Embedded systems

- there are many times an IBM-PC is an overkill solution
- an IBM-PC is a general purpose computer, sometimes a more specific computer is cost effective
- a specific computer executing a specific operating system might be the solution and in a minimal case we might choose to use an embedded operating system
 - or embedded system if we include the application

Characteristics of an embedded system

- these systems can be tiny
 - could be < 1KBytes
- only use software which you need
 - for example many times it is possible to implement an embedded system using no interrupts
 - choice, it might be
 - easier to implement
 - guarentee a hard realtime performance for time critical applications

Case examples: embedded systems built using the ATMega328p

- we will look at building a basic computer running a tiny embedded system
 - flashing LED program
- many, many other examples in which the ATMega328p can be used
 - network on/off switch
 - cruise controller for an electic bicycle
 - amplifier controller
 - tiny web server and LCD panel
 - Arduino!

Why use ATMega processsors?

- support within gcc in mature (it has support ATMega microprocessors for about two decades)
 - the ATMega series of microprocessors have very similar instructions
- harvard risc architecture

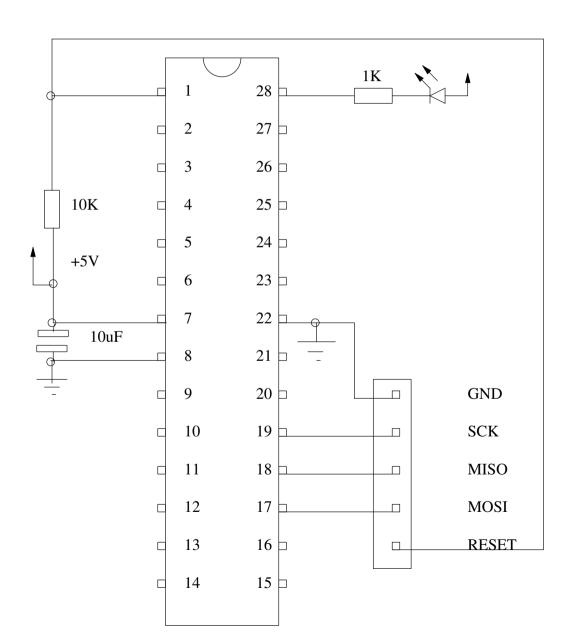
Atmel ATMega series of microprocessors

- the number of components to make a minimal system is tiny
- they are also extremely easy to interface to peripherals
 - A->D, D->A, pwm (servo and motor control) etc
 - multiple hardware timers etc

Atmel 328p

- is an 8-bit AVR RISC-based microcontroller (some of its features include):
 - 32KB flash memory
 - 1024B EEPROM, 2KB SRAM
- 23 general purpose I/O lines
 - 32 general purpose working registers
 - three flexible timer/counters with compare modes
 - internal/external interrupts, a 6-channel 10-bit A/D converter
 - programmable watchdog timer with internal oscillator
 - 2 PWM channels (ie control two servos) in hardware
- many features omitted for sake of brevity

Simple computer flashing a LED with the ATMega328p



Code for the flashing LED

```
MODULE flashled;
PROCEDURE Turn (on: BOOLEAN);
BEGIN
   IF on
   THEN
     (* turn LED on *)
     ASM VOLATILE ("cbi 8,5");
  ELSE
     (* turn LED off *)
    ASM VOLATILE ("sbi 8,5");
   END
END Turn ;
(*
   InitLed - initialize pin 0 as an output
*)
PROCEDURE InitLed;
BEGIN
  ASM VOLATILE ("sbi 7,5")
END InitLed;
```

Code for the flashing LED

```
CONST
Delay = 400;

VAR
i, j: CARDINAL;
```

Code for the flashing LED

```
BEGIN
   InitLed ;
   Turn (FALSE) ;
   LOOP
      FOR i := 0 TO Delay DO
         FOR j := 0 TO Delay DO
            ASM VOLATILE ("nop")
         END
      END ;
      Turn(TRUE) ;
      FOR i := 0 TO Delay DO
         FOR j := 0 TO Delay DO
            ASM VOLATILE ("nop")
         END
      END ;
      Turn (FALSE)
   END
END flashled.
```

Cruise controller for an electric bicycle

- uses PWM device to control the power delivered to the electric motor
- uses a A to D device to take input from the throttle (potentiometer)
- uses several output pins to control status LEDs
- uses input pins for wheel movement sensing
- due to the hardware support inside the Atmega328p the software is extremely simple
 - no need for interrupt service routines
 - no need for separate processes

Amplifier control embedded system

- uses the Atmega328p to
 - turn on the +-12v power
 - turn on the +-9v power
 - connect the speakers after 2 seconds (speaker protection)
 - uses the A to D device to take input from a potentiometer to select input source

Amplifier control embedded system

- uses input lines to detect push button
 - three pulses turns it off
 - two pulses turns off the speakers
 - one pulse resets the power save timer
- software is a simple C program which controls hardware directly

Conclusion

- embedded systems come in many sizes and the examples given here are tiny applications
- embedded systems might range up to and including the Linux kernel (with various scheduling and device driver changes)