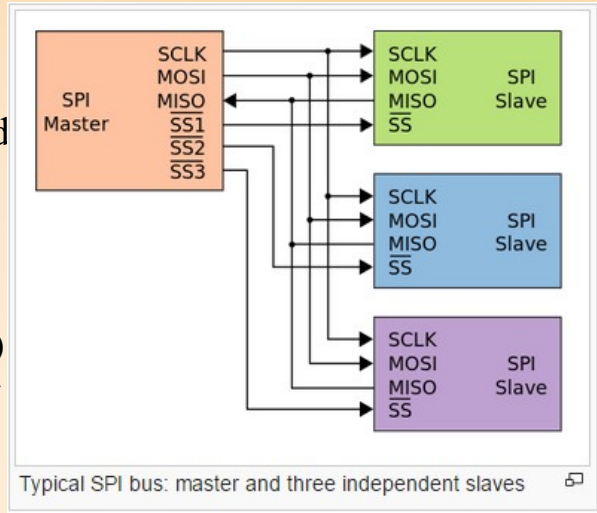
RS232 Handshaking; Additional Signals

- There are two forms of handshaking –software and hardware, to manage sending. Enable the devices to communicate when they are ready for data
  - Software handshaking (also called XON/XOFF) uses control characters in the byte stream to signal the halting and resuming of data transmission. Control-S (ASCII 19) signals the other device to stop sending data. Control-Q (ASCII 17) signals the other device to resume sending data. The disadvantage with this approach is that the response time is slower and two characters in the ASCII character set must be reserved for handshaking use.
  - Hardware handshaking uses additional I/O lines. The most common form of hardware handshaking is to use two additional control wires called RTS (Ready to Send) and CTS (Clear to Send). One line is controlled by each device. The line (either RTS or CTS) is asserted when bytes can be received and unasserted otherwise. These two handshaking lines are used to prevent buffer overruns.
- There are two other less commonly used lines –DTS (Data Terminal Ready) and DSR (Data Set Ready). These lines are typically used by devices signaling to each other that they are powered up and ready to communicate.
- To summarize, RTS/CTS are used for buffer control and DTS/DSR are used for device present and working indicators. In practice, serial communication with no handshaking uses 3 wires (TX, RX and GND). Serial communications with basic hardware handshaking uses 5 wires (TX, RX, RTS, CTS and GND).

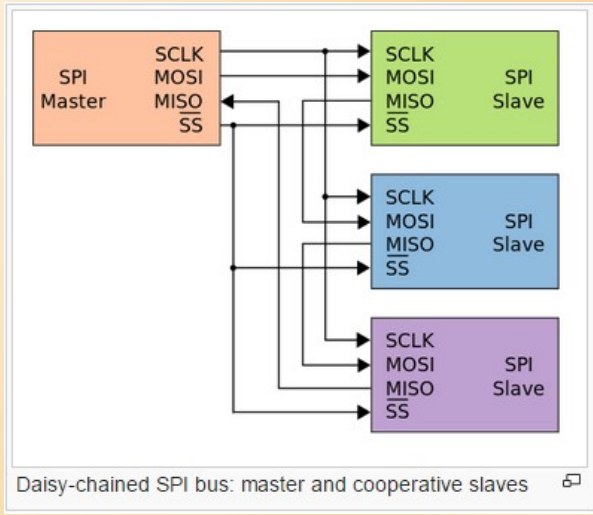
SPI (Serial Peripheral Interface)

- A de facto standard, named by Motorola, that operates in full duplex mode. It is used for short distance, single master communication, for example in embedded systems, sensors, and SD cards.
- The master device initiates the data frame. Multiple slave devices are allowed with individual slave select lines. SPI is often referred to as SSI (Synchronous Serial Interface).
- Four Signals: (or 3+n where n is the number of devices)
  - SCLK - Clock (can be shared)
  - MOSI - Data Out (sometimes master DO can be shared, if multiple Sync(CS) signals are used)
  - MISO - Data In (master requires one data in for every slave data out)
  - SS - Slave Select/SYNC - Frames data, may also serve as CS so data lines can be shared. Master device controls Sync.
- Commonly run at tens of Mbits per second and even faster.
- SPI is not exactly the same on every device, polarities of the sync/select and clock, timing, word length and other factor will vary. Some USB-SPI allow computer programs to interface with SPI parts, obviously these devices require many configurable parameters.





