Hardware-Software Interfaces

CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Spring 2010
Basic Processor Architecture

Controller

Operation

Result

Latch

Latch

Read, Write

Registers

Address Reg.

Memory

Shared Bus

Hardware-Software Interfaces – p.
Typical Processor System

- Processors
- Memory
- Peripheral
- Peripheral

- R/W
- Enable
- Address
- Data

Hardware-Software Interfaces – p.
Simple Bus Timing

Read Cycle

Write Cycle

R/W

Enable

Addr

Data

R/W

Enable

Addr

Data
Strobe vs. Handshake

Strobe

Req / C4 / C4 / A1 / C0 / C0 / C0 / A0 / C4 / C4 / C4 / A0 / C4 / C4 / C4


Handshake

Req / C4 / A1 / C0 / C0 / C0 / C0 / A0 / C4 / C4 / C4 / C4 / C4

Ack / C4 / C4 / C4 / C4 / C4 / A1 / C0 / C0 / C0 / A0 / C4 / C4


Hardware-Software Interfaces – p.
1982: The IBM PC
The ISA Bus: Memory Read

Hardware-Software Interfaces – p.
The ISA Bus: Memory Write

- **CLK**
- **Addr**
- **BALE**
- **MEMW**
- **IOCHRDY**
- **Data**

The diagram shows the timing for the memory write cycle with the following phases:
- **C1**: CLock
- **C2**: Address
- **Wait**: BALE
- **C3**: MEMW
- **C4**: IOCHRDY

The Data signal remains active during the **C4** phase.
The PC/104 Form Factor: ISA Lives

3.8in

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

### Centronics Handshake

- **nStrobe**
- **Busy**
- **nAck**
- **Data**

<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>+Data Bit 0</td>
<td>P</td>
</tr>
<tr>
<td>+Data Bit 1</td>
<td>A</td>
</tr>
<tr>
<td>+Data Bit 2</td>
<td>R</td>
</tr>
<tr>
<td>+Data Bit 3</td>
<td>A</td>
</tr>
<tr>
<td>+Data Bit 4</td>
<td>L</td>
</tr>
<tr>
<td>+Data Bit 5</td>
<td>L</td>
</tr>
<tr>
<td>+Data Bit 6</td>
<td>E</td>
</tr>
<tr>
<td>+Data Bit 7</td>
<td>L</td>
</tr>
<tr>
<td>-Acknowledge</td>
<td>10</td>
</tr>
<tr>
<td>+Busy</td>
<td>11</td>
</tr>
<tr>
<td>+Paper End</td>
<td>D</td>
</tr>
<tr>
<td>+Select</td>
<td>A</td>
</tr>
<tr>
<td>-Auto Feed</td>
<td>P</td>
</tr>
<tr>
<td>-Error</td>
<td>T</td>
</tr>
<tr>
<td>-Initialize</td>
<td>E</td>
</tr>
<tr>
<td>-Select Input</td>
<td>R</td>
</tr>
<tr>
<td>Ground</td>
<td>18-25</td>
</tr>
</tbody>
</table>
### 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>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td>Ack</td>
<td>Paper</td>
<td>Sel</td>
<td>Err</td>
<td>Sel</td>
<td>Init</td>
<td>Auto</td>
<td>Strobe</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
A Parallel Port Driver

```c
#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 */
}
```
The Parallel Port Schematic

Address decoding

Bus driver

Data latch

Data read

Control latch

Control read
Interrupts and Polling

Two ways to get data from a peripheral:

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

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