The AVR Microcontroller

Introduction

Application and programmer boards

WinAVR

Basic I/O

ADC, timers

USART, LCD

Mehul Tikekar
Chiraag Juvekar

Electronics Club

May 26, 2010

What is a microcontroller?

- It is essentially a small computer
- Compare a typical microcontroller (uC) with a typical desktop

	ATmega16	Typical desktop
Clock frequency	16MHz	3GHz
CPU data size	8 bits	32 bits
RAM	1KB	1GB
ROM	16KB	160GB
I/O	32 pins	Keyboard, monitor
Power consumption	20mW	65W

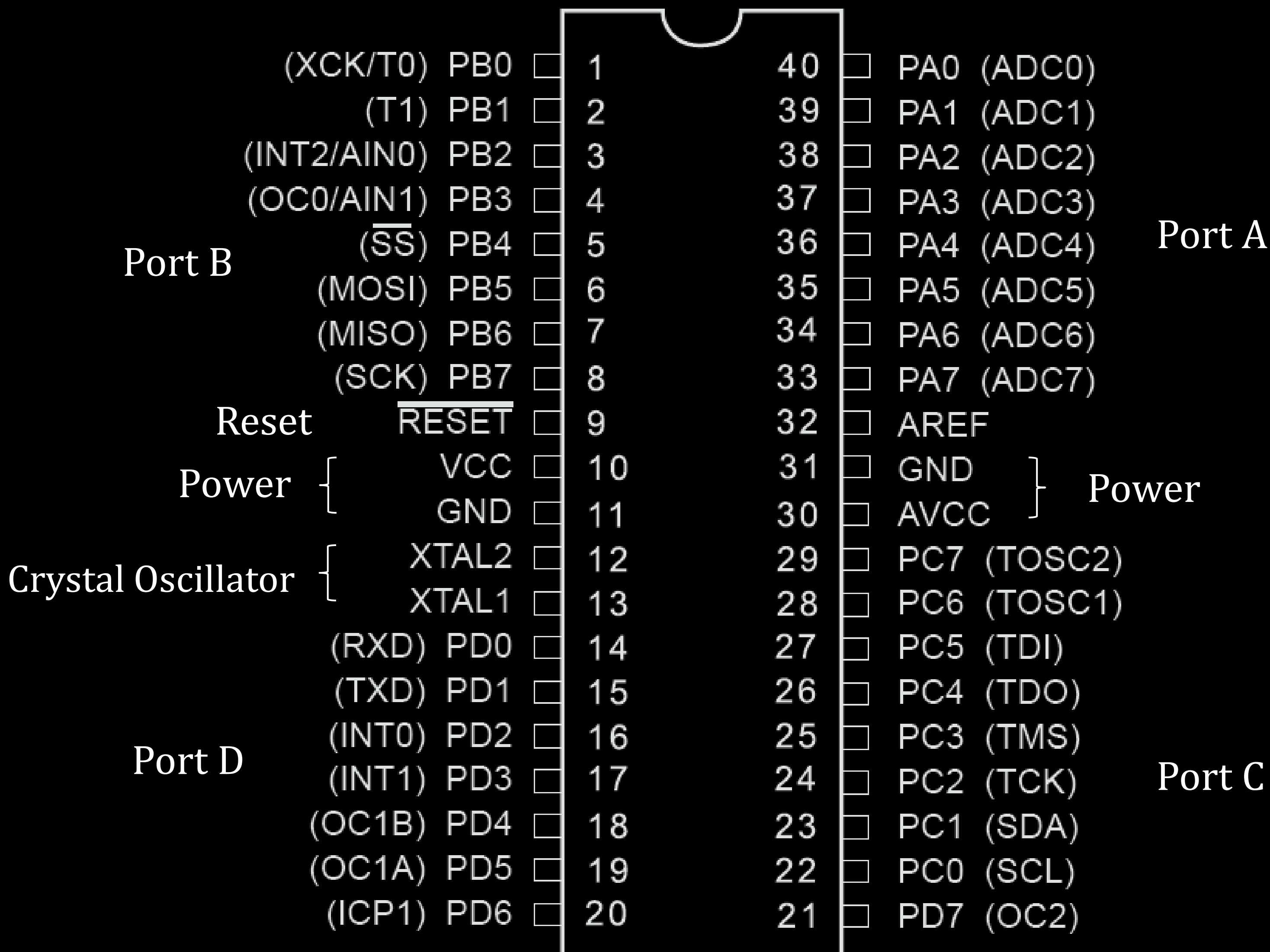
Why use a micro-controller?

