Homeworks

- Grades of HW1 have been posted on Canvas
- HW2 will be due on 10/7 before class

Recap from previous classes

- Design Methodology
  - Non-functional requirements: *real-time, power, memory, cost*
  - Alternative technologies
    - Microprocessor (dominant player), ASIC, FPGA
- Microprocessors
  - Von Neumann vs. Harvard
  - RISC vs. CISC
- I/O, interrupts, bus
  - Busy-wait, interrupts, buffer, priorities and vectors
- Caches and Memory
  - Memory access time, replacement strategies (LRU, random)
  - Cache organizations (direct-mapped, fully/set associative)
  - Segment/page-based memory management

ECE 1175
Embedded Systems Design

Embedded Computing Platform

Wei Gao

Outline

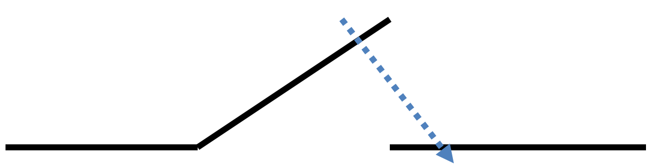
- Typical I/O devices
- Embedded computing platform
  - Put everything together
- Hardware/software development
- Debugging and testing

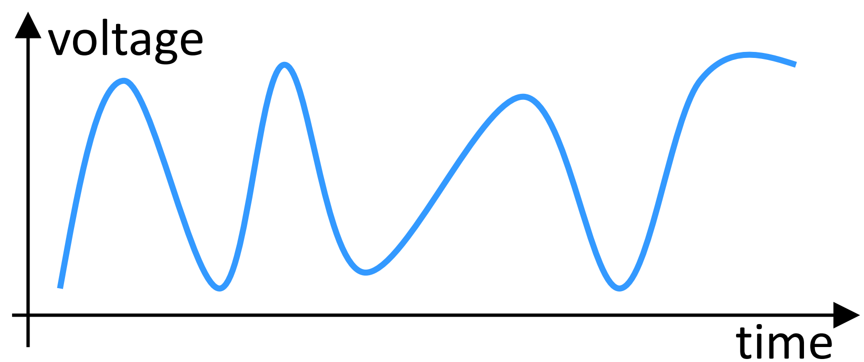
Typical I/O Devices

- Keyboards
- Serial links
- LEDs
- Displays
  - Cathode ray tube (CRT)
  - Liquid crystal display (LCD) - Touchscreens
  - Plasma, etc.
- A/D and D/A converters

Keyboards

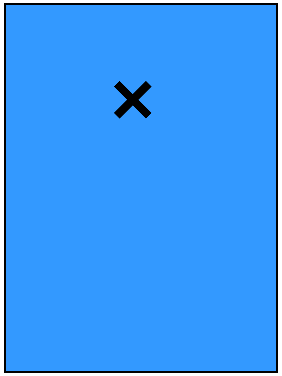
- A keyboard is an array of switches
  - A mechanical contact makes an electrical circuit
- Switch debouncing
  - A switch must be debounced to eliminate multiple contacts caused by mechanical bouncing.
  - A hardware debouncing circuit can be built using a one-shot timer

