**Overview**

- I/O Background
- Polling
- Interrupts

---

### Anatomy: 5 components of any Computer

- **Computer**
  - Processor (active)
  - Memory (passive) (where programs, data live when running)
  - Control ("brain")
  - Datapath ("brawn")

- **Devices**
  - Input
  - Output

- **Keyboard, Mouse**
- **Disk** (where programs, data live when not running)

- **Display, Printer**

---

### Motivation for Input/Output

- I/O is how humans interact with computers
- I/O lets computers do amazing things:
  - Read pressure of synthetic hand and control synthetic arm and hand of fireman
  - Control propellers, fins, communicate in BOB (Breathable Observable Bubble)
  - Read bar codes of items in refrigerator
- Computer without I/O like a car without wheels; great technology, but won’t get you anywhere
I/O Device Examples and Speeds

- **I/O Speed**: bytes transferred per second (from mouse to display: million-to-1)

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (Kbytes/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Line Printer</td>
<td>Output</td>
<td>Human</td>
<td>1.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Network-LAN</td>
<td>I or O</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
</tbody>
</table>

What do we need to make I/O work?

- A way to connect many types of devices to the Proc-Mem
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

Operating System

Files   Windows

Buses in a PC: Connect a few devices

- CPU
- Memory: 133 MHz, 8 bytes ⇒ 1064 MB/s (peak)
- PCI: 33 MHz, 8 bytes wide ⇒ 264 MB/s (peak)
- SCSI: “Ultra3” (80 MHz), “Wide” (2 bytes) ⇒ 160 MB/s (peak)
- Ethernet: 12.5 MB/s (peak)

Instruction Set Architecture for I/O

- Some machines have special input and output instructions

  Alternative model (used by ARM):
  - Input: ~ reads a sequence of bytes
  - Output: ~ writes a sequence of bytes

  Memory access also reading/ writing a sequence of bytes, so use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)
I/O devices often have a few registers
- Status/Control registers
- I/O registers

If these have an interface that looks like memory, we can connect them to the memory bus
- Reads/Writes to certain locations will produce the desired change in the I/O device controller

Typically, devices map to only a few bytes in memory

- 500 MHz microprocessor can execute 500 million load or store instructions per second, or 2,000,000 KB/s data rate
- I/O devices from 0.01 KB/s to 30,000 KB/s

Input: device may not be ready to send data as fast as the processor loads it
- Also, might be waiting for human to act

Output: device may not be ready to accept data as fast as processor stores it
- What to do?
Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - 1 register says it’s OK to read/write (I/O ready), often called Status Register
  - 1 register that contains data, often called Data Register

- Processor reads from Status Register in loop, waiting for device to set Ready bit in Status reg to say its OK (0 ⇒ 1)

- Processor then loads from (input) or writes to (output) data register
  - Load from device/Store into Data Register resets Ready bit (1 ⇒ 0) of Status Register

“What’s This Stuff Good For?”

Remote Diagnosis:
“NeoRest ExII,” a high-tech toilet features microprocessor-controlled seat warmers, automatic lid openers, air deodorizers, water sprays and blow-dryers that do away with the need for toilet tissue. About 25 percent of new homes in Japan have a “washlet,” as these toilets are called. Toto’s engineers are now working on a model that analyzes urine to determine blood-sugar levels in diabetics and then automatically sends a daily report, by modem, to the user’s physician. One Digital Day, 1998

www.intel.com/onedigitalday

DSLMU/Komodo Address Space

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0x003FFFFF</td>
<td>4 MB</td>
<td>Read/write memory</td>
</tr>
<tr>
<td></td>
<td>(RAM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00400000</td>
<td>0x0FFFFFFF</td>
<td>(252 MB)</td>
<td>Unused</td>
</tr>
<tr>
<td>0x10000000</td>
<td>0x1FFFFFFFF</td>
<td>256 MB</td>
<td>Microcontroller I/O space</td>
</tr>
<tr>
<td>0x20000000</td>
<td>0x2FFFFFFFF</td>
<td>256 MB</td>
<td>Spartan-XL FPGA for I/O Expansion</td>
</tr>
<tr>
<td>0x30000000</td>
<td>0x3FFFFFFFF</td>
<td>256 MB</td>
<td>Virtex-E FPGA for Co-processing</td>
</tr>
<tr>
<td>0x40000000</td>
<td>0xFFFFFFFFF</td>
<td>(3072 MB)</td>
<td>Unused</td>
</tr>
</tbody>
</table>

