Hardware-Software Interfaces
CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Spring 2019
Processor System Block Diagram
Simple Bus Timing

Read Cycle

<table>
<thead>
<tr>
<th>R/W</th>
<th>Enable</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
</table>

Write Cycle

<table>
<thead>
<tr>
<th>R/W</th>
<th>Enable</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
</table>
Strobe vs. Handshake

Strobe

- Req
- Data

Handshake

- Req
- Ack
- Data
1982: The IBM PC/XT
The ISA Bus: Memory Read
The ISA Bus: Memory Write

![Diagram illustrating the ISA Bus: Memory Write sequence with signals and timing phases.](image-url)
The PC/104 Form Factor: ISA Lives

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>+Data Bit 0</td>
<td>2P</td>
</tr>
<tr>
<td>+Data Bit 1</td>
<td>3A</td>
</tr>
<tr>
<td>+Data Bit 2</td>
<td>4R</td>
</tr>
<tr>
<td>+Data Bit 3</td>
<td>5A</td>
</tr>
<tr>
<td>+Data Bit 4</td>
<td>6L</td>
</tr>
<tr>
<td>+Data Bit 5</td>
<td>7L</td>
</tr>
<tr>
<td>+Data Bit 6</td>
<td>8E</td>
</tr>
<tr>
<td>+Data Bit 7</td>
<td>9L</td>
</tr>
<tr>
<td>Acknowledge</td>
<td>10</td>
</tr>
<tr>
<td>Busy</td>
<td>11A</td>
</tr>
<tr>
<td>+Paper End</td>
<td>12D</td>
</tr>
<tr>
<td>+Select</td>
<td>13D</td>
</tr>
<tr>
<td>-Auto Feed</td>
<td>14P</td>
</tr>
<tr>
<td>Error</td>
<td>15T</td>
</tr>
<tr>
<td>-Initialize</td>
<td>16E</td>
</tr>
<tr>
<td>-Select Input</td>
<td>17R</td>
</tr>
<tr>
<td>Ground</td>
<td>18-25</td>
</tr>
</tbody>
</table>

**Waveforms**

- **Strobe**: 
  - Low
  - High

- **Busy**: 
  - Low
  - Rise
  - Fall
  - Low

- **Ack**: 
  - Low
  - Rise
  - Fall
  - Low

- **Data**: 
  - Low
  - High
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>0x378</th>
<th>0x379</th>
<th>0x37A</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>
<td></td>
</tr>
</tbody>
</table>

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge
A Parallel Port Driver

#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
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?

ISR polls all potential interrupt sources, then dispatches handler.
Many Different Interrupts

What’s a processor to do?
ISR polls all potential interrupt sources, then dispatches handler.
Prioritizes incoming requests & notifies processor

ISR reads 8-bit interrupt vector number of winner

IBM PC/AT: two 8259s; became standard