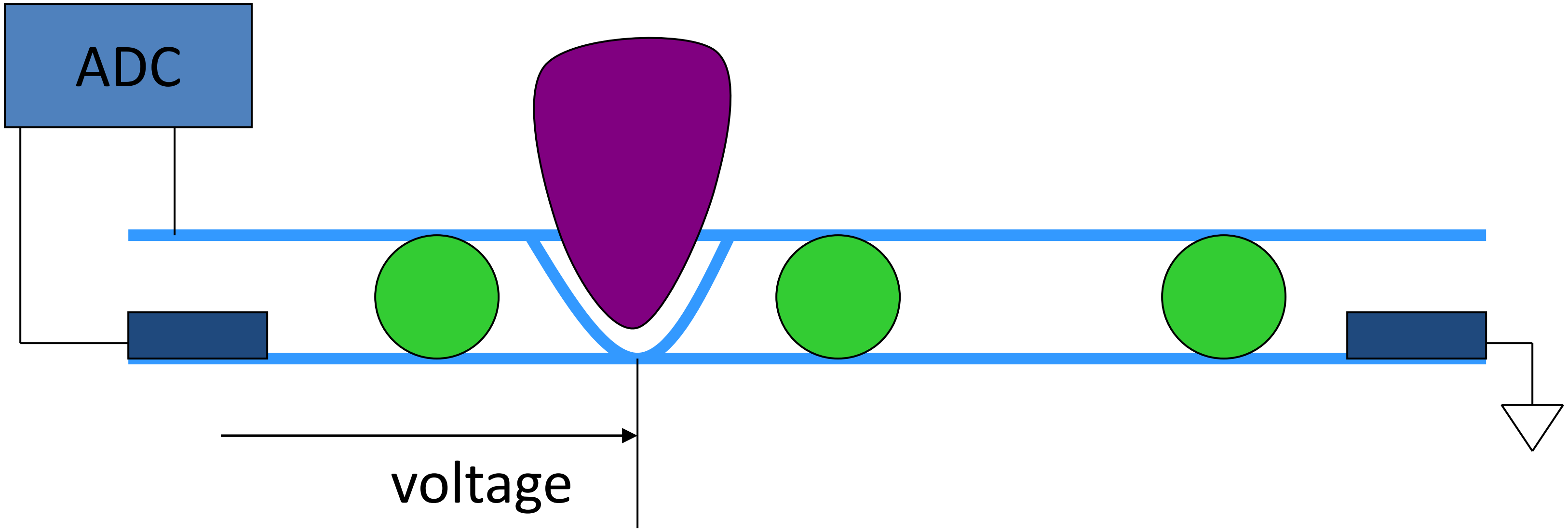




Touchscreens

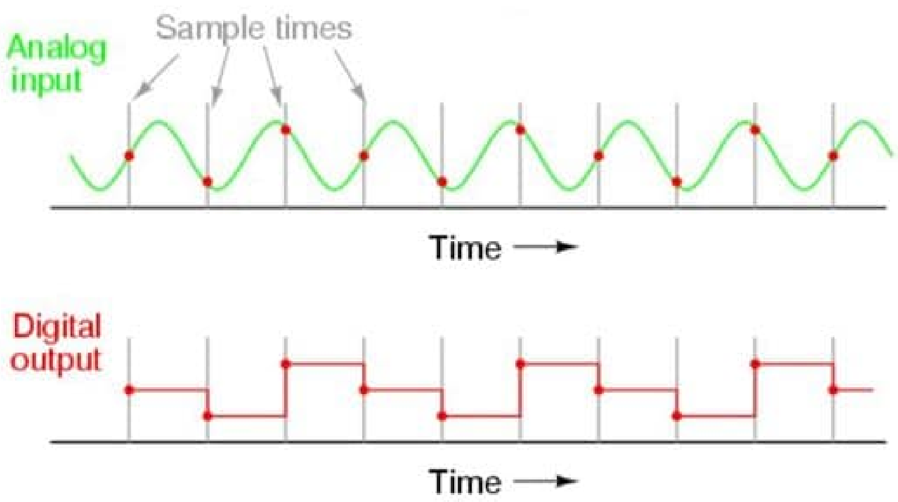
- Includes both input and output device.
  - Registers the position of a touch to its surface
- Input device is a two-dimensional voltmeter:
  - Two conductive sheets separated by spacer balls
  - A voltage is applied across the sheet upon a touch