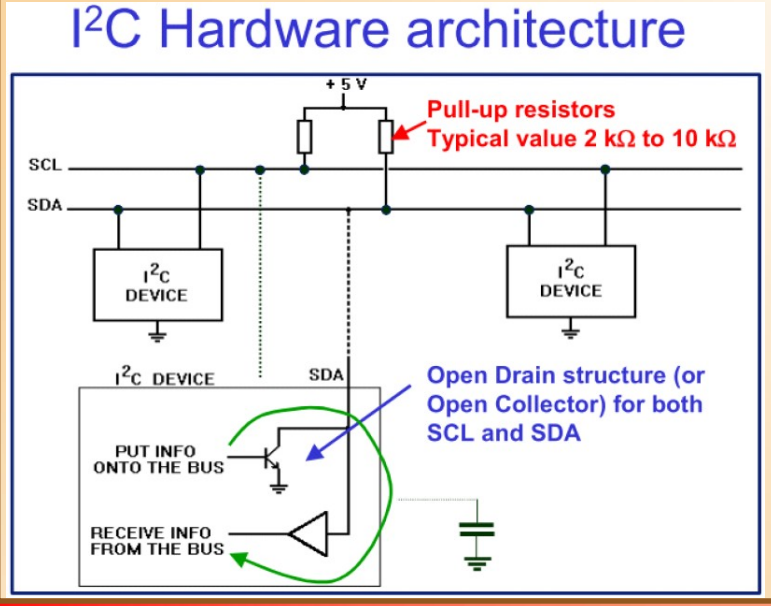
Typical SPI bus: master and three independent slaves

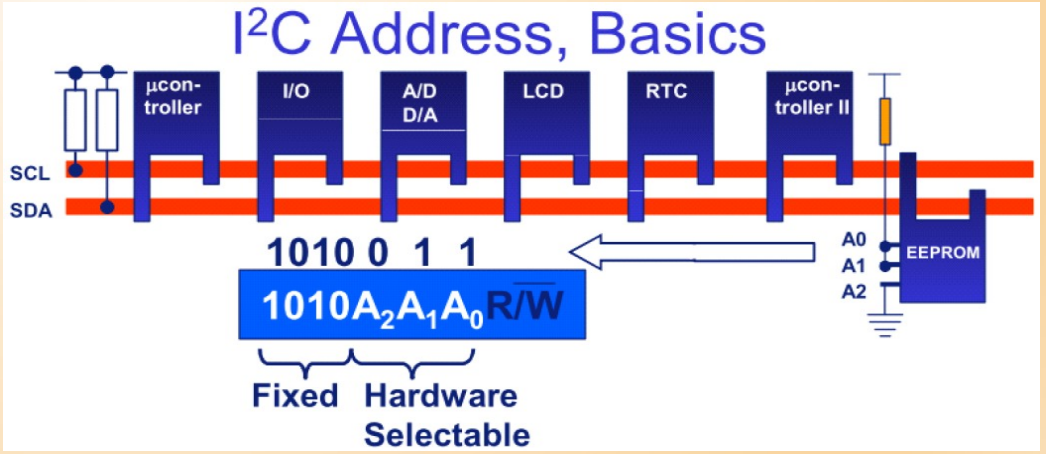


Daisy-chained SPI bus: master and cooperative slaves

I^2C (Inter-Integrated Circuit)

- Also referred to as I2C – "I-two-C" – "I-squared-C"
- Around 400 Kbits/sec originally, but faster versions have been defined (12MHz for version 4.0)
- Intended to serve as universal communication interface. Spec included limit of 400pF load, but does not specify physical connections (plugs and sockets) otherwise.
- Reference: http://www.nxp.com/documents/application_note/AN10216.pdf
- Only two lines:
  - SCL -serial clock
  - SCD -serial data
- Uses "wired and" to allow many devices to share a line. Both clock and data lines have pull-ups. Active devices provide a pull-down signal.





