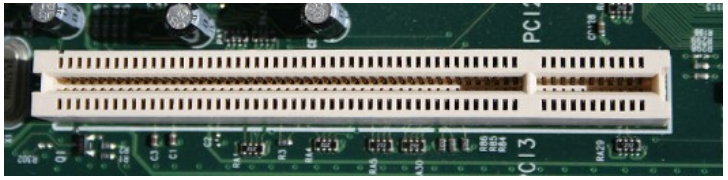
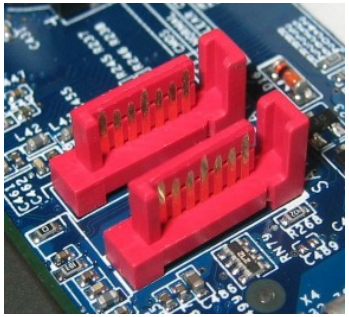
ECSE-426

USART and Buses





Lab Experiment 2 - Introduction to hardware

- Simple memory game
- Uses the UART as a mean of interacting with the MSP430.
  - Mostly used as a display
  - Can be used to send test commands
- Uses timers + Interrupts
- Details on WebCT
- Tutorials are given to get started rapidly with the hardware. Feel free to also use TA help in other periods.
  - Its hard to ensure correct timing of the tutorials...

Note on Hardware Experiments

- Specifications are always a bit vague
  - On purpose – That's usually how they are
  - You need to meet the specs
    - Yet fill the details
    - Make reasonable assumptions
- Importance on robustness
  - It has to work during the demo
  - Start with a solid core and add fluff later
    - Optimize your work...
    - You have other courses too.

Today's lecture

- UART
  - What is it ? historical perspective
  - Principle of operation
  - MSP430 USART module
- BUS
  - Principles
  - Discussion on common buses

Asynchronous Transmission

- Early attempts at serial transmission
  - Smoke signals, Morse code
- Used for traditional (CRT) terminals through serial interfaces
  - Also used to connect with modems
- Transmits characters at bit rate asynchronous to the other device
  - Need for start bit and stop bit(s) - synchronization
- Timing for each character based on selected parameters of USART
  - Transmit and receive functions use same baud frequency
  - 7 or 8 bit data with odd, even or non-parity
- Implementation: independent transmit/receive shift and buffer registers

UART Character Format

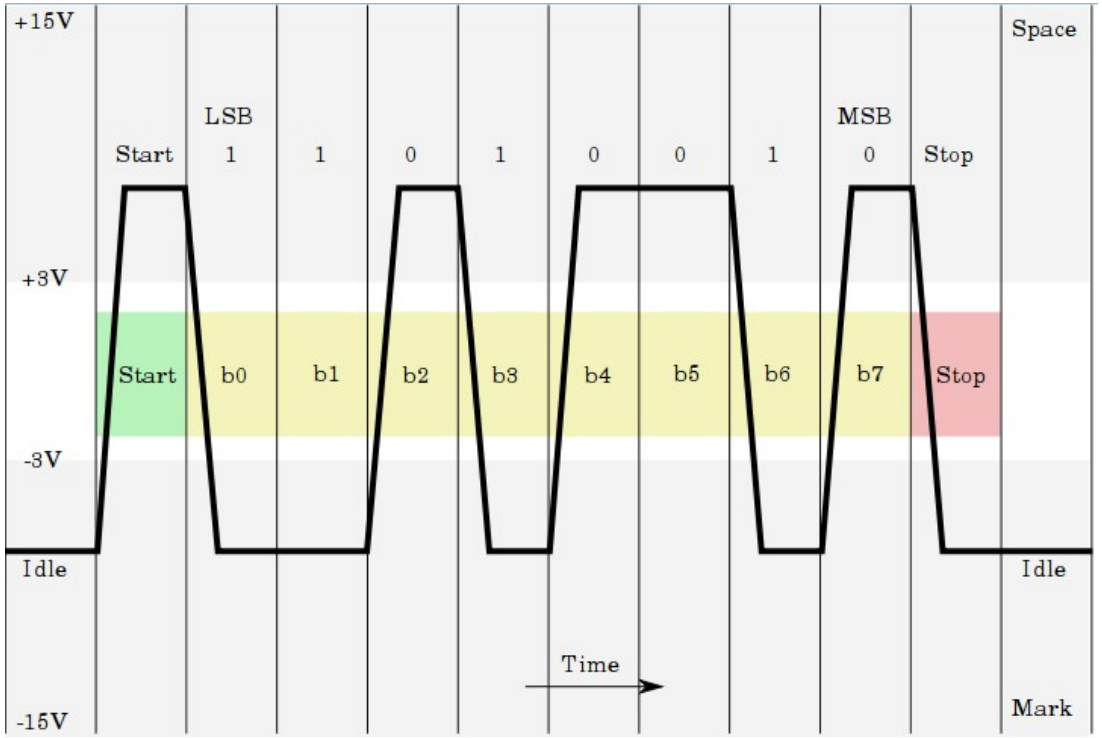
- Character format on the serial line:

- ST: start bit
- D0-D6 (D7) : 7 or 8 data bits ; D0 is least significant bit
- PA : Parity - parity bit, odd, even or no parity
- STP : Stop bit - one or two stop bits.
- 9600-N-8-1
  - 9600 bits per second, No parity, 8 bits of data, 1 stop bit

Physical Interface

- Electronics Industries Association (EIA) – RS-232
  - 1969 (!)
  - Defines the interface
    - Mechanical Interface – Shape, size of pins
    - Electrical levels – Capacitance, slew rate, etc.
    - Connector Pinout
  - Does not define the data format
    - Character encoding, data frames format, etc. are defined by the higher level protocol.