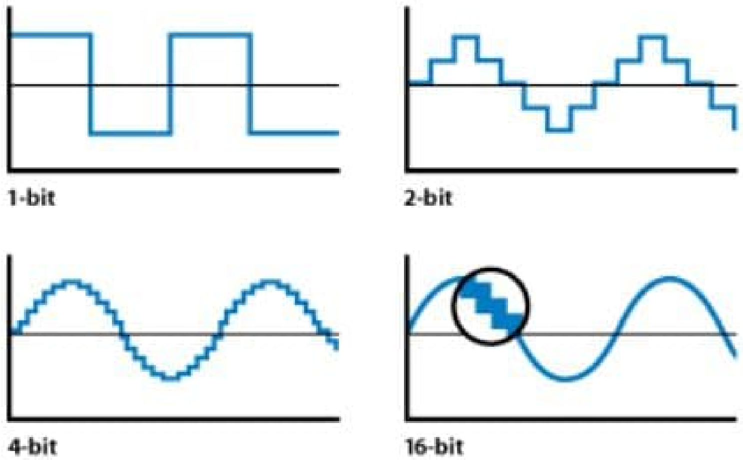


A/D Converter

- Periodic sampling
  - Rate: higher than Nyquist frequency

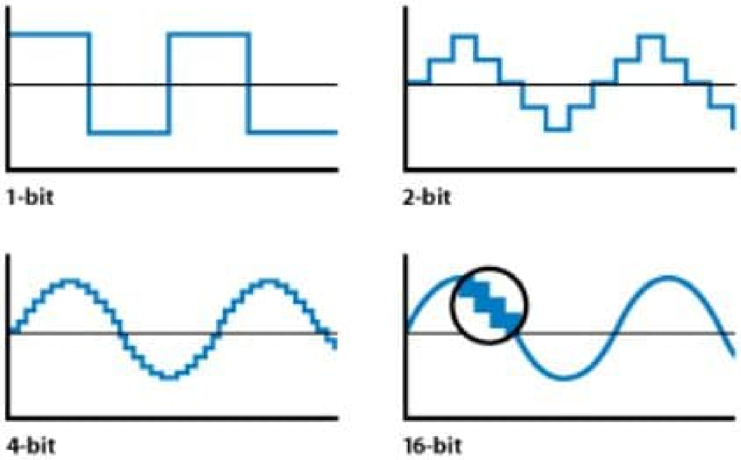


- Resolution: # of bits
  - The number of possible digital outputs

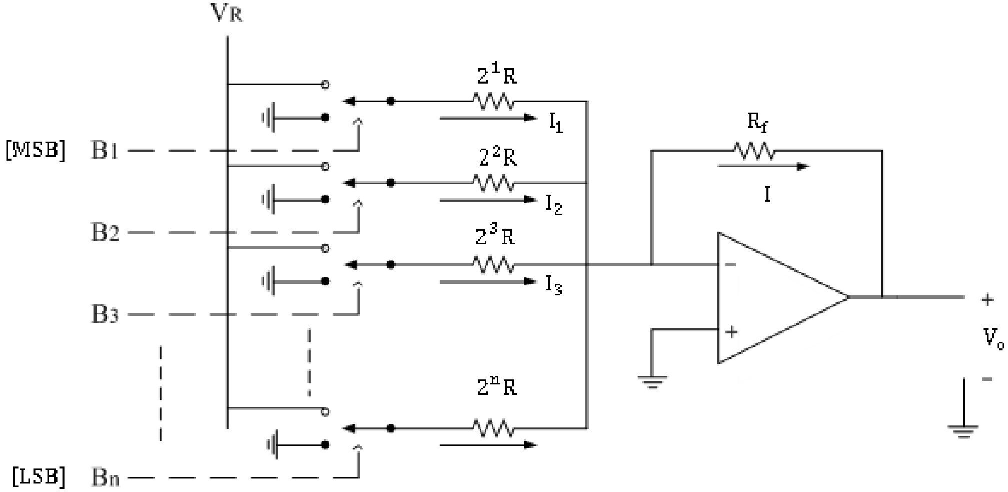


D/A Converter

- The reverse procedure



- The analog circuitry



Embedded Computing Platform

- Designing with microprocessors
- Components that we have learned
  - Microprocessors, caches, memory, CPU bus, I/O devices, interrupt mechanism