Basic I²C Communication Protocol

1) Wait until no activity on the I2C bus. SDA and SCL are both high. The bus is 'free'.
2) Put a message on the bus that says 'its mine' -I have STARTED to use the bus. All other ICs then LISTEN to the bus data to see whether they might be the one who will be called up (addressed).
3) Provide on the CLOCK (SCL) wire a clock signal. It will be used by all the ICs as the reference time at which each bit of DATA on the data (SDA) wire will be correct (valid) and can be used. The data on the data wire (SDA) must be valid at the time the clock wire (SCL) switches from 'low' to 'high' voltage.
4) Put out in serial form the unique binary 'address' (name) of the IC that it wants to communicate with. Addresses are 7 bits, A[7:3] serve as the category and A[2:0] serve as the identifier in the system.
5) Put a message (one bit) on the bus telling whether it wants to SEND or RECEIVE data from the other chip. (The read/write wire is gone!)
6) Ask the other IC to ACKNOWLEDGE (using one bit) that it recognized its address and is ready to communicate.
7) After the other IC acknowledges all is OK, data can be transferred.
8) The first IC sends or receives as many 8-bit words of data as it wants. After every 8-bit data word the sending IC expects the receiving IC to acknowledge the transfer is going OK.
9) When all the data is finished the first chip must free up the bus and it does that by a special message called 'STOP'. It is just one bit of information transferred by a special 'wiggling' of the SDA/SCL wires of the bus.

Figure 5.1 shows a sequence diagram of data transfer on the I²C bus S - Start condition; P - Stop condition; B - One bit transferred; data changes allowed when clock low (blue) *.

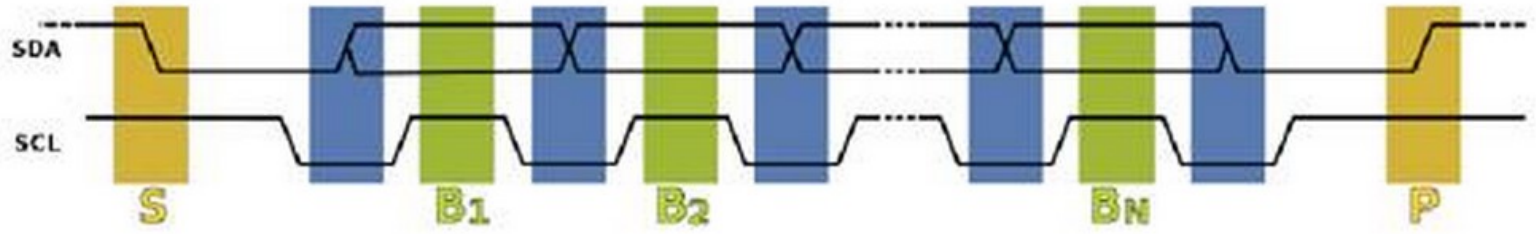


Figure 5.1: I²C Port Timing Diagram Sample

I²C Communication Additional Notes

- Masters are the devices that have the ability to initiate transfers.
  - The start is signaled when the master causes the SDA line to fall while SCL is in the high state.
  - The slave address follows
  - Then a R/W_n bit is sent
  - The slave should then acknowledge pulling the SDA line low (all data sent in I2C is acknowledged)
  - ACK extension allows the receiver to hold the clock line low after an ack to delay the sender from proceeding
  - If master is writing, 8 bits are sent with the clock and on the 9th bit position the slave acknowledges
  - If master is reading, the slave sends 8 bits and the master must send the acknowledge.
  - After one or more send data-cycles or one or more receive cycles, the master rases the SDA line while the SCL line high
  - ...unless it wish to initiate another start to immediately communicate with another device.