- It is programmable. It is fun.
- A code (typically written in C) decides what it does
- Easier to write a code than design and make a custom circuit for complex jobs
- e.g. In a micromouse, a single uC controls the motors, finds distance from the walls, explores and solves the maze
- The same muC can be used in hundreds of applications
- http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/ has 407 projects just on AVR microcontrollers as of 2009

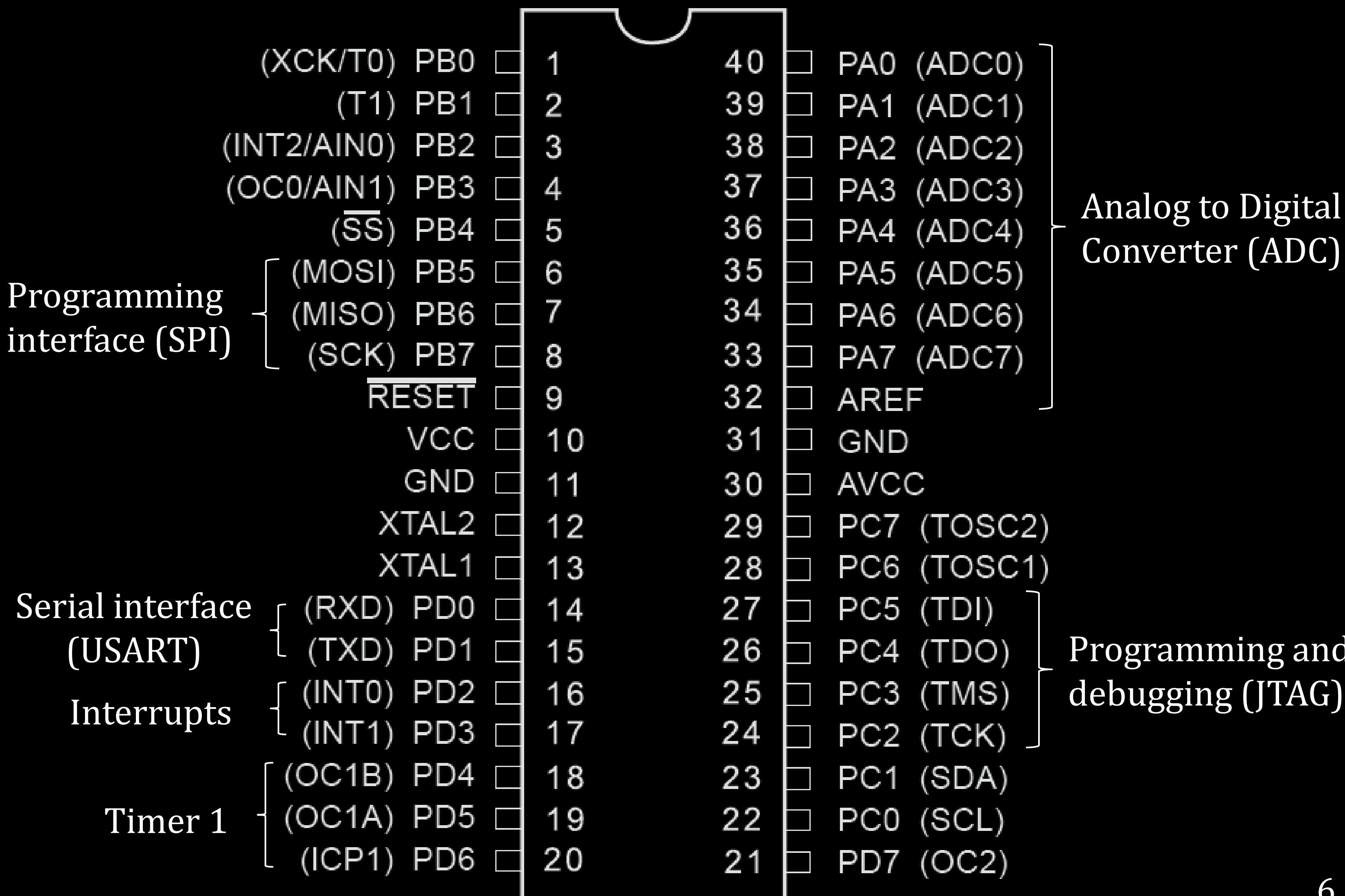
AVR microcontroller

- Lots of micro-controller families
  - 8051, PIC, AVR, MSP430, ARM, etc.
- http://www.instructables.com/id/How-to-choose-a-MicroController/
- AVR: Cheap, easy to use, fast and lots of features
- ATmega16:
  - 16KB flash memory
  - 1KB SRAM
  - Up to 16MHz clock
  - Four 8-bit I/O ports
  - ADC, timers, serial interface, etc.
  - 40 pin DIP, VDD = 5V