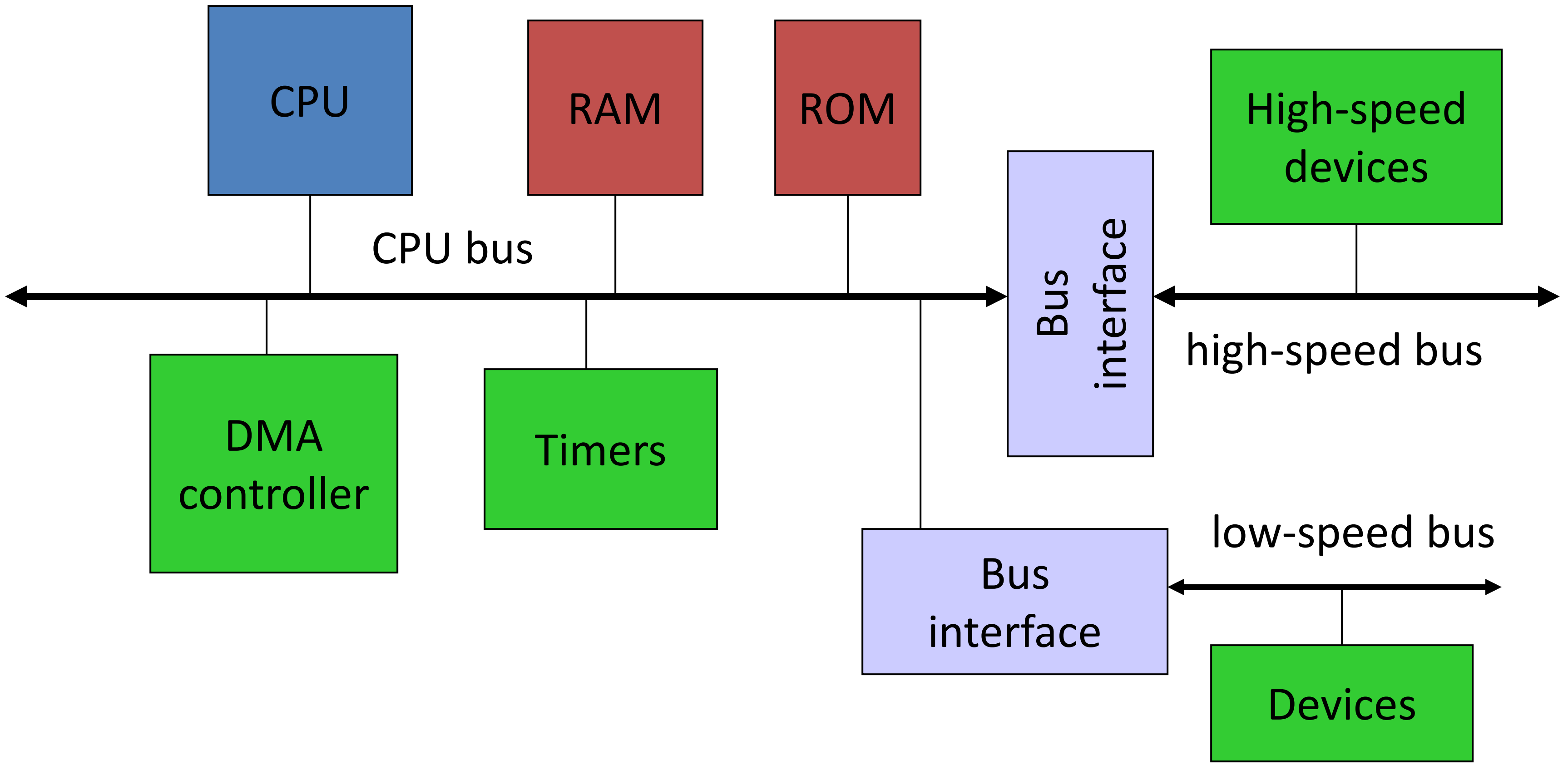
- How to put them all together?
  - As an embedded computing platform
  - Architectures and components:
    - Hardware.
    - Software;
  - Debugging and testing.

Hardware Platform Architecture

- CPU:
  - Most important choice but cannot be made without considering the application software
- Bus:
  - Closed tied to CPU; enough for required data bandwidth
- Memory:
  - Total size? ratio of ROM and RAM, SRAM vs. DRAM
- I/O devices:
  - Networking, sensors, actuators, etc.

How big/fast must each one be? Identify bottleneck?

