CENG3420 Computer Organization & Design
Lecture 10: I/O Systems Review

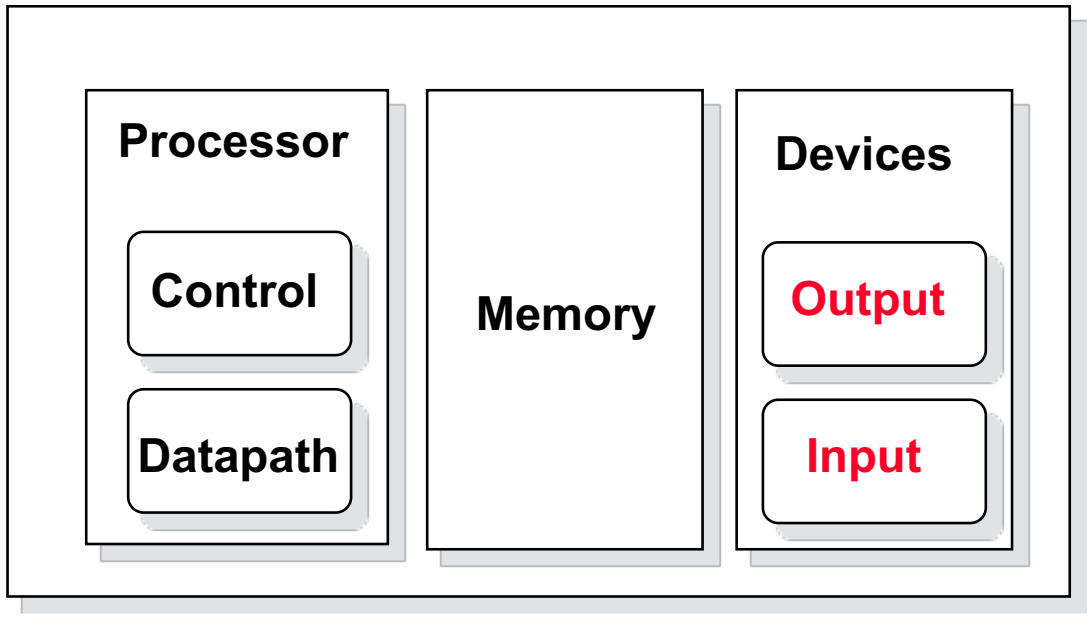
Bei Yu

Spring 2016

byu@cse.cuhk.edu.hk

香港中文大學
The Chinese University of Hong Kong

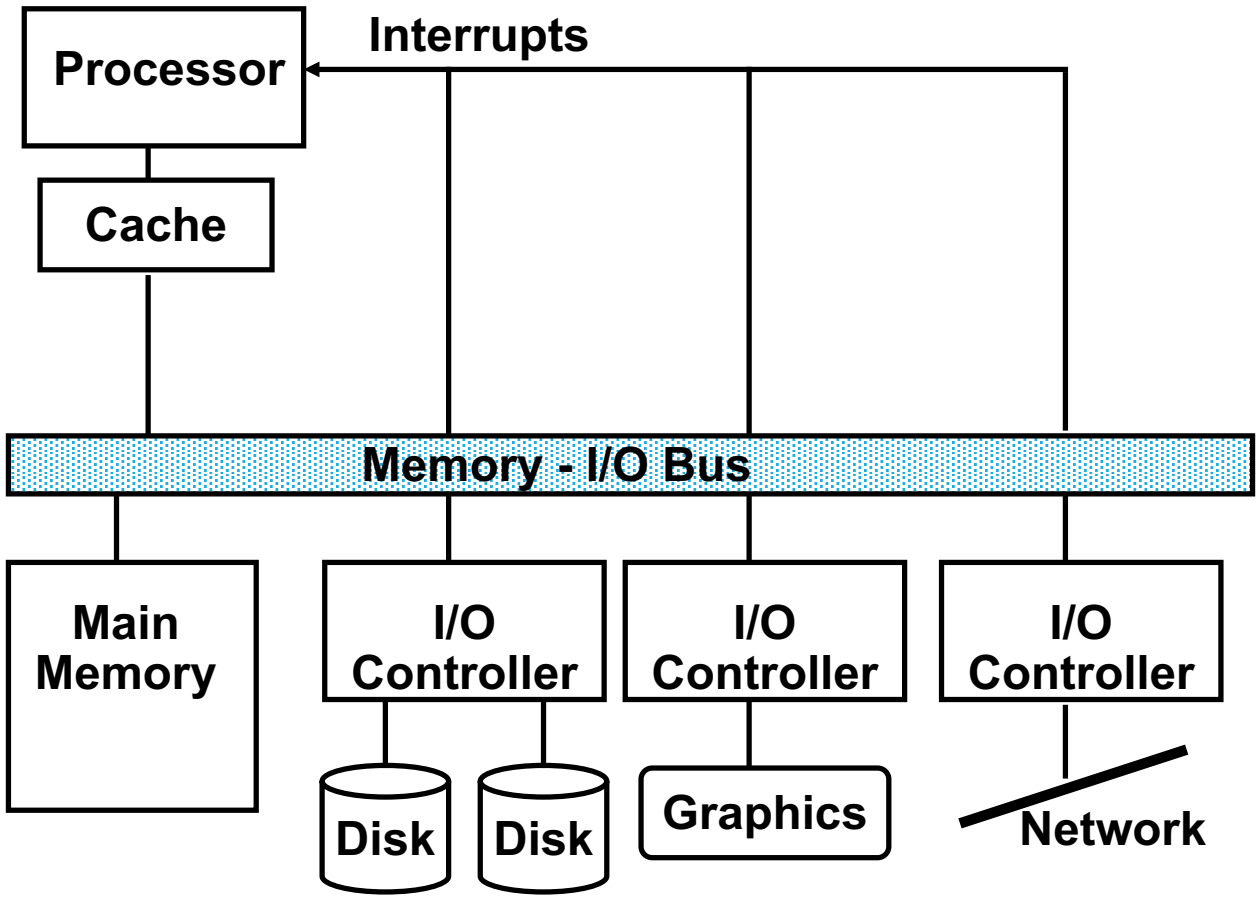
Review: Major Components of a Computer



Important metrics for an I/O system

- Performance
- Expandability
- Dependability
- Cost, size, weight
- Security

A Typical I/O System



I/O Performance Measures

I/O bandwidth (throughput)

- Amount of information that can be input (output) and communicated per unit time
- How much data can we move through the system in a certain time?
- How many I/O operations can we do per unit time?

I/O response time (latency)

- Total elapsed time to accomplish an input or output operation
- An especially important performance metric in real-time systems

Bus

A shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates

Advantages

- Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
- Low cost – a single set of wires is shared in multiple ways

Disadvantages

- Creates a communication bottleneck bus bandwidth limits the maximum I/O throughput

Bus

A shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates

Advantages

- Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
- Low cost – a single set of wires is shared in multiple ways

Disadvantages

- Creates a communication bottleneck bus bandwidth limits the maximum I/O throughput

The maximum bus speed is largely limited by

- The length of the bus
- The number of devices on the bus

I/O Transactions

- An I/O transaction is a sequence of operations over the interconnect that includes a request and may include a response either of which may carry data.
- A transaction is initiated by a single request and may take many individual bus operations.
- An I/O transaction typically includes two parts