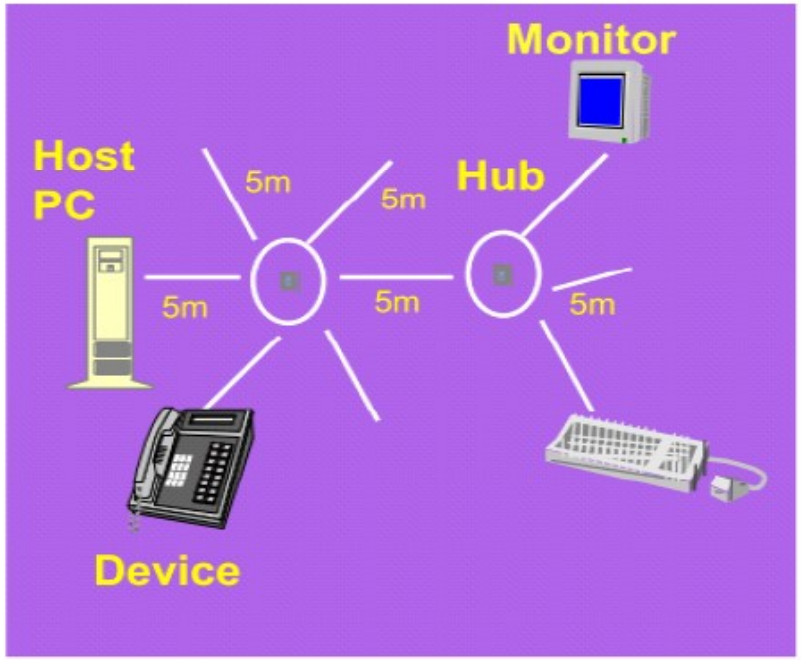
- **Arbitration**-if two masters initiate a transfer arbitration may be needed and it is handled in the following way: if a master is transmitting a one (line is released) and it detects the line is being pulled low, it knows another master is working with the line and it aborts. Notice the master that has the SDA line released is considered to have lost the arbitration.

- **Synchronization**-when a master lowers the SCL line, all other masters follow by lowering their lines. Then each master eventually passes its low period and releases the clock line. Since the master only proceeds after the clock line goes high, that starts the high cycle. In this way, the slowest master can control the speed of the system. (You may note that the low period is controlled by the device with the longest low period and the high time is control by the device with the shortest high internal.)

USB – Topology, Notes

USB Topology (original concept, USB1.1, USB2.0)

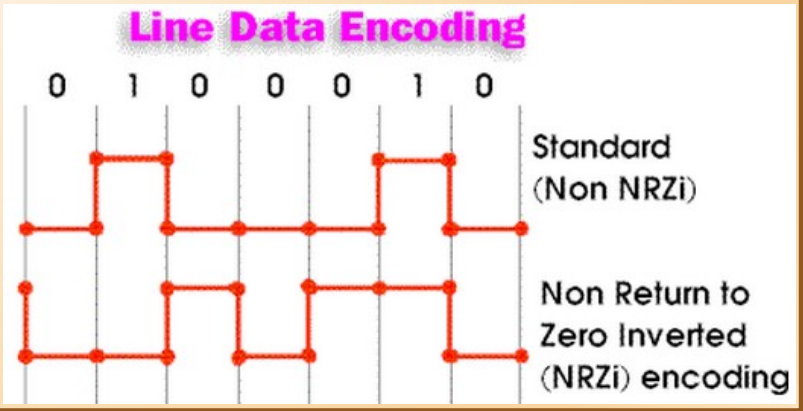
- Host
  - One PC host per system
  - Provides power to peripherals
- Hub
  - Provides ports for connecting more peripheral devices.
  - Provides power, terminations
  - External supply or Bus Powered
- Device, Interfaces and Endpoints
  - Device is a collection of data interface(s)
  - Interface is a collection of endpoints (data channels)
  - Endpoint associated with FIFO(s) - for data I/O interfacing