Typical PC Hardware Architecture



Typical CPU Bus

- ISA (Industry Standard Architecture)
  - Original IBM PC bus, low-speed by today's standard.
  - Primarily used for low-speed devices and backward compatibility,
  - About 2Mbps
- PCI (Peripheral Component Interconnect)
  - Dominant high-performance system bus
  - Standard for high-speed interfacing, up to 264/524 Mbps
- High-speed serial buses
  - ISA and PCI use wide buses with many data/address/control bits
    - High-cost interface and complicated physical connection to the bus
  - Relatively low-cost serial interface with high speed.
  - USB (Universal Serial Bus), 480Mbps (2.0), 12Mbps (1.1).
  - Firewire (IEEE 1394), 400Mbps (1394a), 800Mbps (1394b).

Hardware and Software Architectures

Hardware and software are intimately related:

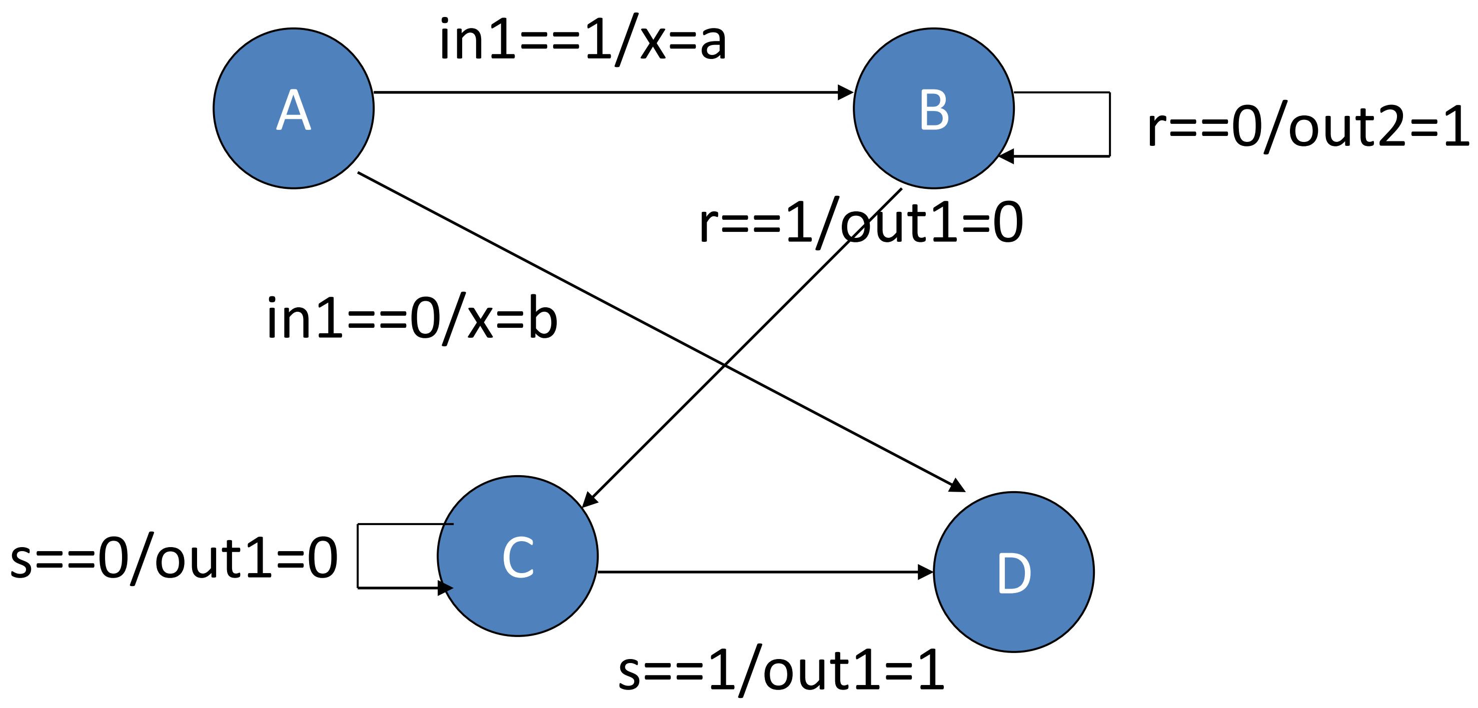
- Software doesn't run without hardware;
- How much hardware you need is determined by the software requirements:
  - Speed;
  - Memory.
- Special-purpose hardware often consumes much less power.