  1. Sending the address
  2. Receiving or sending the data

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
-
-
-

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
-
-

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
- ☺ Can accommodate a wide range of devices and device speeds
- ☺ Can be lengthened without worrying about clock skew
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

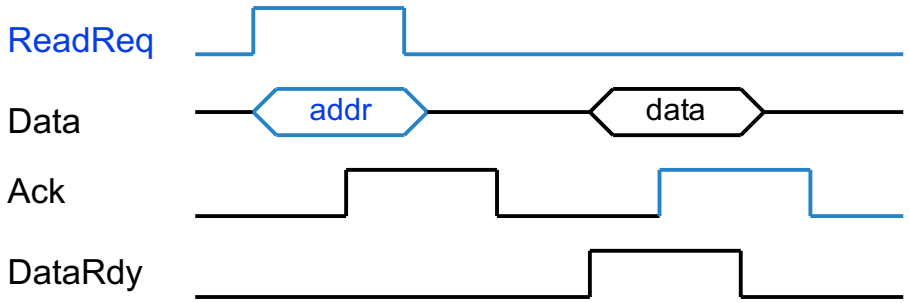
- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
- ☺ Can accommodate a wide range of devices and device speeds
- ☺ Can be lengthened without worrying about clock skew
- ☹ Disadvantage: slow(er)

Asynchronous Bus Handshaking Protocol

Example: data from Memory to I/O devices



1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2.
3.
4.
5.
6.
7.

Interfacing I/O Devices to Processor / Memory

The operating system (OS) acts as the interface between the I/O hardware and the program requesting I/O since

- Multiple programs using the processor share the I/O system
- I/O systems usually use interrupts which are handled by the OS
- Low-level control of an I/O device is complex and detailed

OS must be able to

- give commands to the I/O devices
- be notified the status of I/O device
- transfer data between the memory and the I/O device
- protect I/O devices to which a user program doesn't have access
- schedule I/O requests to enhance system throughput

How Processor Ditects I/O Devices

Port-mapped I/O (PMIO)

- special class of CPU instructions for performing I/O
- EX:

Memory-mapped I/O (MMIO)

- Portions of the high-order memory address space are assigned to each I/O device
- Read and writes to those memory addresses are interpreted as commands to the I/O devices
- Load/stores to the I/O address space can only be done by the OS
- EX:

How Processor Ditects I/O Devices

Port-mapped I/O (PMIO)