Physical Interface



Note the relatively high voltages (nominally -11V to 11V)

When converted digitally, the Mark = +3.3V, Space = 0V
=> Idle is logical '1'

Source : http://en.wikipedia.org/wiki/File:Rs232_oscilloscope_trace.svg

Why study UARTs ?

- Very wide utilization of simple, slow serial links
  - Machine-to-machine communication
  - Industrial computer communication
  - Cellular & Satellite Modems + GPS
    - Satellite modems have a low data rate (a few kbps)
- Very simple interface to learn
  - Compared that to USB or Firewire...
  - You can “see” the data with an oscilloscope
    - Send the letter 'A' (0x41) on the UART, you will see:
      - Start bit (0) + “10000010” + Stop bit (1) (digital levels)
    - Try it on your McGumps board...

USART Communication

- MSP430 Specific Details
- USART Receive Enable (URXEx)
  - Enable reception on URXDx (Module enable)
  - Receive data buffer UxRXBUF contains the character moved from RX shift register after reception is complete.
- USART Transmit Enable (UTXEx)
  - Initiate transmission by writing to UxTXBUF
  - Data moved to TX shift register when it is emptied.

UART Baud Rate Generator

- Produce standard baud rates from non-standard clock frequencies
- Choose the clock source BRCLK
- N = BRCLK/baud rate
- Now write a 16-bit value UxBR into the registers BCR0 and BCR1. This is floor(N)
- The modulator is used to match the fractional portion (an iterative process to minimize timing error)
- Values available for standard rates

USART Initialization

- Set SWRST (software reset)
  - BIS.B #SWRST, &UxCTL
- Initialize all USART registers keeping SWRST = 1 (including UxCTL).
- Enable USART module via MEx SFRs (URXEx or UTXEx): Module Enable Special Function Registers.
- Clear SWRST
  - BIC.B #SWRST, &UxCTL
- Enable Interrupts via the IEx SFRs (URXIEx or UTXIEx). Interrupt Enable.

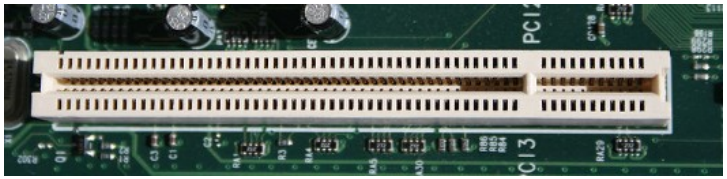
USART Errors

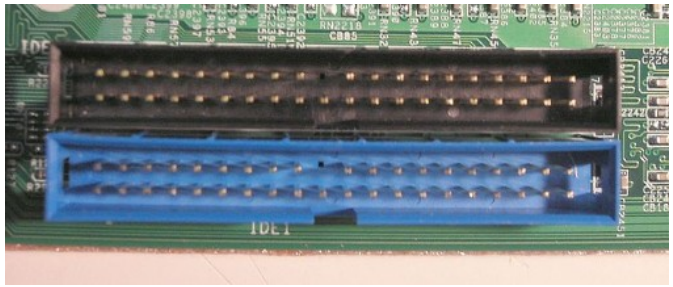
- Framing error (FE bit)
  - Low stop bit detected
- Parity error (PE bit)
  - Mismatch between number of 1's and parity bit
- Receive overrun error (OE bit)
  - Character loaded into RXBUF before previous character is read.
- Break Condition (BRK bit or interrupt flag)
  - 10 or more low bits after missing stop bit.
- Set any of these -> also set RXERR bit.

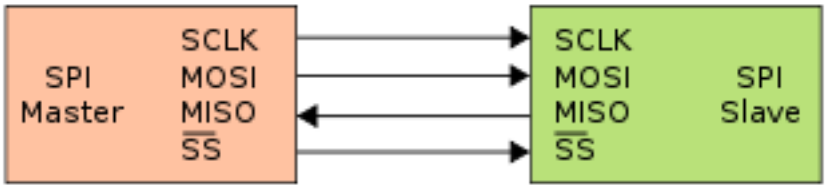
USART Interrupts

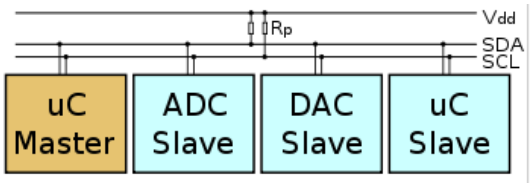
- UTXIFGx Transmit interrupt flag
  - Set by transmitter to indicate that UxTXBUF is ready for another character
  - Interrupt request if UTXIEx and GIE are set.
  - Automatically reset if interrupt is serviced or if another character is written to UxTXBUF
- URXIFGx Receive interrupt flag
  - Set each time character received and loaded into UxRXBUF
  - Interrupt request if URXIEx and GIE are set.
  - Reset if interrupt is serviced or if the character is read from UxRXBUF