ATmega16 pin diagram



Alternate pin functions: Peripherals



A typical code for the microcontroller

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
   int i;
   DDRB = 255;
   for(i=0; i<100; i++)
   {
        PORTB  = i;
        _delay_ms(250);
   }
   return 0;
}
```

The next steps:

1. Compile
2. Program
3. Run

Let’s get started

Software

- WinAVR

Hardware

- Programmer board
- Application board

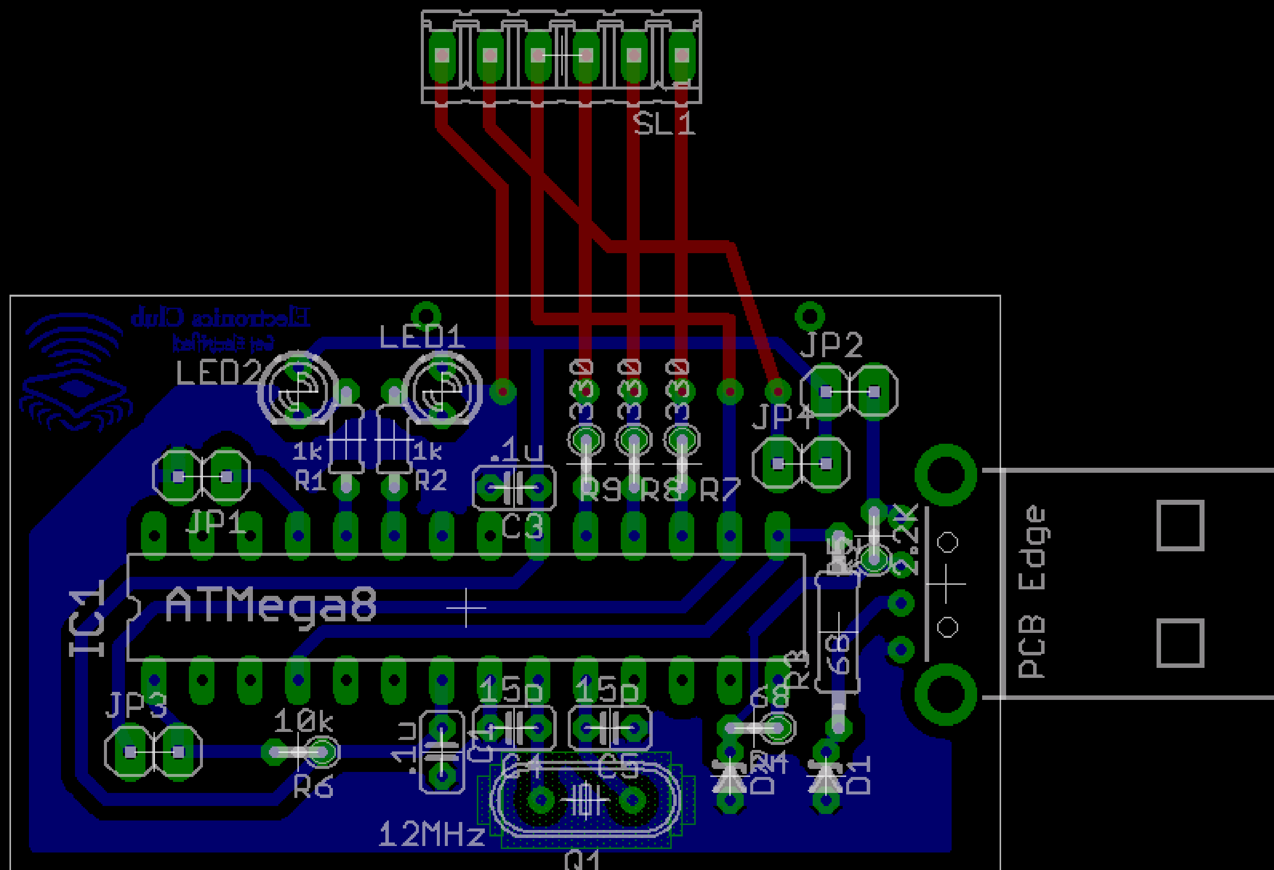
WinAVR

