ECE 280 / CSE 280
Digital Design Laboratory
Lecture 6

I/O
Input/Output Module

• Interface to CPU and Memory
• Interface to one or more peripherals
Generic Model of I/O Module

Address Lines

Data Lines

Control Lines

System Bus

I/O Module

Links to peripheral devices
External Devices

- Human readable
  - Screen, printer, keyboard

- Machine readable
  - Monitoring and control

- Communication
  - Modem
  - Network Interface Card (NIC)
External Device Block Diagram
Typical I/O Data Rates

- Gigabit Ethernet
- Graphics display
- Hard disk
- Ethernet
- Optical disk
- Scanner
- Laser printer
- Floppy disk
- Modem
- Mouse
- Keyboard

Data Rate (bps)
I/O Module Function

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection
Input Output Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)
Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data

- CPU waits for I/O module to complete operation

- Wastes CPU time
Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later
I/O Commands

• CPU issues address
  – Identifies module (& device if >1 per module)

• CPU issues command
  – Control - telling module what to do
    • e.g. spin up disk
  – Test - check status
    • e.g. power? Error?
  – Read/Write
    • Module transfers data via buffer from/to device
Addressing I/O Devices

- Under programmed I/O data transfer is like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)
I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available

- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set
Interrupt Driven I/O

• Overcomes CPU waiting
• No repeated CPU checking of device
• I/O module interrupts when ready
Interrupt Driven I/O

- Basic Operation
  - CPU issues read command
  - I/O module gets data from peripheral whilst CPU does other work
  - I/O module interrupts CPU
  - CPU requests data
  - I/O module transfers data
CPU Viewpoint

• Issue read command

• Do other work

• Check for interrupt at end of each instruction cycle

• If interrupted:
  – Save context (registers)
  – Process interrupt
    • Fetch data & store
Design Issues

- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted
Identifying Interrupting Module (1)

- Different line for each module
  - PC
  - Limits number of devices
- Software poll
  - CPU asks each module in turn
  - Slow
Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places vector on bus
  - CPU uses vector to identify handler routine

- Bus Master
  - Module must claim the bus before it can raise interrupt
  - e.g. SCSI
Multiple Interrupts

- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines
- If bus mastering only current master can interrupt
Polling vs. Interrupts

- Polling is faster than interrupts because
  - Compiler knows which registers in use at polling point. Hence, do not need to save and restore registers (or not as many).
  - Other interrupt overhead avoided (pipeline flush, trap priorities, etc).

- Polling is slower than interrupts because
  - Overhead of polling instructions is incurred regardless of whether or not handler is run. This could add to inner-loop delay.
  - Device may have to wait for service for a long time.
Polling vs. Interrupts

• When to use one or the other?
  – Multi-axis tradeoff
    • Frequent/regular events good for polling, as long as device can be controlled at user level.
    • Interrupts good for infrequent/irregular events
    • Interrupts good for ensuring regular/predictable service of events.
I/O without Interrupts

- Polling based I/O

I/O module

CPU

idle

prepare to write
I/O without Interrupts

I/O module

CPU

writing

idle
I/O without Interrupts

I/O module

CPU

idle

processing

complete write
I/O with Interrupts

I/O module

CPU

idle

prepare to write
I/O with Interrupts

I/O module

CPU

writing

processing
I/O with Interrupts

I/O module

CPU

idle

processing

complete write
Interfacing

• Connecting devices together
• Bit of wire?
• Dedicated processor/memory/buses?
• Similar characteristics to processor/memory buses
• E.g. RS-232, USB, PS/2, FireWire, InfiniBand, SCSI, IDE
Data Transmission

- How do you synchronize the signal?
- Transmit clock with data
  - OK for short distances
- Recover clock from transmission signal
  - Requires PLL
  - Network transmission
  - Disk/Tape read
- Synchronize data to local clock
  - Not as fast
  - Simpler to design
Data Transmission

• RS-232
  – -3 to -15V for logic 1
  – +3 to +15V for logic 0

• Up to 192Kbps
Data Transmission

• Transmission Line Characteristics
  – Attenuation - wire resistance causes signal loss
  – Reflection - Caused by improper resistance termination
Data Transmission

- Adjacent signals will have cross-coupled capacitances and inductances that will degrade signal
- Can be minimized by putting ground signals between each data signal
- External EM effects can be reduced with adequate shielding or using twisted pair
Lab 5

- PS/2 Keyboard controller
  - Keyboard reader
    - Serial port state machine gets scan codes from PS/2 port
    - Convert scan codes to ASCII
    - Reader state machine handles special keys - shift, ctrl, alt
  - LED display
    - Gets data_available interrupt from keyboard reader
    - Holds onto last eight ASCII characters received
Lab 5

ps2_clk

ps2_data
Lab 5

- Scan codes are 11 bits long
  - 0 start bit
  - 8 scan code bits
  - Odd parity bit
  - 1 stop bit
Lab 5

- Keyboard scancodes

- Scancode F0 followed by key scancode indicates key up