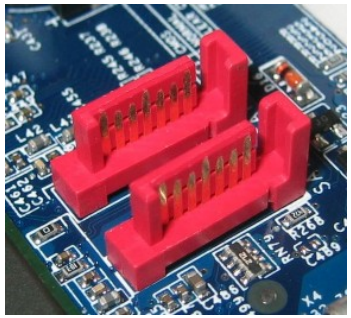
Computer Buses

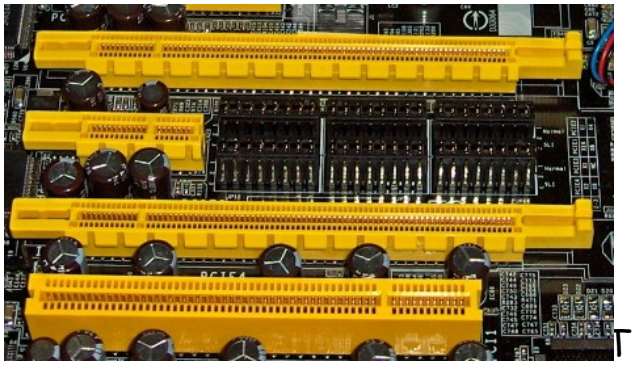












Source : Wikipedia

Bus

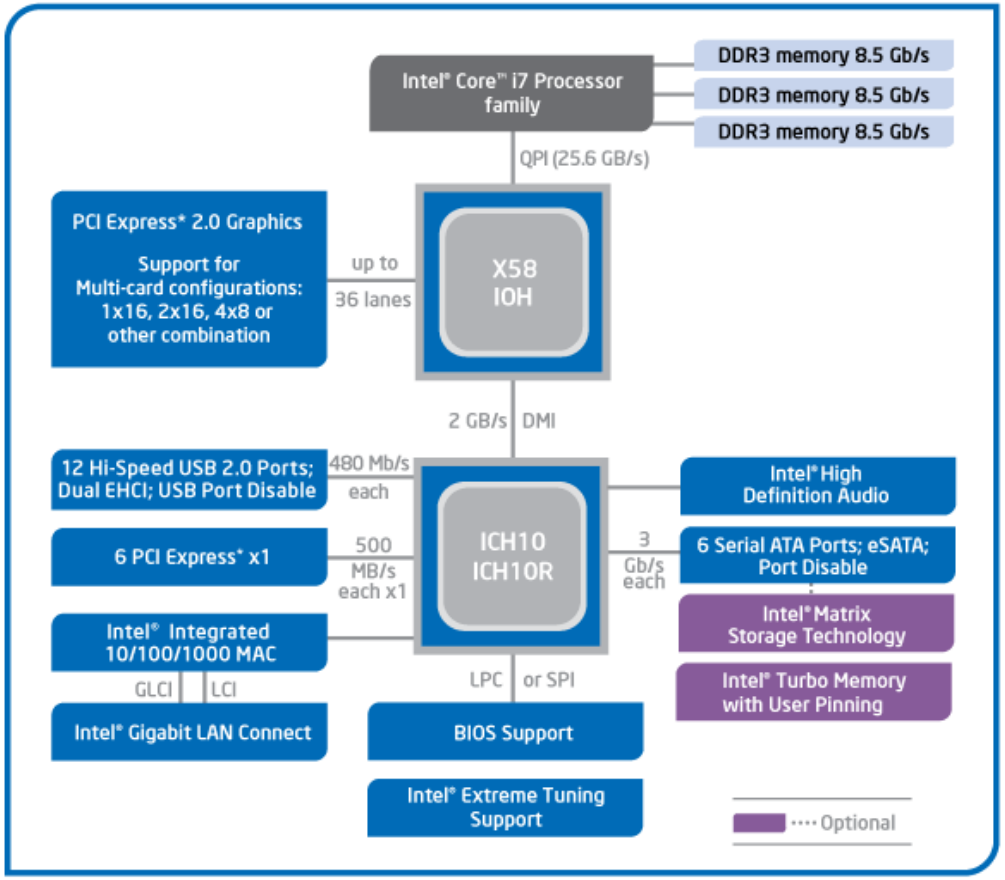
- Common electrical pathway between multiple devices
- Can be internal to CPU (e.g., data to ALU) or external (to memory and I/O devices).
- Modern computers:
  - Special-purpose bus(es) between CPU & memory
  - Other bus(es) for I/O devices
- Protocols govern operation (e.g., PCI, SCSI, ISA).
- Scalable Interconnect: Infiniband

System Buses/Backplanes

- Systematic way to create extendible and open hardware systems
  - Also: to reduce prices, reuse designs, reach market
- Standardization: the key
  - Industry associations (USB, PCI)
  - Standardization bodies (IEEE488, Firewire)
- Standard content:
  - Physical, Mechanical, Electrical and Logical
- Example: PC Platform
  - PCI, PCIe, ISA, USB, RS232, SCSI, IDE, SATA, Ethernet, Rambus, Infiniband, Hypertransport, AGP

Multiple Buses: PC platform

- Modern Processor (Core i7)
  - Integrated Memory Controller
  - DDR3 Memory Bus
- QPI (QuickPath Interconnect)
  - Allows Multi-processor (NUMA)
  - Similar to AMD HyperTransport
- IO Hub (X58)
  - Direct Media Interface (DMI) to SouthBridge
- ICH10 (SouthBridge)
  - Slower Peripherals



Intel® X58 Express Chipset Block Diagram

Source : Intel web site

Bus Principles

- Masters: active devices that can initiate bus transfers)
- Slaves: passive devices that wait for requests
- Most masters produce binary signals too weak to power a bus.
  - Especially on long links
  - Connected to bus via a bus driver, essentially an amplifier.
- Slaves connected via a bus receiver.