A complete package for Windows

- Programmer's Notepad (user interface)
- avr-gcc (C/C++ compiler)
- Mfile (makefile generator)
- avrdude (programmer software)

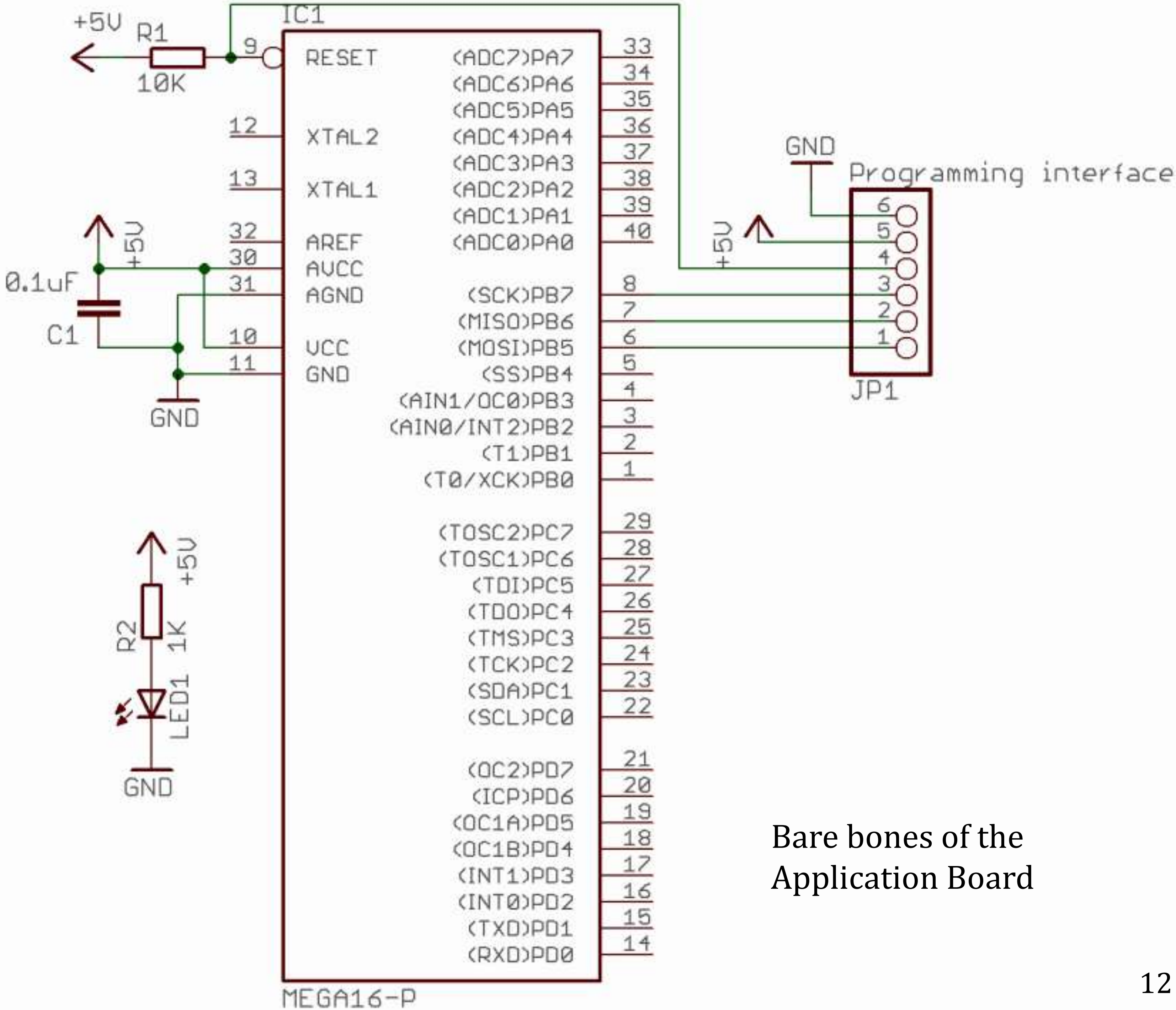
Totally free! : http://winavr.sourceforge.net/

USBasp – USB Programmer



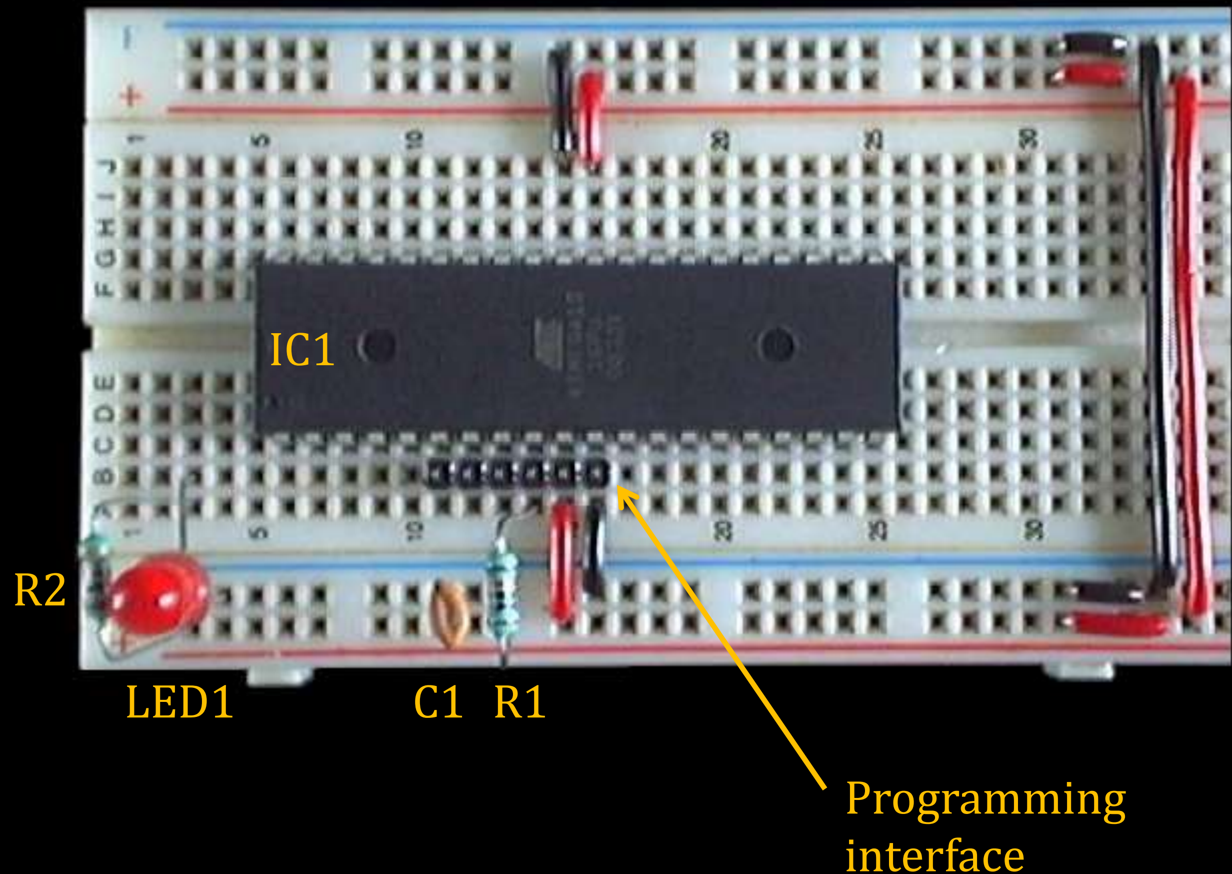
www.fischl.de/usbasp for more information and usb driver

USBasp

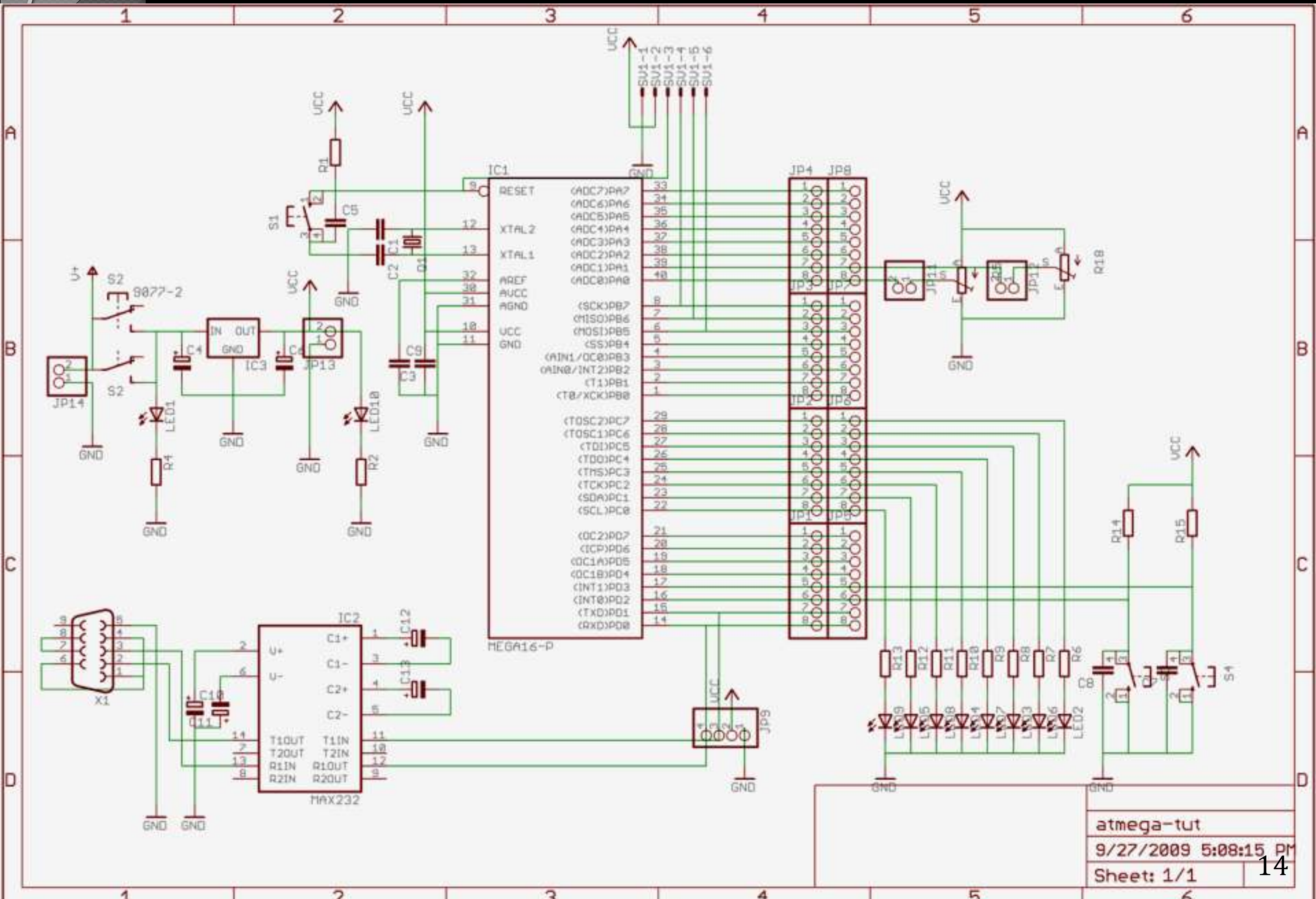


Bare bones of the Application Board

