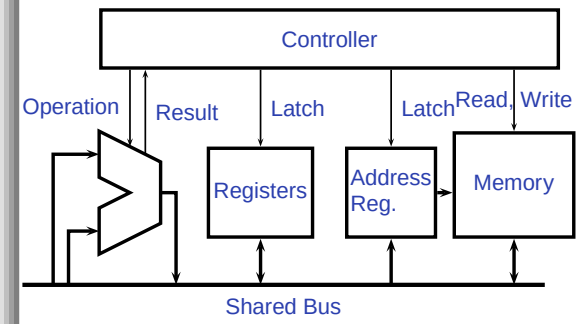
Hardware-Software Interfaces

CSEE W4840

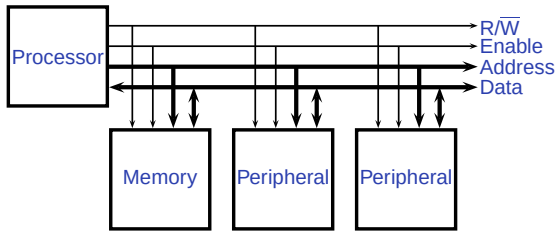
Prof. Stephen A. Edwards

Columbia University

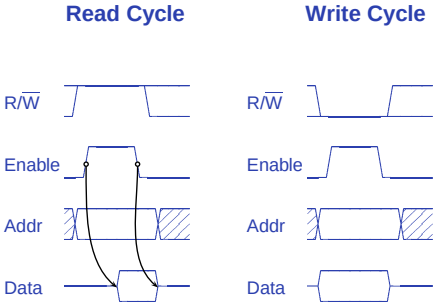
Basic Processor Architecture



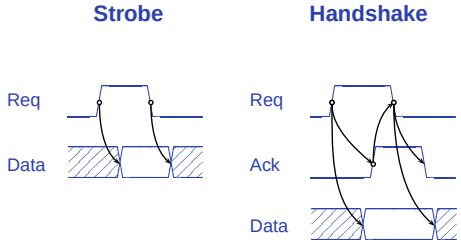
Typical Processor System



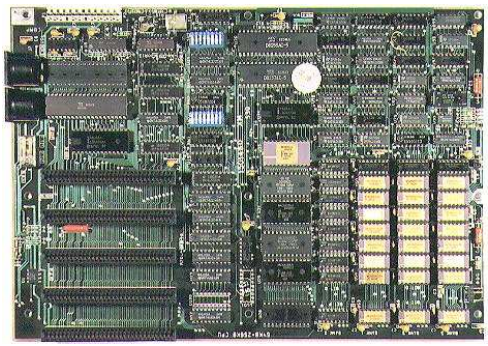
Simple Bus Timing



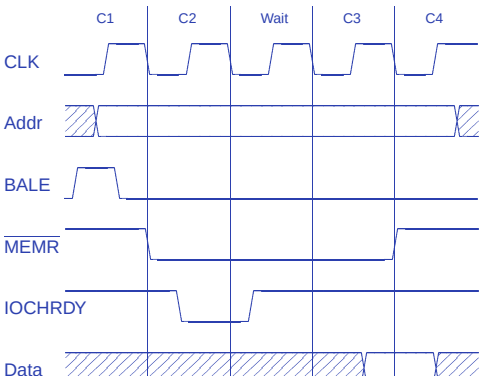
Strobe vs. Handshake



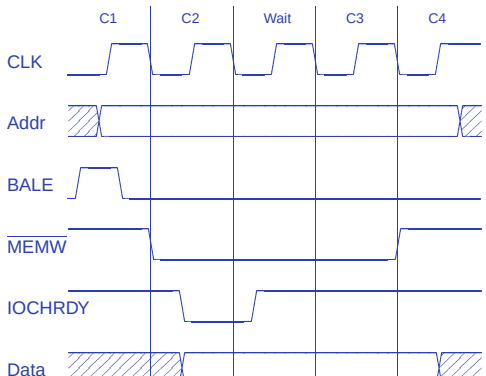
1982: The IBM PC



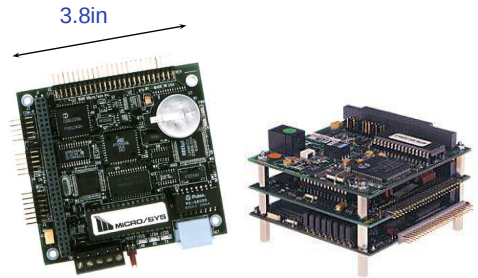
The ISA Bus: Memory Read



The ISA Bus: Memory Write



The PC/104 Form Factor: ISA Lives

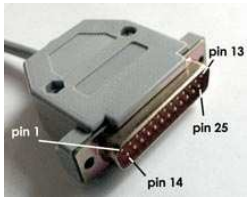


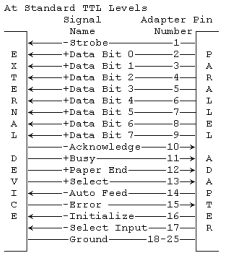
Embedded System Legos. Stack 'em and go.