Bus principles (cont'd)

- For devices that can be masters or slaves, there are bus transceivers.
- Often tri-state devices, allowing them to float (disconnect) when not using the bus.
- Address, data and control lines.
- Design issues: bus width, bus clocking, bus arbitration, and bus operations.
- Each has substantial impact on speed and bandwidth.

Bus Width

- Wide buses need more wires, more physical space, bigger connectors.
- Hence width is expensive, and there is a trade-off between maximum memory size and system cost.
- Short-sighted designs:
  - 8088 – 20-bit address bus (1 MB)
  - 80286 – add 4 more bus lines (16 MB) + extra control
  - 80386 – add 8 more. (E-ISA) much messier!

Bus Width (cont'd)

- Data lines also tend to grow over time.
- Two ways to increase bandwidth: decrease cycle time or increase width.
- Speeding bus up is possible but leads to problems with bus skew.
- IBM PC data lines: 8->16->32->64
- Multiplexed bus: use same lines for data and address (address first, data later).
- Wide buses introduce bit skew problems
  - Must keep trace length balanced => Difficult!
  - Longer Buses => Slower due to handshakes

Parallel Bus clocking

- Synchronous
  - Line driven by crystal oscillator (square wave)
  - Example: 40 MHz, 40 nsec memory read, 1 nsec for signal change.
  - T1 starts on rising edge of clock
  - CPU places address on bus lines
  - Asserts MREQ (memory being accessed) and RD.
  - T2 is a wait state (memory asserts WAIT)
  - T3: memory places data on DATA lines
  - At T3 falling edge, CPU strobes DATA and latches; releases MREQ and RD.

Parallel Bus Clocking

- Synchronous
- One clock goes to all the elements
- Every element "see" the same clock at the same time
  - Clock distribution is important
  - Data is "launched" on rising edge of clock
  - Captured on next edge by the target
    - Takes time to exit the chip (Clock to Out, Tco)
    - Propagation Delay (Tp) due to limited speed
      - About 60% of speed of light on printed boards
      - Limited rise time due to capacitance
    - Hold time (Th)
      - How long the data must be maintained on the input of target before the clock.

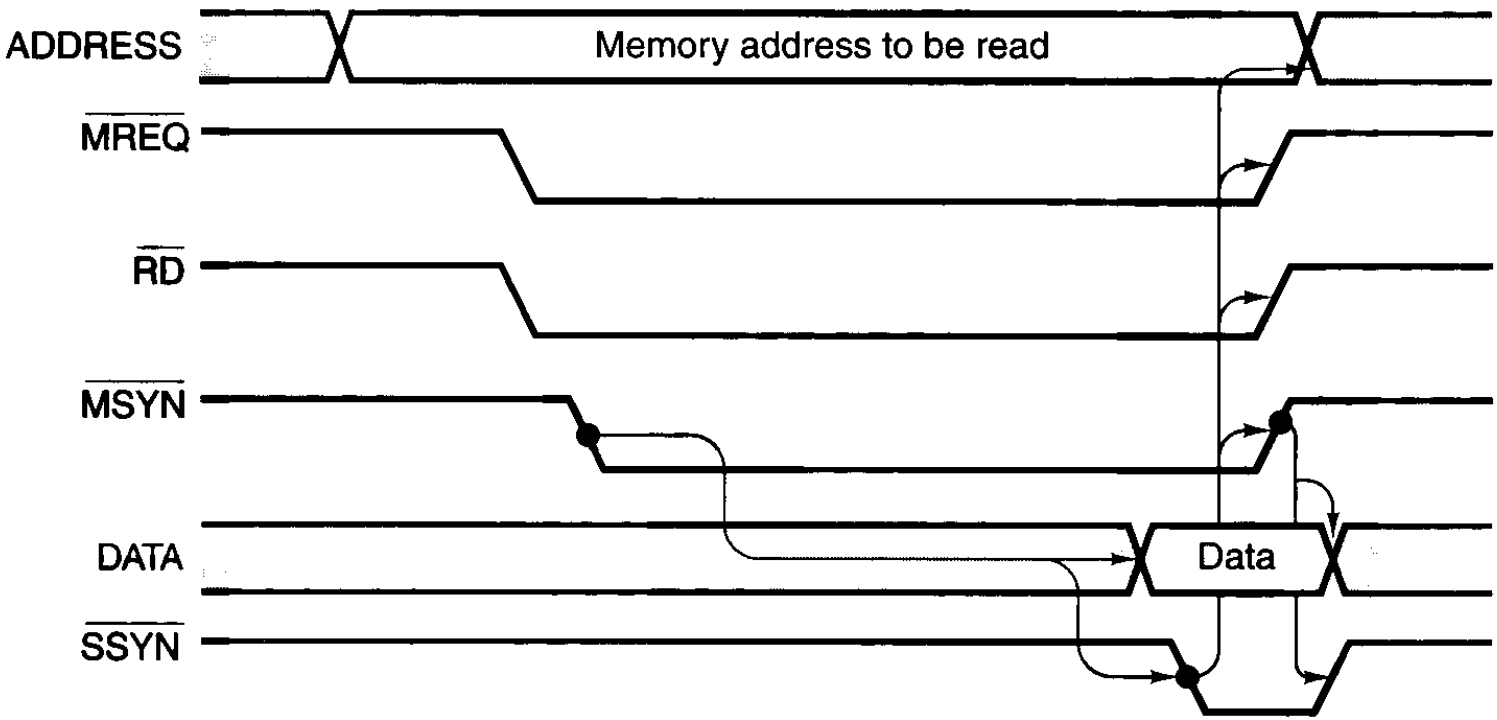
Parallel Bus Clocking (cont.)