DSLMU I/Os

- Two RS232 serial port connectors
- LEDs on the MU Board,
- Boot Select switches
- LCD module
- Spartan-XL FPGA for I/O Expansion
- Virtex-E FPGA for Co-processing
- Single-chip 10 Mb Ethernet
- Uncommitted Peripherals
- Timers
- Ref: Hardware Ref Manual on CD-ROM.
### DSLMU/Komodo I/O Addressing

<table>
<thead>
<tr>
<th>Offset</th>
<th>Mode</th>
<th>Port Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>R/W</td>
<td>Port A</td>
<td>Bidirectional data port to LEDs, LCD, etc.</td>
</tr>
<tr>
<td>0x04</td>
<td>R/W</td>
<td>Port B</td>
<td>Control port (some bits are read only)</td>
</tr>
<tr>
<td>0x08</td>
<td>R/W</td>
<td>Timer</td>
<td>8-bit free-running 1 kHz timer</td>
</tr>
<tr>
<td>0x0C</td>
<td>R/W</td>
<td>Timer Compare</td>
<td>Allows timer interrupts to be generated</td>
</tr>
<tr>
<td>0x10</td>
<td>RO</td>
<td>Serial RxD</td>
<td>Read a byte from the serial port</td>
</tr>
<tr>
<td>0x10</td>
<td>WO</td>
<td>Serial TxD</td>
<td>Write a byte to the serial port</td>
</tr>
<tr>
<td>0x14</td>
<td>WO</td>
<td>Serial Status</td>
<td>Serial port status port</td>
</tr>
<tr>
<td>0x18</td>
<td>R/W</td>
<td>IRQ Status</td>
<td>Bitmap of currently-active interrupts</td>
</tr>
<tr>
<td>0x1C</td>
<td>R/W</td>
<td>IRQ Enable</td>
<td>Controls which interrupts are enabled</td>
</tr>
<tr>
<td>0x20</td>
<td>WO</td>
<td>Debug Stop</td>
<td>Stops program execution when written to</td>
</tr>
</tbody>
</table>

### DSLMU/Komodo Ports A & B:

**Port A:**
- **Bidirectional:**
- **Port B:** Bit 4: LEDs Enable, Bit 2: Port A direction, Bit 1: LC_RS, Bit 0: LC_EN, Bit 4: LEDs enable

### I/O Example

**Output: Write to LED Port**

```
.set io_base, 0x10000000 ; Base of the DSLMU I/O space
.set portA, 0x00 ; Offset of Port A in the I/O space
.set portB, 0x04 ; Offset of Port B in the I/O space
mov r2, #io_base ; Use R2 as a base address pointer
mov r0, #0b00010000 ; Set bit 4 and reset all other bits
strb r0, [r2, #portB] ; Send the data to Port B (R2 + portB)
mov r0, #0b10100101 ; R0 = data for the LEDs
strb r0, [r2, #portA] ; Stops program execution when written to
```

### DSLMU/Komodo Serial I/Os

**DSLMU Serial Port:** memory-mapped terminal (Connected to the PC for program download and debugging)

- **Read from PC Keyboard (receiver); 2 device regs**
- **Writes to PC terminal (transmitter); 2 device regs**

**Receiver Status**

```
0x10000010  Unused (00...00)
0x10000014  Received Byte
```

**Transmitter Status**

```
0x10000010  Unused (00...00)
0x10000014  Transmitted Byte
```
DSLMU/Komodo Serial I/Os

- Status register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read (or ready to be read);
    \[ 1 \Rightarrow 0 \text{ when data is read from Data Reg} \]
  - Transmitter: Ready==1 means transmitter is ready to accept a new character;
    \[ 0 \Rightarrow \text{Transmitter still busy writing last char} \]

- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display

Serial I/O Example (Read)

- Input: Read from PC keyboard into R0
  
  ```
  .set iobase, 0x10000000 ; Base of DSLMU I/O space
  .set ser_RxD, 0x10      ; Serial RxD port
  .set ser_stat, 0x14     ; Serial Status port
  .set ser_Rx_rdy, 0b01   ; Test bit 0 for RxD ; ready status

  readbyte:
  ldr  r1, =iobase ; R1 = base address of I/O Space
  Waitloop:
  ldrb r0,[r1,#ser_stat] ; Read the serial port status
  tst r0,#ser_Rx_rdy ; Check whether a byte is ready to be read
  beq Waitloop ; (No: jump back and try again
  ldrb r0,[r1,#ser_RxD] ; Read the available byte into R0
  mov pc, lr
  ```
  
- Processor waiting for I/O called “Polling”