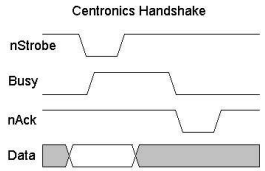
Memory-Mapped I/O

- 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



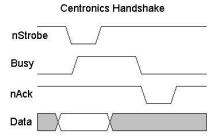




Parallel Port Registers

D7	D6	D5	D4	D3	D2	D1	D0	
$\overline{\text{Busy}}$	Ack	Paper	Sel	Err				0x378
								0x379
				$\overline{\text{Sel}}$	Init	$\overline{\text{Auto}}$	$\overline{\text{Strobe}}$	0x37A

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 */
}
```


Interrupts and Polling

Two ways to get data from a peripheral:

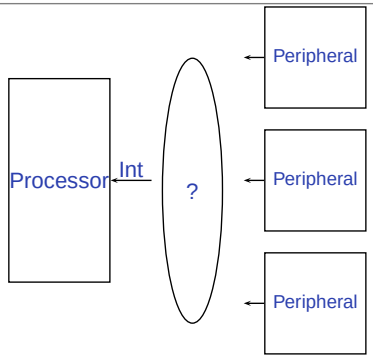
- Polling: “Are we there yet?”
- Interrupts: Ringing Telephone

Interrupts

Basic idea:

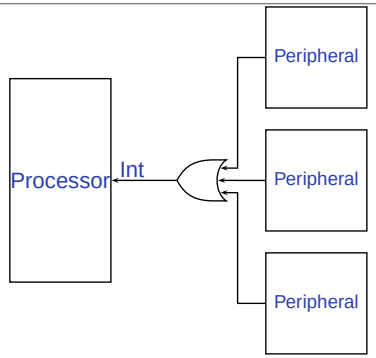
1. Peripheral asserts a processor’s interrupt input
2. Processor temporarily transfers control to interrupt service routine
3. ISR gathers data from peripheral and acknowledges interrupt
4. ISR returns control to previously-executing program

Many Different Interrupts



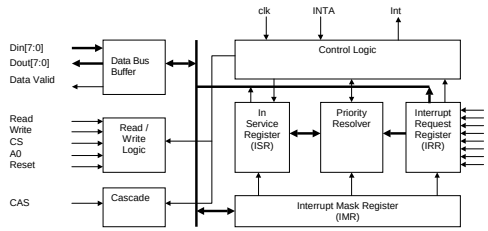
What’s a processor to do?

Interrupt Polling



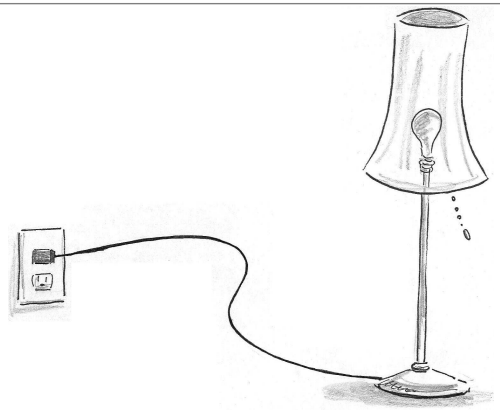
Processor receives interrupt
ISR polls all potential interrupt sources

Intel 8259 PIC



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

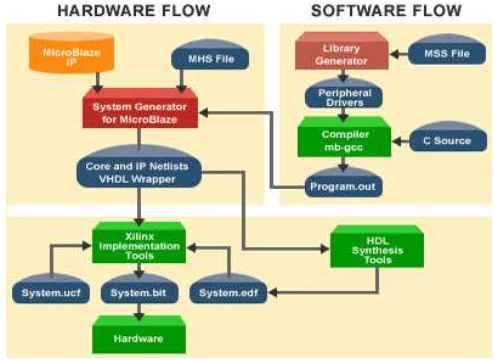
Debugging Skills



The Edwards Way to Debug

1. Identify undesired behavior
2. Construct linear model for desired behavior
3. Pick a point along model
4. Form desired behavior hypothesis for point
5. Test
6. Move point toward failure if point working, away otherwise
7. Repeat #4–#6 until bug is found

The Xilinx Tool Chain



The .mhs File

Xilinx *platgen* uses this to piece together the netlist from library components. Excerpt:

```
PORT VIDOUT_GY = VIDOUT_GY, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BCB = VIDOUT_BCB, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
END
```


The .mss File

Used by Xilinx *libgen* to link software. Excerpt:

```
BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER EXECUTABLE = hello_world.elf
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
 PARAMETER DEFAULT_INIT = EXECUTABLE
 PARAMETER STDIN = myuart
 PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 1
END
```


The .ucf file

Pin assigments and other global chip information.

```
net sys_clk period = 18.000;
net pixel_clock period = 36.000;

net VIDOUT_GY<0> loc="p9" ;
net VIDOUT_GY<1> loc="p10";
net VIDOUT_GY<2> loc="p11";

net VIDOUT_BCB<0> loc="p42";
net VIDOUT_BCB<1> loc="p43";
net VIDOUT_BCB<2> loc="p44";

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
```


Lab 1

Write and execute a C program that counts in decimal on the two 7-segment displays on the XSB-300E.

We supply

- A hardware configuration consisting of a processor, UART, and
- A simple memory-mapped peripheral that latches and displays a byte controlling each segment of the displays.
- A skeleton project that compiles, downloads, and prints “Hello World” through the serial debugging cable.

Your Job

Write and test C code that

- Counts
- Converts the number into arabic numerals on the display
- Transmits this to the display

Goal: Learn basics of the tools, low-level C coding, and memory-mapped I/O.

Debugging Lab 1

- Examine build error messages for hints
- “make clean” sometimes necessary
- Call *print* to send data back to the host
- Run Minicom on /dev/ttyS0 (9600 8n1) to observe output