- The previous timing apply to each and every signal lane
  - Many lines, many constraints to meet
    - Meet timing for the slowest line of the whole bus
  - Many target devices
    - Higher Capacitance + longer bus = Slower system
    - Slow device slows everybody else down
- Workarounds
  - Handshakes (send command, wait for ack)
  - Multi cycle paths (multiple clocks given to the target)
    - If Tcycle is 20ns, timing is multiple of 20ns

Asynchronous Buses

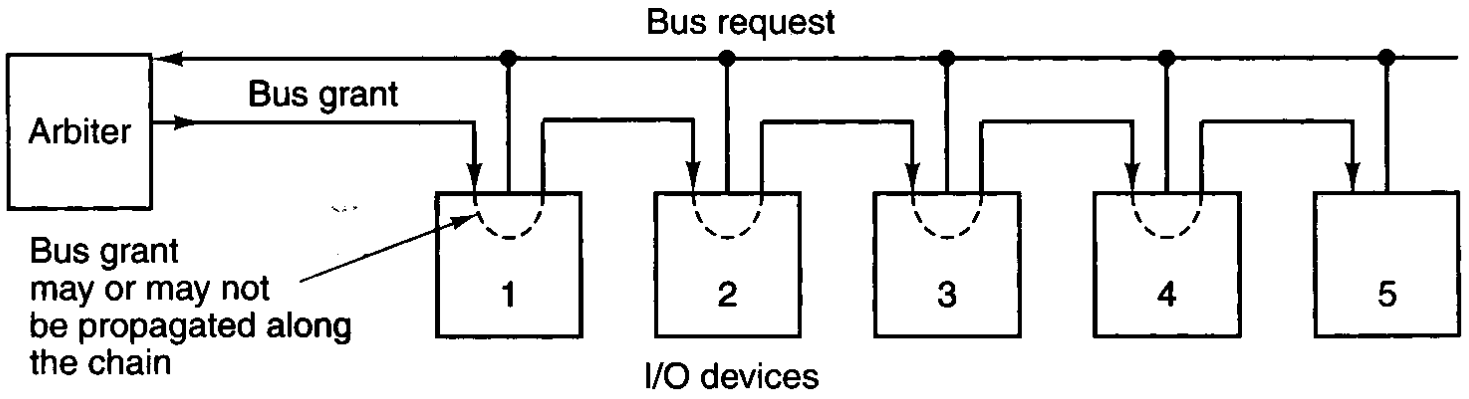
- Synchronous buses
  - waste time (no fractional cycles)
  - cannot take advantage of tech. improvements
- Asynchronous
  - No master clock
  - Bus master asserts address, MREQ, RD; asserts special signal MSYN (Master Synch.)
  - Slave then performs work as fast as it can, then asserts SSYN (Slave Synch.)
  - Master latches data, negates MREQ, RD, address, negates MSYN.
  - Slave negates SSYN.
  - Full handshake.

Asynchronous handshake



Bus Arbitration

- Single central bus arbiter
  - Devices assert bus request line (wired-OR)
  - Arbiter issues a grant by asserting bus-grant line, wired through all devices.
  - Daisy-chaining: device physically closest gets grant if it made request.
  - Priority according to physical location.
  - Alternatively -> multiple request/grant lines.

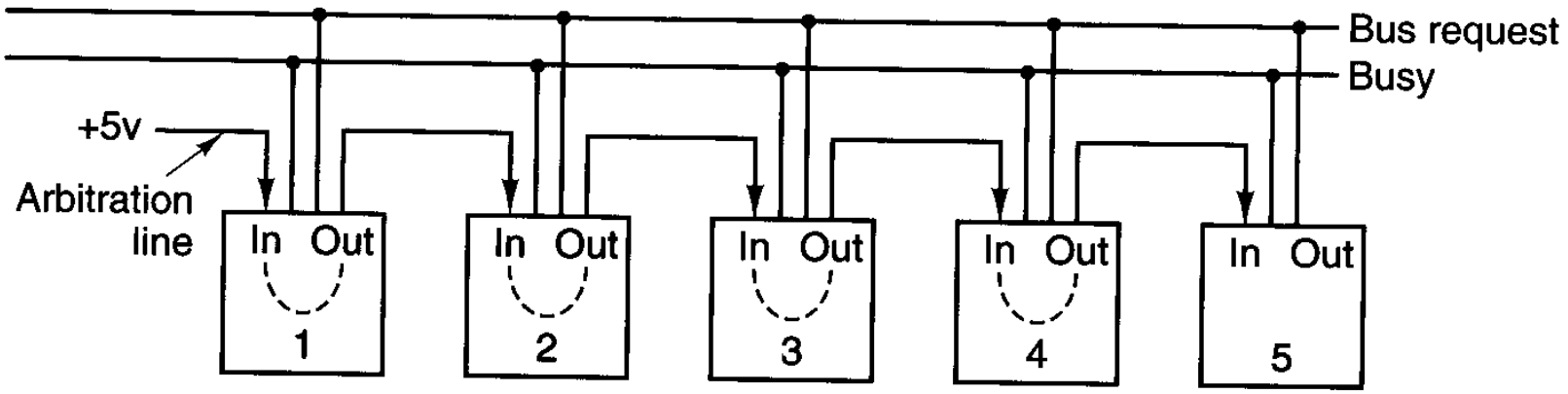


Decentralized Bus Arbitration (1)

- Example 1:
  - Each device has individual request line.
  - All devices monitor all request lines and can determine if it had highest priority.
- Example 2:
  - Three lines: wired-OR request, BUSY (asserted by current master), and arbitration line
  - Acquire: check to see if bus is idle and IN is asserted.
  - If IN is negated, then it negates OUT.

