Hardware-Software Interfaces

CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Spring 2008
Basic Processor Architecture

Controller

Operation

Result

Latch

Latch

Read, Write

Registers

Address Reg.

Memory

Shared Bus
Typical Processor System

- Processor
- Memory
- Peripheral
- Peripheral

R/W
Enable
Address
Data
Simple Bus Timing

Read Cycle

- **R/W**: Read/Write signal
- **Enable**: Enable signal
- **Addr**: Address signal
- **Data**: Data signal

Write Cycle

- **R/W**: Read/Write signal
- **Enable**: Enable signal
- **Addr**: Address signal
- **Data**: Data signal

Hardware-Software Interfaces – p.
Strobe vs. Handshake

Strobe

Req

Data

Handshake

Req

Ack

Data
1982: The IBM PC
The ISA Bus: Memory Read
The ISA Bus: Memory Write

- **CLK**
- **Addr**
- **BALE**
- **MEMW**
- **IOCHRDY**
- **Data**
Embedded System Legos. Stack ’em and go.
Memory-Mapped I/O

- To a processor, everything is memory.
- Peripherals appear as magical memory locations.
- Status registers: when read, report state of peripheral
- Control registers: when written, change state of peripheral
Typical Peripheral: PC Parallel Port

At Standard TTL Levels

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Adapter Pin Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Strobe</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>+Data Bit 0</td>
</tr>
<tr>
<td>X</td>
<td>+Data Bit 1</td>
</tr>
<tr>
<td>T</td>
<td>+Data Bit 2</td>
</tr>
<tr>
<td>E</td>
<td>+Data Bit 3</td>
</tr>
<tr>
<td>R</td>
<td>+Data Bit 4</td>
</tr>
<tr>
<td>N</td>
<td>+Data Bit 5</td>
</tr>
<tr>
<td>A</td>
<td>+Data Bit 6</td>
</tr>
<tr>
<td>L</td>
<td>+Data Bit 7</td>
</tr>
<tr>
<td>-Acknowledge</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>+Busy</td>
</tr>
<tr>
<td>E</td>
<td>+Paper End</td>
</tr>
<tr>
<td>V</td>
<td>+Select</td>
</tr>
<tr>
<td>I</td>
<td>-Auto Feed</td>
</tr>
<tr>
<td>C</td>
<td>-Error</td>
</tr>
<tr>
<td>E</td>
<td>-Initialize</td>
</tr>
<tr>
<td></td>
<td>-Select Input</td>
</tr>
<tr>
<td></td>
<td>Ground</td>
</tr>
</tbody>
</table>

Centronics Handshake

nStrobe

Busy

nAck

Data
Parallel Port Registers

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td>Ack</td>
<td>Paper</td>
<td>Sel</td>
<td>Err</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 0x378 | 0x379 | 0x37A |

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge

Centronics Handshake

- nStrobe
- Busy
- nAck
- Data
#define DATA 0x378
#define STATUS 0x379
#define CONTROL 0x37A
#define NBSY 0x80
#define NACK 0x40
#define OUT 0x20
#define SEL 0x10
#define NERR 0x08
#define STROBE 0x01

#define INVERT (NBSY | NACK | SEL | NERR)
#define MASK (NBSY | NACK | OUT | SEL | NERR)
#define NOT_READY(x) ((inb(x) ^ INVERT) & MASK)

void write_single_character(char c) {
    while (NOT_READY(STATUS)) ;
    outb(DATA, c);
    outb(CONTROL, control | STROBE); /* Assert STROBE */
    outb(CONTROL, control ); /* Clear STROBE */
}
Interrupts and Polling

Two ways to get data from a peripheral:

- Polling: “Are we there yet?”
- Interrupts: Ringing Telephone
Interrupts

Basic idea:

1. Peripheral asserts a processor’s interrupt input
2. Processor temporarily transfers control to interrupt service routine
3. ISR gathers data from peripheral and acknowledges interrupt
4. ISR returns control to previously-executing program
Many Different Interrupts

Processor \( \text{Int} \) Peripheral

What’s a processor to do?
Interrupt Polling

Processor receives interrupt
ISR polls all potential interrupt sources
Prioritizes incoming requests & notifies processor
ISR reads 8-bit interrupt vector number of winner
IBM PC/AT: two 8259s; became standard