Software Architecture

- Functional description must be broken into pieces:
  - Division among people;
  - Conceptual organization;
  - Performance;
  - Testability;
  - Maintenance.
- Need to break the design up into pieces to be able to write the code

Software State Machine

- State machine keeps internal state as a variable, changes state based on inputs.
- Examples: control-dominated code; reactive systems.

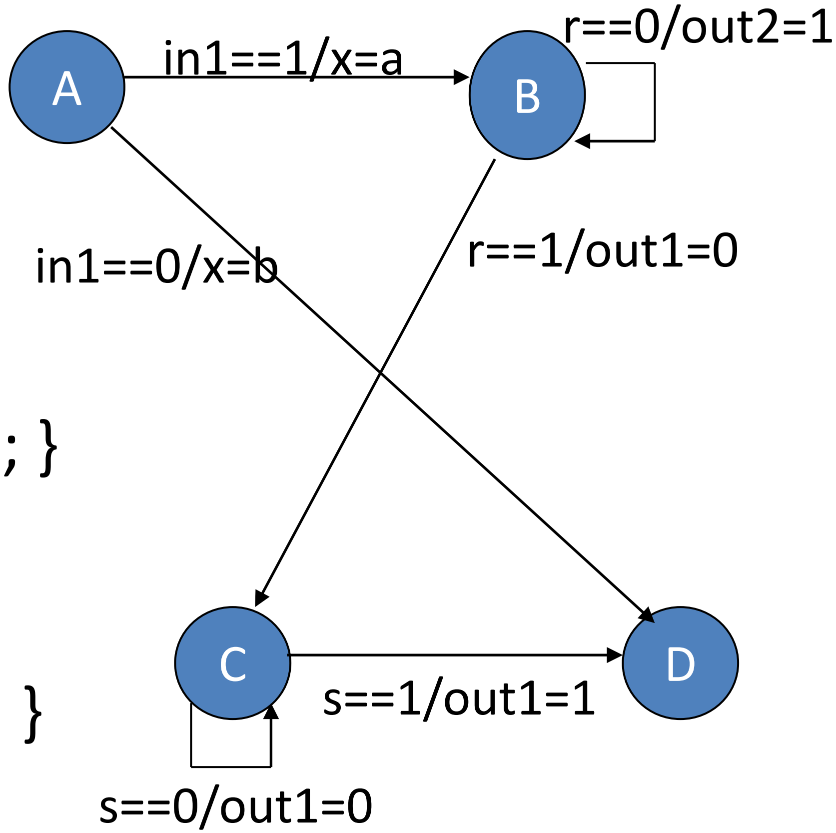


C Code Structure

- Current state is kept in a variable.
- State table is implemented as a switch.
  - Cases define states.
  - States can test inputs.
- Switch is repeatedly evaluated in a while loop.

C Implementation of State Machine

```
while (TRUE) {
    switch (state) {
    case A: if (in1==1)
                  { x = a; state = B; }
              else { x = b; state = D; }
              break;
    case B: if (r==0)
                  { out2 = 1; state = B; }
              else { out1 = 0; state = C; }
              break;
    case C: if (s==0)
                  { out1 = 0; state = C; }
              else { out1 = 1; state = D; }
              break;
}
```