Decentralized bus arbitration (2)

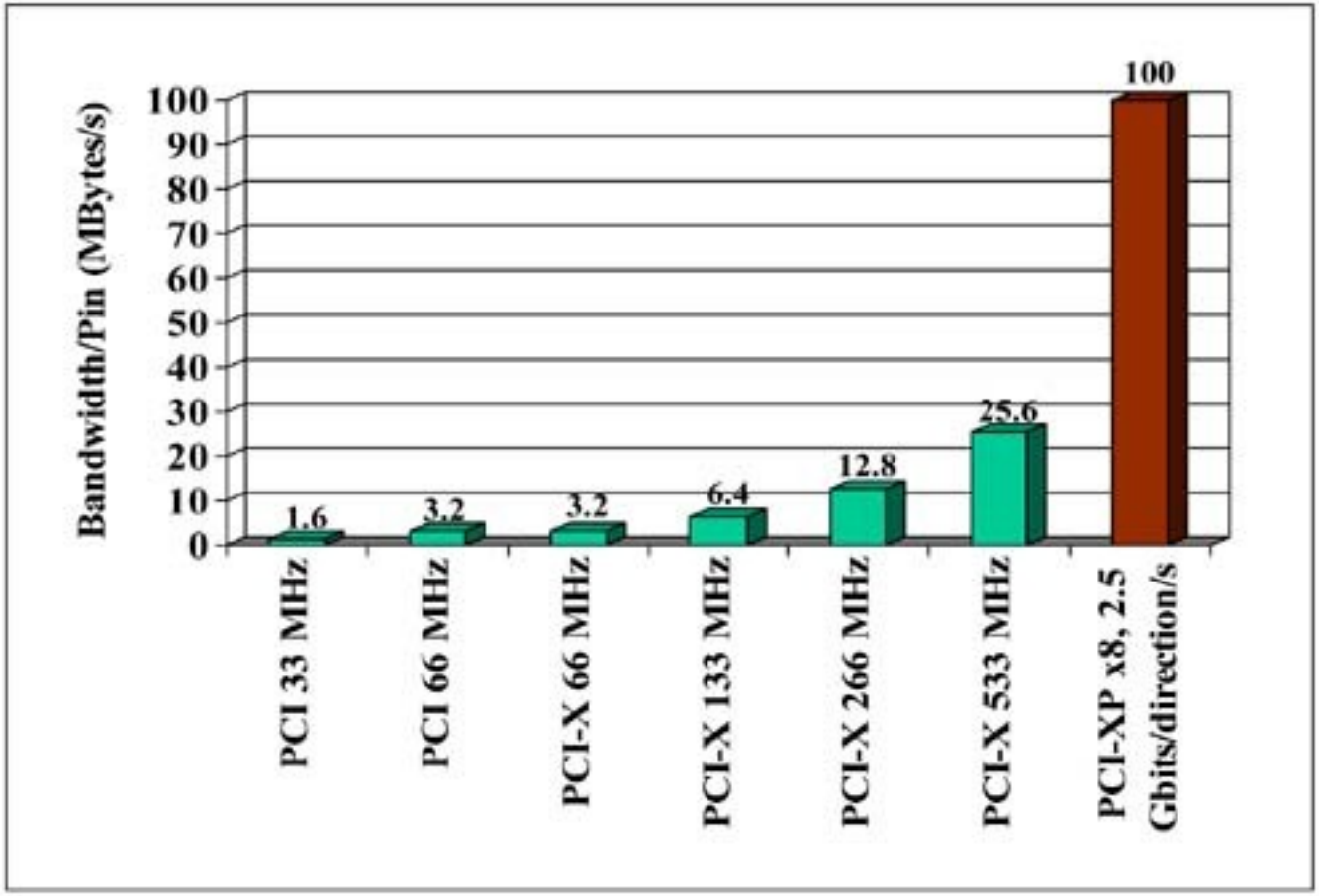
- Example 2 (Bus acquisition):
  - Check to see if bus is idle and IN asserted.
  - If IN is negated then it can't become master, and it negates OUT.
  - If IN asserted: negate OUT -> deprive downstream
  - At end of cycle, only one device has IN asserted and OUT negated.



Reducing Bus Width – Serial I/O

- Recent advances are substantial
  - PCIe 1.1 => 2.5 Gbps per pair
  - PCIe 2.0 => 5.0 Gbps per pair
  - Future PCIe 3.0 => 8 Gbps per pair (yet twice the performance due to change in coding)
- High Speed Serial Interfaces
  - A few differential signal pairs running at very high data rates
  - Replaces a wide bus by a few signal pairs
- Clock and data on the same pair
- Protocols similar to networking
  - Flow control, classes of traffic
- Similar trend with Parallel ATA to SATA
  - Sata 1.5 Gbps, 3.0 Gbps, 6.0 Gbps
    - Increased bandwidth to use of latest solid state disks
- USB 1.1 (1.5 Mbps / 12 Mbps), USB 2.0 (480 Mbps)

Serial Buses – Bandwidth per pin



Source : Budruk & Al., *PCI Express System Architecture*

Serial Buses – Clock and Data

- Signals are sent differentially to increase immunity to noise
- Clock is embedded with data
  - Needs encoding to avoid long sequences of '0' or '1'
    - Those will mess up clock recovery
  - 8B10B encoding is a well used code
    - Developed by IBM in 1983
    - Take 8-bit data => transform into a 10-bit symbol
    - Keep track of number of zeros and ones and balance numbers on average to keep DC balance.
    - 8B10B used in PCIe, Firewire, SATA, GigE, DVI/HDMI, USB 3.0.
- Clock is recovered at the receiver
  - Need a phased-locked loop (PLL) to achieve this
  - Uses the transitions of bits to keep synchronization

Serial Buses (contd.)