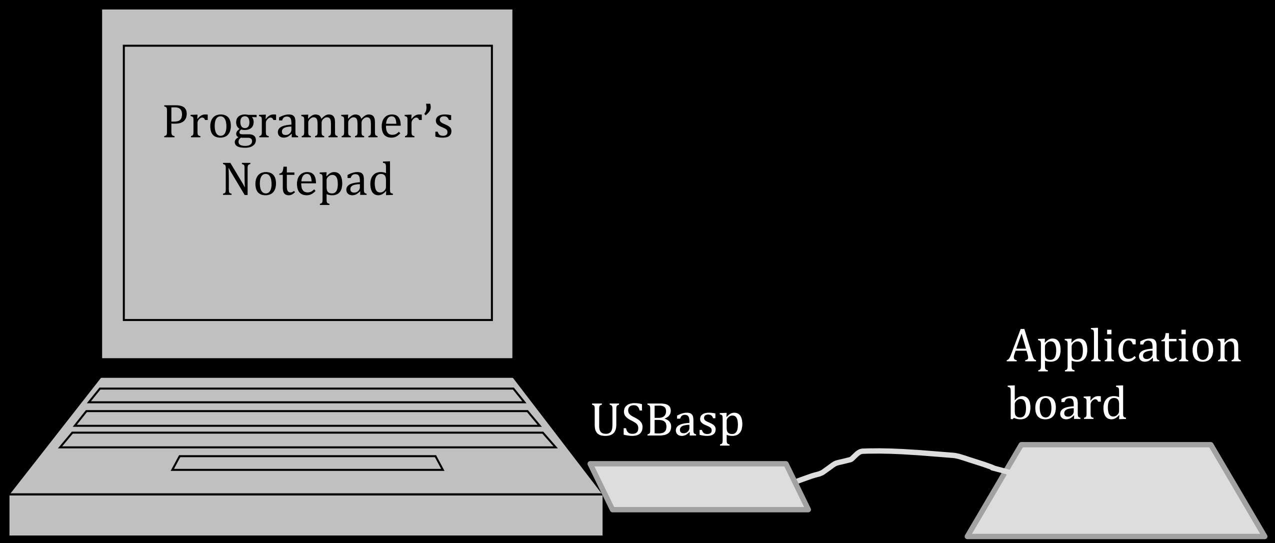
On a bread-board



Complete application board



The hardware setup



Resources and Links

- The ATmega16 datasheet – a 358 page bible
- Tutorials and sample codes:
  1. http://www.avrtutor.com
  2. http://winavr.scienceprog.com
  3. http://kartikmohta.com/tech/avr/tutorial
- Atmel application notes: projects, tutorials, code-examples, miscellaneous information
  http://www.atmel.com/dyn/products/app_notes.asp?family_id=607
- Forums: http://www.avrfreaks.net
- Advanced projects:
  1. http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects
  2. http://www.obdev.at/products/vusb/projects.html

Configuring the microcontroller before running it the first time

- Fuse bytes : high and low
- Program them once before you start using the micro-controller
- Disable JTAG to free up PORTC for normal use
- Set the correct clock clock option
- With the hardware set up, run in Command Prompt :
- For 1MHz internal clock:

```
avrdude -c usbasp -P usb -p m16 -U hfuse:w:0xd9:m -U lfuse:w:0xe1:m
```

- For 16MHz external crystal:

```
avrdude -c usbasp -P usb -p m16 -U hfuse:w:0xc9:m -U lfuse:w:0xef:m
```

- Refer to datasheet sections on “System clock and clock options” and “Memory programming” for other clock options and their settings.
- Setting the wrong fuse values may render the uC unusable

Hello world

- Blink an LED
- Choose a port and a pin
- In the code
  1. Set pin direction to output
  2. Set pin output to high
  3. Wait for 250ms
  4. Set pin output to low
  5. Wait for 250ms
  6. Go to 2

Port B

Relevant registers :
DDR – set pin data direction
PORT – set pin output
PIN – read pin input

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

blink.c

```c
#include <avr/io.h>      // contains definitions for DDRB, PORTB
#include <util/delay.h> // contains the function _delay_ms()

int main(void)
{
   DDRB = 0b11111111;   // set all pins on PortB to output
   while(1)
   {
        PORTB  = 0b00000001;  // Set PortB0 output high, others low
        _delay_ms(250);       // Do nothing for 250 ms
        PORTB  = 0b00000000;  // Set all of them low
        _delay_ms(250);       // Do nothing for 250 ms
   }
   return 0;
}

/*  DDRB is a 8-bit register which sets direction for each pin in PortB
    PORTB decides output for output pins
    0b - prefix for binary numbers, 0x - hex, no prefix - decimal
    Thus, 15 = 0xf = 0b1111
*/
```

Compiling and Programming

- Save the code as blink.c in a separate folder (not strictly necessary, just a good practice)
- Create a makefile using Mfile and save in the same folder
- Open it in Programmer's Notepad and change:
  - `Line 44: MCU = atmega16`
  - `Line 65: F_CPU = 1000000`
  - `Line 73: TARGET = blink, Line 83: SRC = $(TARGET).c`
  - `Alternatively, TARGET = anything_you_want and SRC = blink.c`
  - `Line 278: AVRDUDE_PROGRAMMER = usbasp`
  - `Line 281: AVRDUDE_PORT = usb`
- In Programmer's Notepad, *Tools > Make All* to compile
- Connect USBasp to computer and ATmega16
- *Tools > Program* to program ATmega16

A better code

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
   DDRB = DDRB | 0b00000001;
  // Set PortB0 as output, leave other pin directions unchanged
   while(1)
   {
        PORTB = PORTB | 0b00000001;
        // Set output high without affecting others
        _delay_ms(250);
        PORTB = PORTB & 0b11111110;
        // Set output low without affecting others
        _delay_ms(250);
    }
   return 0;
}
```

Try this out : Toggle the pin instead of set and clear

A more readable code

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
   DDRB = DDRB | _BV(PB0);
/* _BV(x) = 2^x and PB0 is defined to be 0 in avr/io.h
   So, _BV(PB0) = 2^PB0 = 2^0 = 1 = 0b00000001  */
   while(1)
   {
        PORTB = PORTB | _BV(PB0);
        // Set output high without affecting others
        _delay_ms(250);
        PORTB = PORTB & (~(_BV(PB0)));
        // Set output low without affecting others
        _delay_ms(250);
   }
   return 0;
}
```

Input and output

```
/* If input on PortB0 is low, set output on PortB1 high.
   Else, set output on PortB1 low.   */