DesignCon 2003 TecForum I²C Bus Overview 19

USB – Additonal Notes

- Devices are **hot pluggable** (can plug in devices with power on and systems running)
- One host allows several hubs and endpoint devices
- USB assumes guaranteed timing and guaranteed delivery(no packet loss detection). Its communication involves sending packets of synchronization, id, data, crc, and end of packet.
- Three phases are defined for each communication:
  - Phase 1: Token packet phase specifies communication type to take place
  - Phases 2 and 3: Data and Handshake packet phases communicate the data payloads and manage the communication
- Physical Signals:
  - USB uses differential signals(pairs of lines always kept in opposite states), called D+ and D-. The state, 1 or 0, is interpreted from the difference of the two lines rather than referencing the lines to ground. If there is noise on ground or both of the signals, it effectively cancels itself. Differential signaling increases noise margins, prevents radiation, and prevents picking up noise.
  - Furthermore, no clock is used. It relies on the timing of transitions on the signal lines instead. Two things are done to help ensure many transitions
    - Non-Return to Zero Inverted (NRZI) scheme-transition encoded as 0 and non-transitions encoded as 1
    - Bit stuffing -if six ones are encountered, a "dummy" zero is sent



USB – Device Detection

- Detection is accomplished by monitoring the differential data lines after cable power is applied to the port. Figure 18.14 illustrates the device configured for full-speed operation. The channel from the device back to the host has been left off.
- Devices provide pullup resistors to indicate their presence. It is placed on the D+ line to indicate a high-speed 2.0 device and on D-to indicate a low-speed device.