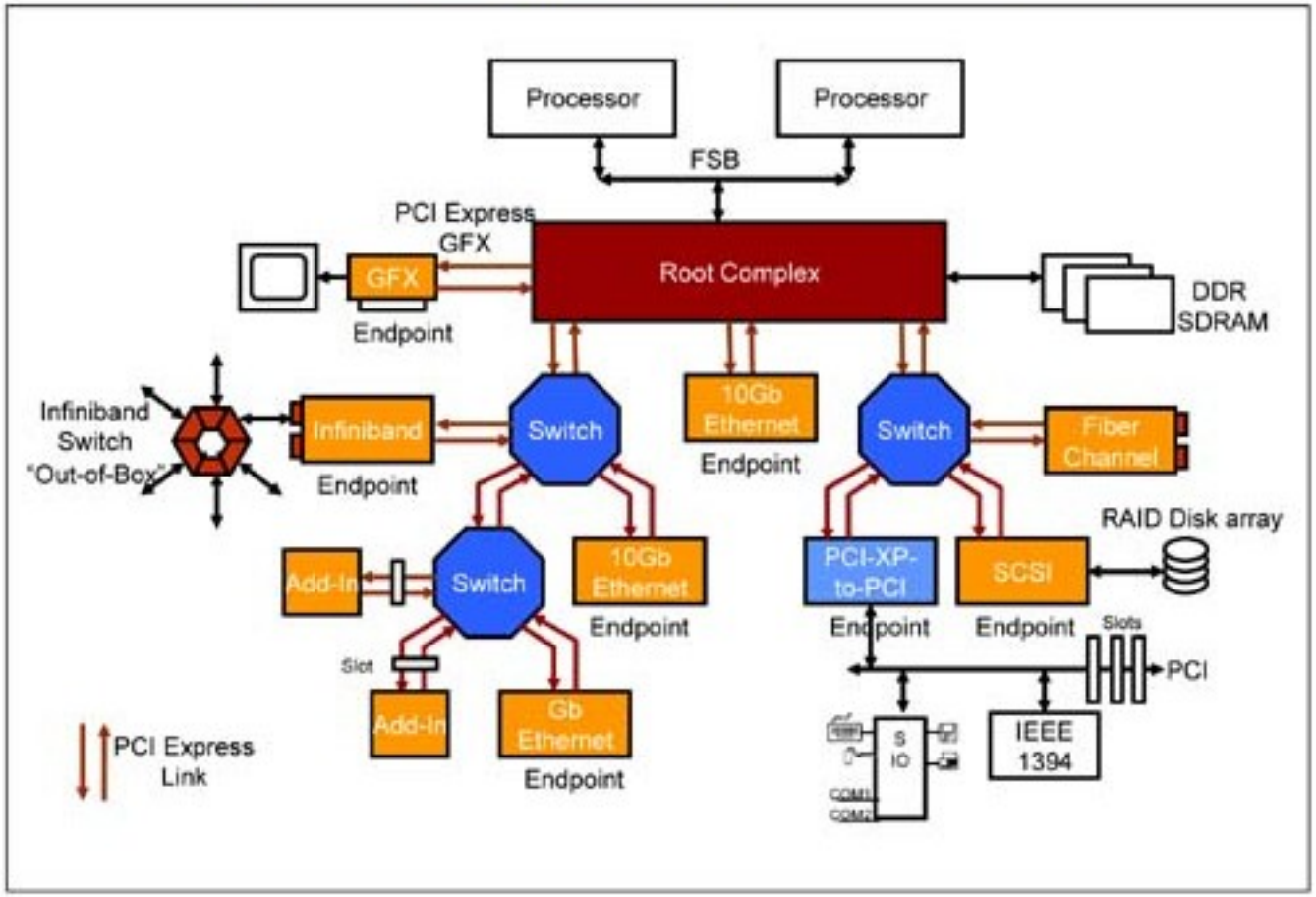
- High speed serial buses - why not used earlier ?
  - Requires high integration and cheap transistors because of added complexity
- Still used only where high performance is needed
  - Small microcontroller systems tend to stay with parallel buses for memory and fast peripherals
    - Easier to debug
    - Cheaper to produce
- Serial buses are used on microcontrollers too
  - For low pin count, slow interfaces
    - SPI (4 lines + 1 line for each additional slave)
    - I2C (2 lines, many slaves)
    - 1-Wire (1 line, many slaves possible)

PCI, PCI Express

- ISA at 8.33 MHz; max. bandwidth of 16.7 MB/sec; video alone = 135 MB/sec.
- Intel designed PCI bus in 1990
- Dominant standard
  - Not only PCs, but also communication systems, embedded systems
- Originally 33 MHz and 32 bit -> 133 MB/sec
- PCI 2.2 -> 66 MHz and 64 bit -> 528 MB/sec
- Not good enough for a memory bus. Not compatible with ISA cards.
- PCI Express is gaining grounds
  - Shift from parallel bus to high-speed serial links
  - PCI is still a dominant player due to large base

