Lecture 14: I/O Controllers & Devices





Inf2C Computer Systems - 2010-2011

Computer System Organization





Inf2C Computer Systems - 2010-2011

Example: RS232 Serial Interface

- I/O controller: UART (Universal Asynchronous Receiver Transmitter) or ACIA (Asynchronous Communications Interface Adapter)
- Used for modems and other serial devices
- Physical Implementation:
 - 2 signal wires (one for each direction) + ground reference



Tx data wire (CPU to I/O device) Rx data wire (I/O device to CPU) GROUND wire



Example: RS232 Serial Interface

- Encoding:
 - 1 character = 10 or 11 bits (including signaling)
 - Idle state is represented by a constant "1"



- Parity: for detection of transmission errors
 - odd \rightarrow total number of "1"s (including parity bit) is odd
 - even \rightarrow total number of "1"s is even



UART Controller



Inf2C Computer Systems - 2010-2011

Connecting CPU and I/O Controllers

- Option 1: connect the I/O Tx and Rx registers directly into some special <u>CPU I/O registers</u> → not flexible
- Option 2: keep I/O registers in separate I/O controller and connect CPU to I/O controller through special <u>I/O bus</u> → expensive, not flexible
 - I/O bus:
 - data lines (8 bits)
 - control lines (READ and WRITE signals),
 - address lines (some few bits) → each I/O controller is assigned a range of addresses for its registers

Data is accessed through special I/O loads and stores



Connecting CPU and I/O Controllers

Option 3: keep I/O registers in I/O controller and connect CPU to I/O controller through <u>memory bus</u>

Memory mapped I/O:

- I/O controller registers (data and control) are mapped to a dedicated portion of memory and are accessed with <u>regular</u> load and store instructions
- Takes bus bandwidth away from CPU-memory
- Option 4: connect I/O controllers to I/O bus and the I/O bus to the memory bus through a bus adapter
 - More flexible
 - Off-load memory bus (multiple I/O devices appear as a single device to the memory bus)



- Used in high-performance systems

I/O via I/O or memory bus



Polling and interrupt-based I/O

Checking I/O status:

Note: only OS has access to registers in I/O controller (memory mapped)

- Option 1: Polling
 - User process calls OS at regular intervals to check status of I/O operation
 - Time-consuming
 - Not flexible
 - Used in embedded systems
- Option 2: <u>Interrupt</u>
 - I/O controller interrupts user process to signal an I/O event
 - Used in general purpose systems



Hard Disks



- 10-50k tracks per surface
 100-500 sectors per track
 512B 4KB per sector
- Spinning speed:
 - 5400-15000 rpm



Disk Performance

- Total time of a disk operation is divided in two parts:
- Access time: time to get head into position to read/write data access time = seek time + rotational latency
 - Seek time: time to move head to appropriate track (< 10ms for hard disk;
 < 100ms for floppy)
 - Rotational latency: time to wait for appropriate sector to arrive underneath the head (< 10ms for hard disk)
 - Spinning speed
- Transfer time: time to move data to/from disk

transfer time = time to transfer 1 byte * number of bytes of data

- Spinning speed and recording density
- 75-125 MB/s for hard disk



Disk Controllers

- Disk controller inside disk unit → responsible for all mechanical operation of disk + interface with CPU
- Disk interface standards
 - EIDE: simple, bus structure is similar to I/O or memory bus
 - SCSI: flexible, requires special bus controller to connect to I/O or memory bus
- I/O registers (usually 8 registers):
 - Exchange data and control between CPU and controller
 - Command register \rightarrow tells controller what to do next
 - e.g. Seek n Read Sector m Write Sector m Format Track

High-level commands



Using a Disk Controller

- Step 1: user program requests data from a file
- Step 2: OS file system determines sector(s) to be accessed
- Step 3: OS disk handler issues Seek command and CPU goes to work on some other process (multi-tasking)
- Step 4: I/O controller <u>interrupts</u> CPU to signal completion of seek
- Step 5: OS disk handler issues Read Sector command and CPU goes to work on some other process
- Step 6: I/O controller <u>interrupts</u> CPU to signal data ready
- Step 7: OS disk handler transfers data to/from disk
- Step 8: go to step 3 or 5 and repeat until all data transferred



Interrupt Approach

- CPU (through OS) has to issue individual commands to read every sector from disk
- Data transfer is very slow (~100µs for a 512 byte sector)
- Interrupt mechanism is very expensive (Lecture 11)
- Time taken by interrupt mechanism makes it difficult to synchronize head position with Read Sector command
- Solution: <u>Direct Memory Access (DMA)</u>



Direct Memory Access

- DMA controller: sits on the memory bus and can <u>independently</u> transfer data to/from memory from/to disk
- DMA registers:
 - Address register \rightarrow position in memory of next data to be read/written
 - Data register \rightarrow temporary storage for data to be transferred
 - Length register \rightarrow number of bytes remaining to be transferred



DMA Organization





DMA Operation

- Step 1, 2: user program requests data, OS determines location of data on disk
- Step 3: OS disk handler issues Seek command and sets up DMA registers (address, length); CPU goes to work on another process
- Step 4, 5: I/O controller <u>interrupts</u> CPU, OS disk handler issues Read Sector command
- Step 6: I/O controller informs DMA controller that data is ready (no need to interrupt CPU)
- Step 7: DMA controller transfers data into memory; length register is decremented until all data is moved (advanced DMA controllers can access multiple tracks with a single operation)
- Step 8: DMA controller <u>interrupts</u> CPU to inform completion of
 DMA operation



Bus Arbitration

- DMA and CPU connect to memory bus → access must be somehow <u>arbitrated</u> to avoid conflicts
- Solution: additional logic (bus arbiter)
 - Authorizes CPU or DMA controller to access memory at any given time
- 2 new wires on memory bus:
 - Bus Request \rightarrow asserted by DMA when it requires the bus
 - Bus Grant → asserted by the CPU when it is not using the bus and thus DMA can use it
 - In case of conflicting requests in the same cycle, CPU usually has priority



A bit of perspective (or, a shameless plug)

- The dimensions of the head are impressive. With a width of less than a hundred nanometers and a thickness of about ten, it flies above the platter at a speed of up to 15,000 RPM, at a height that's the equivalent of 40 atoms. If you start multiplying these infinitesimally small numbers, you begin to get an idea of their significance.
- Consider this little comparison: if the read/write head were a Boeing 747, and the hard-disk platter were the surface of the Earth:-
 - The head would fly at Mach 800
 - At less than one centimeter from the ground
 - And count every blade of grass
 - Making fewer than 10 unrecoverable counting errors in an area equivalent to all of Ireland.
 - (source: Matthieu Lamelot, Tom's Hardware)

