

Embedded Systems

- in an operating systems level 3 course it would be remiss not to discuss embedded systems
- an embedded system is typically a low cost, low memory, low power hardware device which is controlled by a microprocessor
- typically the microprocessor has all the flash memory, ram, i/o all on board
 - for example the AVR series of microprocessors from Atmel
 - Atmel produce literally hundreds of microprocessors of different specifications
 - you need to choose the correct one!

Small sample of AVR microprocessors and features

Device Family	Applications	Key Parameters
32-bit AVR UC3	General Purpose	16-512KB Flash 48-144 pins Up to 66 MHz
8/16-bit AVR XMEGA	General Purpose	1.5 MIPS/MHz 16-384KB Flash 44-100 pins Up to 32 MHz
8-bit megaAVR	General Purpose	1.0 MIPS/MHz 4-256KB Flash 28-100-pins Up to 20 MHz
8-bit tinyAVR	Small space	1.0 MIPS/MHz 0.7V operation 0.5 - 8KB Flash 6-32 pins Up to 20MHz 1.0 MIPS/MHz

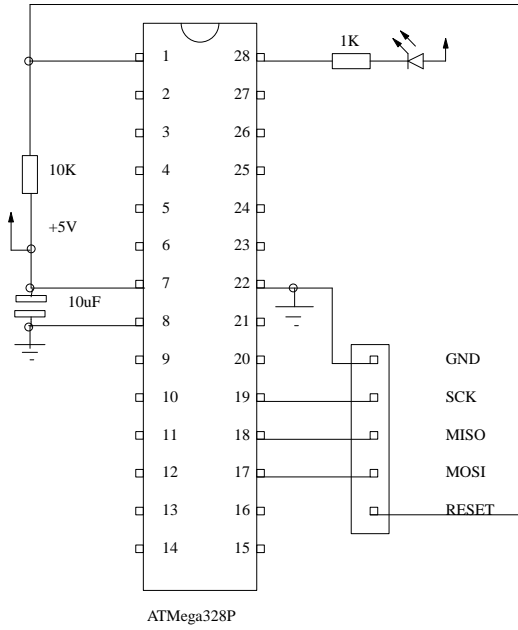
Atmel ATMega series of microprocessors

- the number of components to make a minimal system is tiny
- these devices are RISC microprocessors employing a Harvard processor architecture
 - gcc supports these devices!
- they are also extremely easy to interface to peripherals
 - A->D, D->A, pwm (servo and motor control) etc
 - multiple hardware timers etc

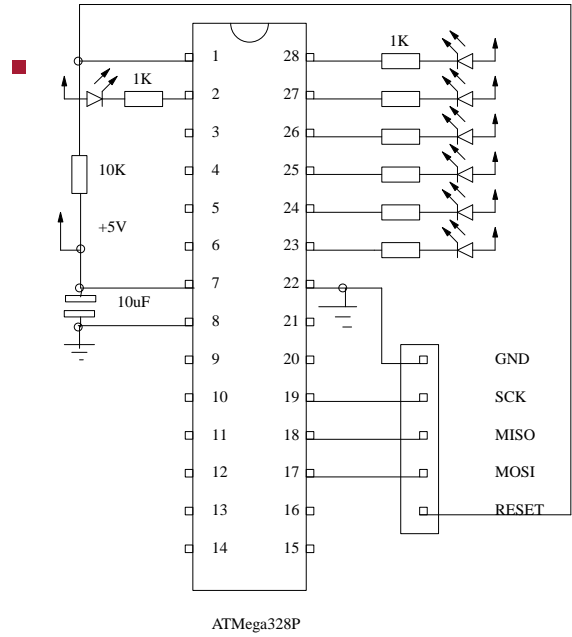
Atmel 328p

- is an 8-bit AVR RISC-based microcontroller (some of its features include):
 - 32KB flash memory with read-while-write capabilities
 - 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
- many features omitted for sake of brevity

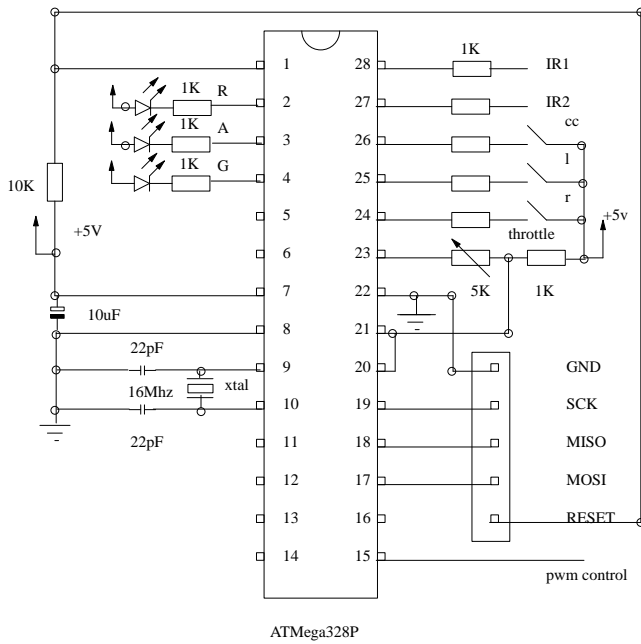
Flashing LED with ATmega328p



Electric die with ATmega328p



E-bike cruise controller with ATmega328p



Flashing LED software

- system contains a minimal device driver!

```
static void turn (int on)
{
  if (on)
    (* turn LED on *)
    asm volatile ("cbi 0x15, 5");
  else
    (* turn LED off *)
    asm volatile ("sbi 0x15, 5");
}
```