PCI Express Organization

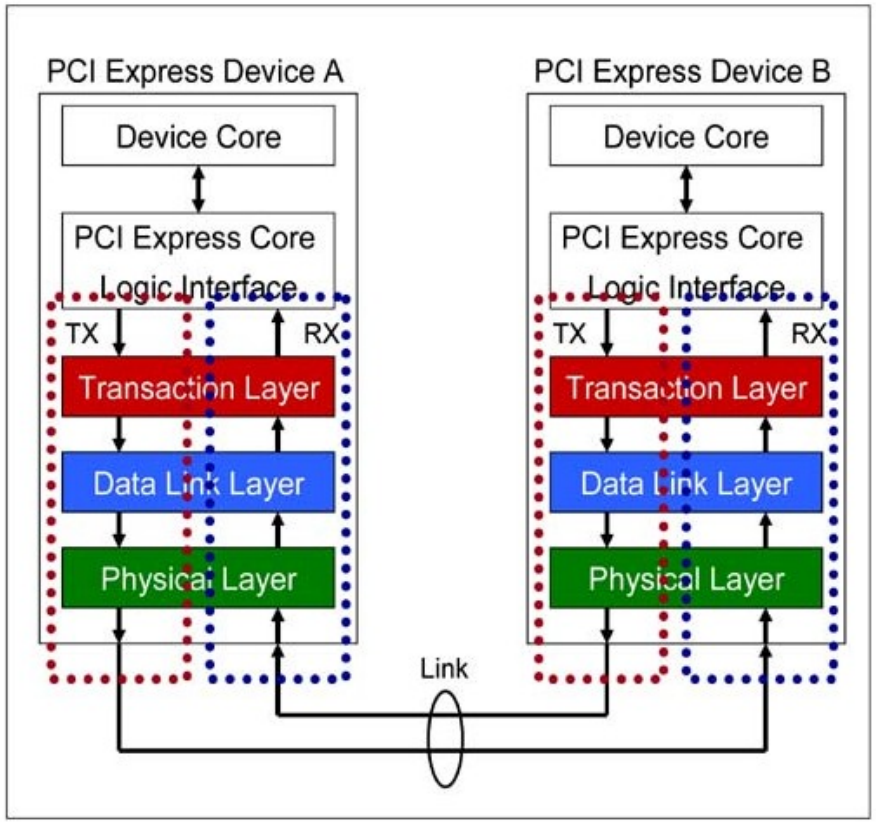
Figure 1-25. PCI Express High-End Server System



Source : Budruk & Al., *PCI Express System Architecture*

PCI Express Bus Transactions

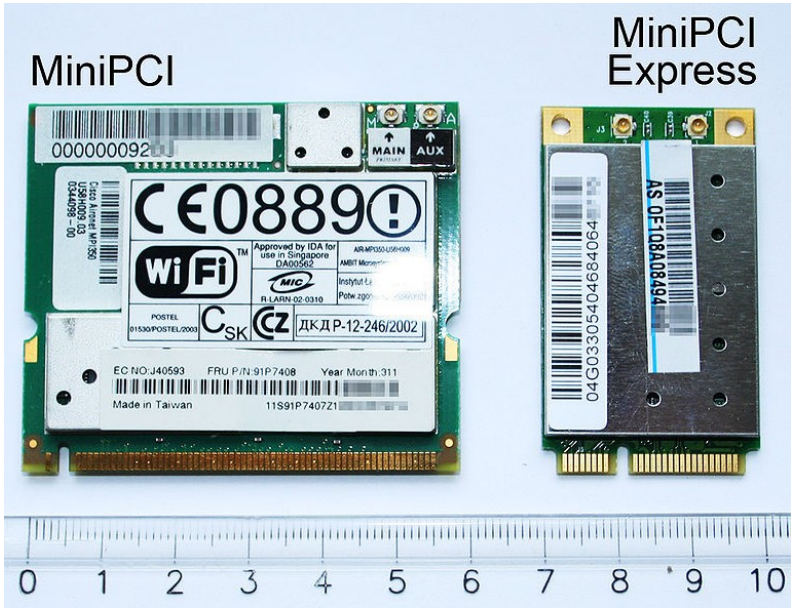
- All links are point to point
- To reach multiple endpoints, traffic must use switches
- Supports interesting features:
  - Hotplug
  - Link Training
  - Power Saving



Source : Budruk & Al., *PCI Express System Architecture*

PCI Express - Architecture

- Bus arbitration is more abstract
  - Using network messages
- Much more scalable
  - Can mix different speeds
  - Backward compatible
- Bandwidth can scale
  - On same connector, x1, x2, x4, x8, x16
  - Rate is ***negociated***
- Future looks good for PCIe



Source : http://en.wikipedia.org/wiki/PCI_Express

Work for this week

- Read Lab Experiment #2 handout
- Attend tutorial given by the TA in the lab
  - Useful information on how to use UART and Timers
- Obtain the keypad if not already in your kit
- Discuss a strategy to meet Experiment #2 objectives
  - Prototype the algorithm/game on PC
  - Get UART and Timers working
- Don't hesitate to use the TA for guidance
  - They will not give you “the answer”
  - They will guide you in finding it
    - Best return on your education investment...