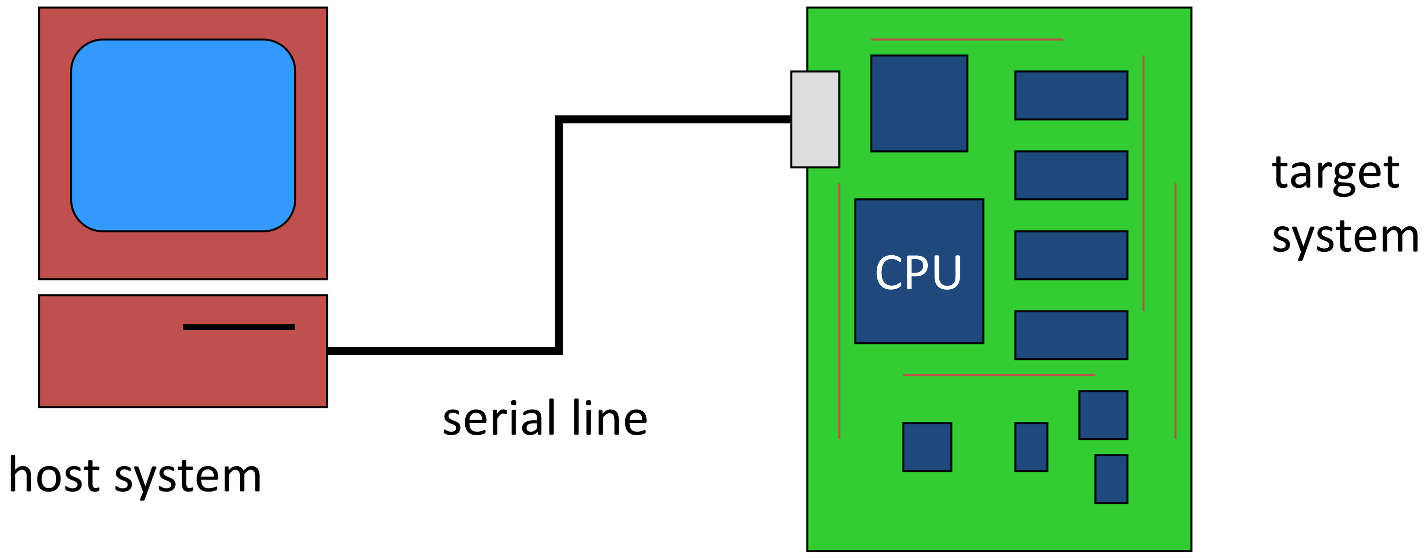


Software Design Techniques

- Want to develop as much code as possible on the host system (usually the PC):
  - More friendly programming environment;
  - Easier debugging.
- Testability concern
  - May need to devise software stubs to allow testing of software elements without the full hardware/software platform.
  - How to test your project?

Development and Debugging of Embedded Systems

- Development environments
  - Use a host system to prepare software for target system:
  - Host should (1) load programs to the target; (2) start and stop program execution on the target; and (3) debug the program



Debugging Embedded Systems

- Challenges:
  - Target system may be hard to observe;
    - e.g., only three LEDs on the motes
  - Target may be hard to control;
    - e.g., packets may get lost in wireless communication
  - May be hard to generate realistic inputs;
    - e.g., fire event, plume, etc
    - Need to emulate the events from host systems by injecting packets
  - Setup sequence may be complex.
    - e.g., need additional tools and environment

Common Debugging Techniques

- Compiling and simulating the code on a PC
- Debugging tool through serial port
  - Using MSP430 debugging tool with GDB
- Breakpoints
  - A breakpoint allows the user to stop execution, examine system state, and change state
- LEDs as debugging devices
- Microprocessor in-circuit emulator (ICE)
  - Need special version of the microprocessor that allows its internal registers to be read out
- Logic analyzer
  - Sample many signals and display only 0, 1 and changing
- Exercise code through hardware/software co-verification

How to Exercise Code

- Run on host system.
  - To simulate the target system, e.g., make pc
- Run on target system.
  - Use debugging tool to monitor execution
- Run in instruction-level simulator.
- Run on cycle-accurate simulator.
  - Simulate the hardware operation within clock-cycle accuracy
- Run in hardware/software co-simulation environment.
  - Injecting a packet to the network of real motes

Software vs. Hardware Testing

- Implementation testing
  - No fault model: don’t know exactly what potential faults we are looking for
  - We verify the implementation, not the manufacturing.
    - e.g., compare to the video online
  - Simple tests (e.g., ECC) work well to verify software manufacturing.
- Hardware requires manufacturing tests in addition to implementation verification.

Manufacturing Testing

- Goal: ensure that manufacturing produces defect-free copies of the design.
- Different from implementation testing
  - Know particular faults may appear
  - Assume design is correct
  - Look for variations between the design and copies
- Challenge: maximize confidence while minimizing testing cost.
  - Shortest test to determine if a particular fault appears

Summary

- Typical I/O devices
  - Timer, counter, keyboards, touchscreens
- Embedded computing platform
  - Put everything together
- PC as a platform
- Hardware/software development
  - State machine implementation
- Debugging and testing