Serial I/O Example (Write)

- Input: Write from to Display from R0
  
  ```
  .set iobase, 0x10000000 ; Base of DSLMU I/O space
  .set ser_RxD, 0x10      ; Serial TxD port
  .set ser_stat, 0x14     ; Serial Status port
  .set ser_Tx_rdy, 0b10   ; Test bit 1 for TxD ; ready status

  writebyte:
  ldr r1, =iobase ; R1 = base address of I/O Space
  Waitloop:
  ldrb r2,[r1,#ser_stat] ; Read the serial port status
  tst r2,#ser_Tx_rdy ; Check whether is ready to accept new data
  beq Waitloop ; (No: jump back and try again
  strb r0,[r1,#ser_TxD] ; Send the next byte from R0
  mov pc, lr
  ```
  
- Processor waiting for I/O called “Polling”

Reading Material

- Reading Material:
  - Experiment 4 Documentation
  - Hardware Reference Manual on CD-ROM
Serial I/O Example Quiz

° What gets printed out?

1. ABC
2. AB
3. AC

```assembly
ldr r1,=iobase
mov r0, 'A';
strb r0,[r1,#ser_TxD]
mov r0, 'B';
strb r0,[r1,#ser_TxD]
mov r0, 'C';

Waitloop:
ldrb r1,[r1,#ser_stat] ; Read the serial port
                       ; status
tst  r1,#ser_Tx_rdy    ; Check whether is ready
                       ; to accept new data
beq  Waitloop          ; (No: jump back and try
                       ; again
strb r0,[r1,#ser_TxD] ; Send the next byte
                       ; from R0
```

Cost of Polling?

° Assume for a processor with a 500-MHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling:

- Mouse: polled 30 times/sec so as not to miss user movement
- Floppy disk: transfers data in 2-byte units and has a data rate of 50 KB/second. No data transfer can be missed.
- Hard disk: transfers data in 16-byte chunks and can transfer at 8 MB/second. Again, no transfer can be missed.

% Processor time to poll mouse, floppy

° Mouse Polling Clocks/sec
  = 30 * 400 = 12000 clocks/sec

° % Processor for polling:
  12*10^3/500*10^6 = 0.002%
  ⇒ Polling mouse little impact on processor

° Times Polling Floppy/sec
  = 50 KB/s /2B = 25K polls/sec

° Floppy Polling Clocks/sec
  = 25K * 400 = 10,000,000 clocks/sec

° % Processor for polling:
  10*10^6/500*10^6 = 2%
  ⇒ OK if not too many I/O devices

% Processor time to hard disk

° Times Polling Disk/sec
  = 8 MB/s /16B = 500K polls/sec

° Disk Polling Clocks/sec
  = 500K * 400 = 200,000,000 clocks/sec

° % Processor for polling:
  200*10^6/500*10^6 = 40%
  ⇒ Unacceptable
What is the alternative to polling?

° Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready

° Wish we could have an unplanned procedure call that would be invoked only when I/O device is ready

° Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer

Interrupt Driven Data Transfer

1. I/O interrupt
2. Save PC
3. Interrupt service addr
4. Read store
5. Return

Benefit of Interrupt-Driven I/O

° 500 clock cycle overhead for each transfer, including interrupt. Find the % of processor consumed if the hard disk is only active 5% of the time.

° Interrupt rate = polling rate
  • Disk Interrupts/sec = 8 MB/s /16B = 500K interrupts/sec
  • Disk Polling Clocks/sec = 500K * 500 = 250,000,000 clocks/sec
  • % Processor for during transfer: 250*10^6/500*10^6= 50%

° Disk active 5% ⇒ 5% * 50% ⇒ 2.5% busy

Polling vs. Interrupt Analogy

° Imagine yourself on a long road trip with your 10-year-old younger brother... (You: I/O device, brother: CPU)

° Polling:
  • “Are we there yet? Are we there yet? Are we there yet? ....”
  • CPU not doing anything useful

° Interrupt:
  • Stuff him a color gameboy, “interrupt” him when arrive at destination
  • CPU does useful work while I/O busy
Conclusion (#1/2)

° I/O is how humans interact with computers
° I/O lets computers do amazing things
° I/O devices often have a few registers
  • Status registers
  • I/O registers
  • Typically, devices map to only a few bytes in memory

Conclusion (#2/2)

° I/O gives computers their 5 senses
° I/O speed range is million to one
° Processor speed means must synchronize with I/O devices before use
° Polling works, but expensive
  • processor repeatedly queries devices
° Interrupts works, more complex