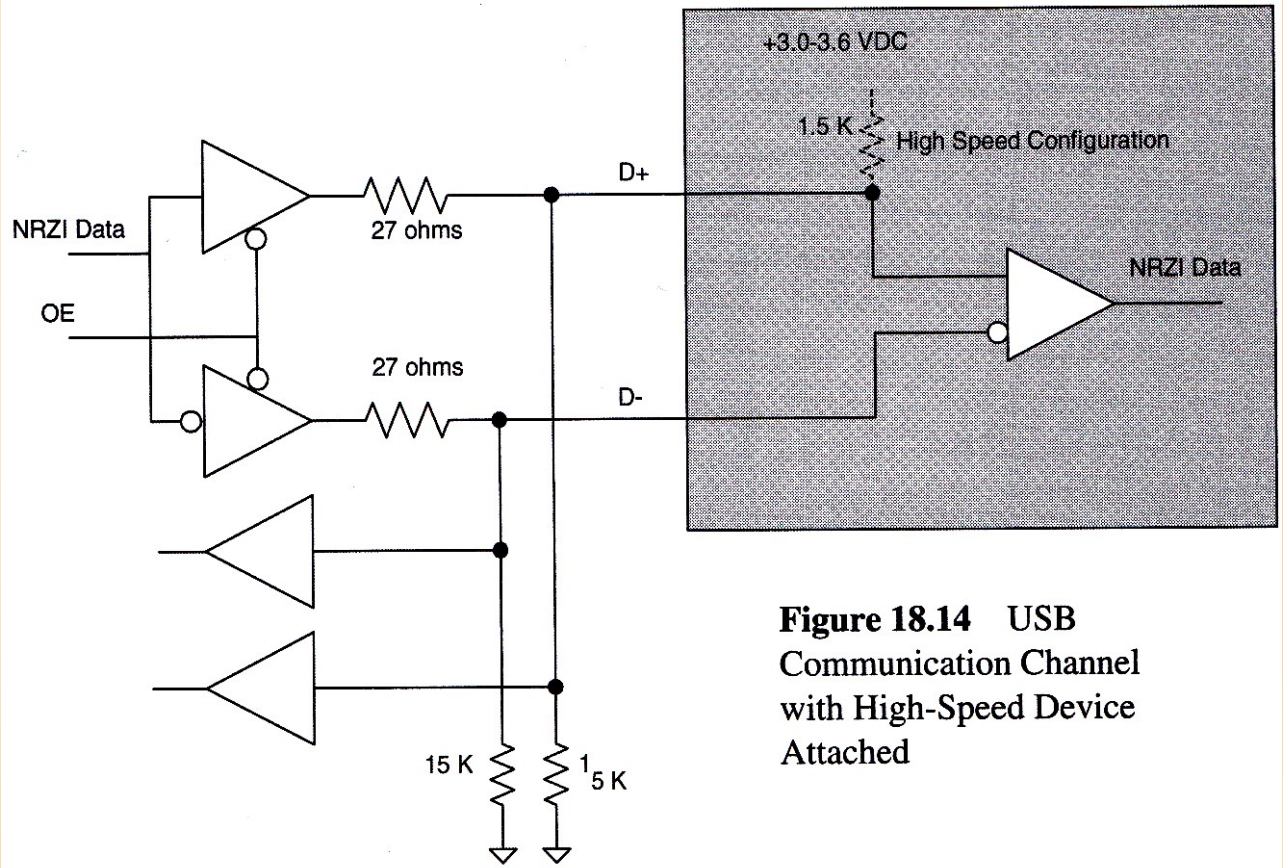


Figure 18.14 USB Communication Channel with High-Speed Device Attached

Applying USB

- Several microcontrollers have built-in USB capability, otherwise you will need to use an external IC.
- If it is an in-built peripheral, template code should be provided to send data and handle incoming data using interrupts or polling. USB communications distinguish between host mode and slave mode. Don't assume both are supported, check.
- FTDI and other companies provide USB-to-serial and USB to parallel ICs . FTDI provides windows drivers to communicate with the ICs. You can use their provided c libraries with the full speed driver or install their virtual com port that allows any program to communicate with the device as if it is a com port.
- USB is Hot pluggable (can plug in devices with power on and systems running)

Communications Summary

- Selected details that can fit on a page:

- **RS232**
  - Host-Slave, point-to-point
  - As little as two signal wires
  - Well-known
  - Simple interface, though transceiver IC often required for voltage conversion
  - Allows 10s of feet distance unless special low-capacitance cables are used
  - Cost effective
  - High-data rates not universally available, typically around 200kbits/sec or 1Mbit/sec
  - Connectors defined in standard
- **CAN**
  - Network connections in automobiles
  - Good Security (encryption)
  - Reliable (high-rate of error detection)
  - Fast
- **USB**
  - One master, multiple hubs and devices
  - 2-wire differential signaling
  - Common, but a few different standards exist
  - Built into microcontrollers, simple usb-to-serial chips available to interface with microcontroller uarts, as well as usb-to-parallel
  - Uses addresses
  - Standards exceeding Gbit/sec
  - Defines supplied power to devices 5V,1/2A (except USB "On The Go" allows 0 and 1.8A)
  - Defines connectors
  - differential-signaling for noise rejection
  - Guaranteed packet delivery
- **SPI**
  - Single Master with multi Slave (use select lines, which uses more pins than others)
  - 4 Signals: (or 3+n where n is the number of devices)
  - Full-Duplex, Fast (no define speed or number of bits)
  - Less defined spec means not all devices work together and no standard error-checking or other elements of a communication protocol
  - Simpler interface circuitry than most others
- **I2C**
  - Multi-master, multi-slave (uses addresses)
  - Minimal wiring interface: two signal wires
  - 100 kb/s standard, 400kbit/s fast, 3.4Mbit/s high-speed
  - 5V interface
  - Well defined standard allows universal operability (though no connectors are defined)
- **FIREWIRE**
  - Peer-to-peer tree network
  - Designed for standalone devices with more capability than some USB devices
  - Allowance for very long cables (100 meters), differential-signaling for noise rejection
  - Some implantations of the connector with power allow 8-30V,1A over-the-line
  - Minimal licensing fee
  - Standards exceeding Gbit/sec, guaranteed bandwidth
  - Simpler interfacing than USB allows better efficiency and less load on CPU than USB
- **ETHERNET**
  - Fast, but requires complex hardware ICs and software APIs