- special class of CPU instructions for performing I/O
- EX: `in` and `out` instructions in x86 architecture

Memory-mapped I/O (MMIO)

- Portions of the high-order memory address space are assigned to each I/O device
- Read and writes to those memory addresses are interpreted as commands to the I/O devices
- Load/stores to the I/O address space can only be done by the OS
- EX: MIPS, LC-3b

How I/O Devices Communicate with Processor

Polling

- Processor periodically checks the status of an I/O device (through the OS) to determine its need for service
- Processor is totally in control but does all the work
- Can waste a lot of processor time due to speed differences

Interrupt-driven I/O

- I/O device issues an interrupt to indicate that it needs attention

Interrupt Driven I/O

Asynchronous

- does NOT prevent any instruction from completing
- Need a way to identify the device generating the interrupt
- Can have different urgencies (so need a way to prioritize them)

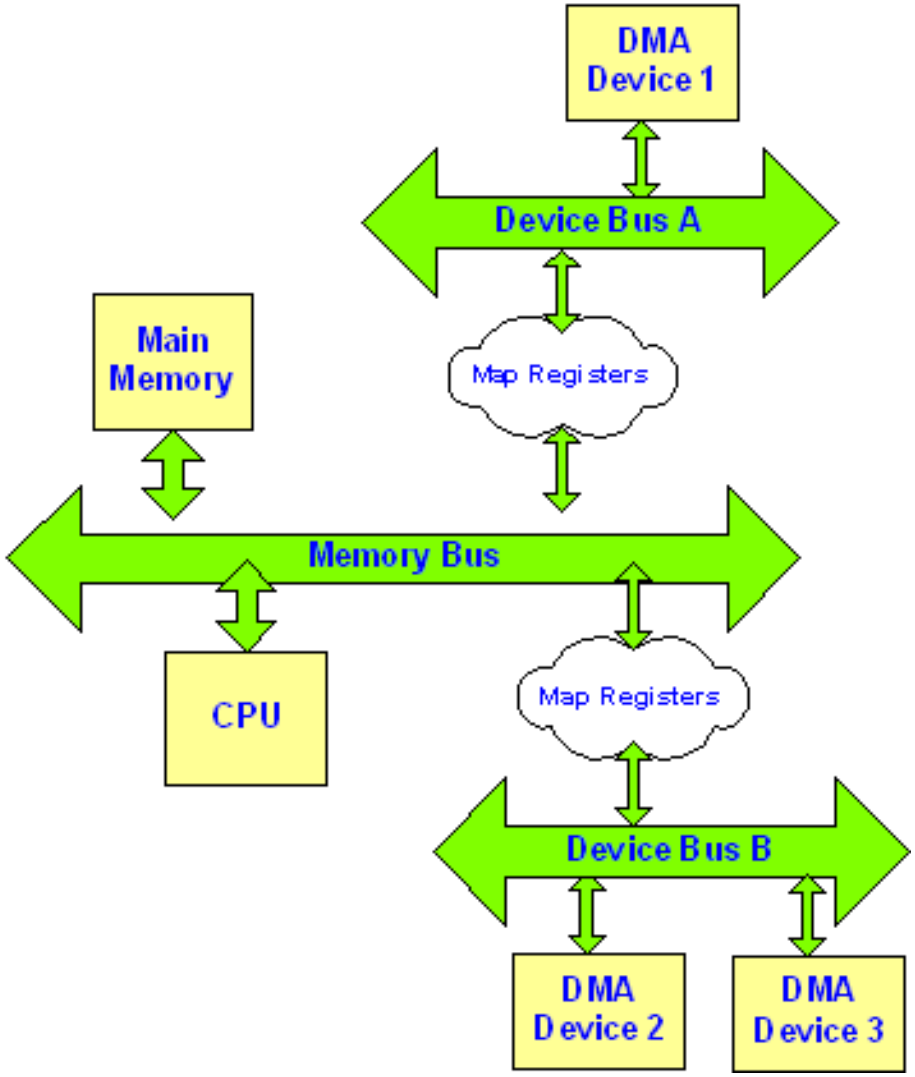
Advantages

- Relieves the processor from having to continuously polling
- user program progress is only suspended during the actual transfer of I/O data to/from user memory space

Disadvantage

- need special hardware support

DMA Example


DMA
Device 1
Device Bus A
Main
Memory
Map Registers
Memory Bus
CPU
Map Registers
Device Bus B
DMA
Device 2
DMA
Device 3

DMA & Virtual Memory Considerations

Should the DMA work with virtual addresses or physical addresses?

If with Physical Address:

- Must constrain all of the DMA transfers to stay within one page because if it crosses a page boundary, then it wont necessarily be contiguous in memory
- If the transfer won't fit in a single page, it can be broken into a series of transfers (each of which fit in a page) which are handled individually and chained together

If with virtual Address:

- The DMA controller will have to translate the virtual address to a physical address (i.e., will need a TLB structure)