Flashing LED software

```

/*
 * initLed - initialize pin 0 as an output
 */
static void initLed (void)
{
    asm volatile ("sbi 0x14, 5");
}

```

Flashing LED software

```

#define Delay 200
main (void)
{
    int i, j;

    initLed();
    turn(FALSE);
    while (TRUE) {
        for (i=0; i++; i<Delay)
            for (j=0; j++; j<Delay)
                asm volatile ("nop");
        turn(TRUE);
        for (i=0; i++; i<Delay)
            for (j=0; j++; j<Delay)
                asm volatile ("nop");
        turn(FALSE);
    }
}

```

Cruise controller

- cruise controller for an E-bike
- inputs: 5K pot (throttle), left/right break switch
 - two infrared switches which measure rotation speed of the motor and bicycle wheel
- bicycle wheel speed is dependent upon throttle position
 - if the bicycle wheel rotation \geq throttle position then motor is off
 - cruise controller can also dampen acceleration
 - not apply any motor until the bicycle is moving at say 3mph
 - regulate speed to a max of 15mph

Servo specifications

- typically a servo has three wires: red, black, white
 - black (GND)
 - red (+4.8v..6.0v)
 - white signal
- signal is normally a square wave with a period of
 - 1.0ms full left
 - 1.5ms middle
 - 2.0ms full right

Configuring the ATmega8/168/328 to control a Servo

- use the phase and frequency correct mode of the on chip timer
 - to generate square wave of period between 1.0ms..2.0ms
 - frequencies of between 1000Hz..500Hz

Motor speed test code

```
#define F_CPU 16000000UL
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>
```

Motor speed test code

```
#define ePRESCALER8
#if defined(ePRESCALER8)
# define eICR1value 39936
# define ePRESCALEBITS (_BV(CS11))
# define eLOW 2048
# define eHIGH 3968
# define eDELAYms 3
#endif
```

Motor speed test code

```
#if defined(ePRESCALER256)
# define eICR1value 1248
# define ePRESCALEBITS (_BV(CS12))
# define eLOW 64
# define eHIGH 124
# define eDELAYms 100
#endif

#if defined(ePRESCALER1024)
# define eICR1value 1248*4
# define ePRESCALEBITS (_BV(CS10) | _BV(CS12))
# define eLOW 16
/* not a good choice as eHIGH is not exact,
 * 124/4 is not an integer value
 */
# define eHIGH 31
# define eDELAYms 400
#endif
```

Initialise the PWM output devices

```

/*
 * initServo - initialises timer 1 for fast PWM operation
 *             servo is enabled on B1.
 */

void initServo (void)
{
  /* initialize TMR1 (PWM) */
  /* clear on compare, fast PWM, TOP=ICR1 (WGM13/WGM12 in
  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM11);
  /* set prescaler value */
  TCCR1B = _BV(WGM12) | _BV(WGM13) | ePRESCALEBITS;
  ICR1 = eICR1value;
  OCR1A = -1; /* off */
  OCR1B = -1; /* off */
  DDRB |= _BV(1) | _BV(2); /* output on PB1 and PB2 */
}

```

Main device driver :-)

```

/*
 * set_servo1 - sets the pwm length for servo1.
 *             Value should be between eLOW..eHIGH
 */

void inline set_servo1 (int pwm)
{
  OCR1A = pwm;
}

/*
 * set_servo2 - sets the pwm length for servo1.
 *             Value should be between eLOW..eHIGH
 */

void inline set_servo2 (int pwm)
{
  OCR1B = pwm;
}

```

Motor test code ramp up and down

```

main ()
{
  int i;
  initServo ();

  while (TRUE) {
    for (i=eLOW; i<=eHIGH; i++) {
      _delay_ms (eDELAYms);
      set_servo1 (i);
    }
    for (i=eHIGH; i>=eLOW; i--) {
      _delay_ms (eDELAYms);
      set_servo1 (i);
    }
  }
}

```

Compiling

```
$ avr-gcc -mmcu=atmega168 pwmtest.c -o cflashled.elf
```

Linking

■ `$ avr-objcopy -R .eeprom -O ihex cflashled.elf cflashled.i`

Loading the code

- note that the actual device is the ATmega328p
- we compiled using the `-mmcu=atmega168` switch on `gcc` since `gcc-4.1.2` does not know about the newer ATmega328p
 - fortunately they are upward compatible
 - but the top 16K flash ram will not be used
 - should really use `gcc >=4.6.x`

■ `$ sudo avrdude -p m328p -U lfuse:w:0xff:m -U hfuse:w:0xd9 -U efuse:w:0x07:m -c stk500v2 -e -v -U flash:w:flas`

Simulation of the E-bike transmission under computer cruise control

- <http://floppsie.comp.glam.ac.uk/download/avi/3-low-motor.ogv>