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
   DDRB |= _BV(PB1);   // x |= y; is same as x = x|y;
   while(1)
   {
         if((PINB & _BV(PB0)) == 0) PORTB |= _BV(PB1);
         else PORTB &= ~(_BV(PB1));
    }
   return 0;
}
```

Try these out:

Blink LED on PortB1 if PortB0 is high, else turn LED off.

What happens when an input pin if left floating?

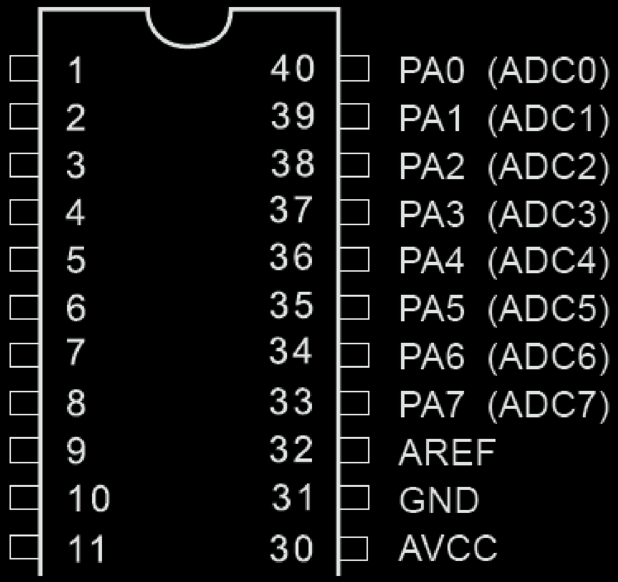
Data types available

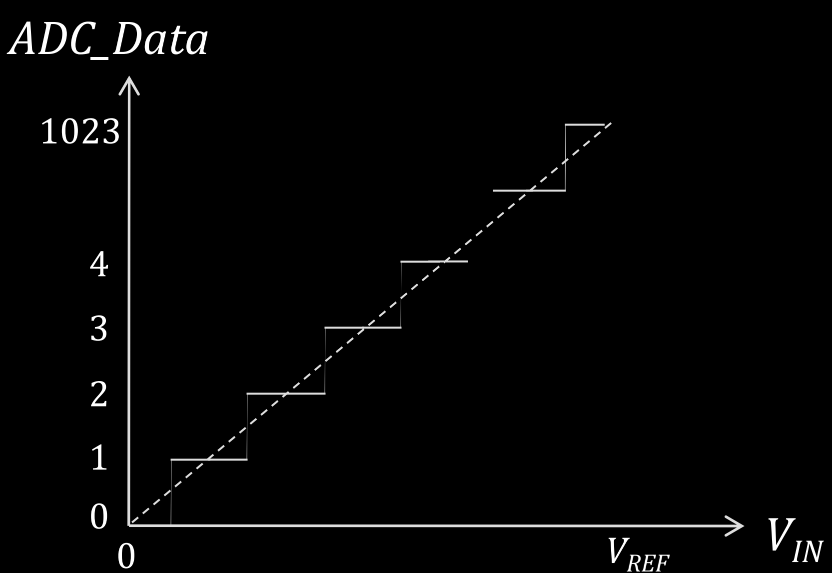
Data type	Size in bits	Range
char	8	-128 – 127
unsigned char	8	0 – 255
int	16	-32768 – 32767
unsigned int	16	0 – 65535
(unsigned) long	32	(0 – 2^{32}-1) -2^{31} – 2^{31}-1
(unsigned) long long	64	(0 – 2^{64}-1) -2^{63}– 2^{63}-1
float, double	32	±1.175*10^{-38} – ±3.402*10^{38}

1. Since AVR is an 8-bit uC, char and unsigned char are natural data types
2. int should be used only when the range of char is not sufficient
3. Replace floating point operations by int or char operations wherever possible. e.g. Instead of `y = x*0.8`, use `y = (x*4)/5`
4. Take care of overflow. Stay within the range of the data-type used
5. Beware of integer round-off

Analog to Digital converter

- Converts an analog voltage V_{IN} to a digital number ADC_Data
- 10-bit conversion result
- Conversion time = 13.5 * ADC clock period
- Up to 15k conversions per sec
- 8 inputs (Port A)





$$ADC\_Data = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

(rounded off to nearest integer)

Initialize ADC and read input

- Setting up the ADC
  - Select reference voltage : REFS1:0
  - Select prescaler : ADPS2:0
  - Select output format (left adjust/right adjust) : ADLAR
  - Enable the ADC : ADEN
- Reading an analog input
  - Select input pin to read : MUX4:0
  - Start conversion : ADSC
  - Wait for conversion to finish : ADSC
  - Read the result registers :   ADCH:ADCL

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCH	ADC Data Register High Byte							
ADCL	ADC Data Register Low Byte							
SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10

adcroutines.c

```
#include <avr/io.h>
void adc_init(void)
{
   DDRA   = 0x00;     // Set all PortA pins to input
   PORTA  = 0x00;     // Disable pull-ups
   ADMUX  = _BV(REFS0) | _BV(ADLAR);
   ADCSRA = _BV(ADEN) |_BV(ADPS2)|_BV(ADPS1);
  /* Use AVcc as reference, Left adjust the result
     Enable the ADC, use prescaler = 64 for ADC clock */
}

unsigned char adc_read (unsigned char channel)
// valid options for channel : 0 to 7. See datasheet
{
   ADMUX   = (ADMUX&0xe0) + channel;
    // Set channel bits in ADMUX without affecting other bits
   ADCSRA |= _BV(ADSC);  // Start conversion
   while((ADCSRA & _BV(ADSC)) != 0) {};
    // Do nothing until conversion is done
   return(ADCH);  // Return upper 8 bits
}
```

Try these out :
Control the blinking speed using potentiometer
Try the other ADC options : Free running, Auto-trigger, different channels, prescalers, etc.

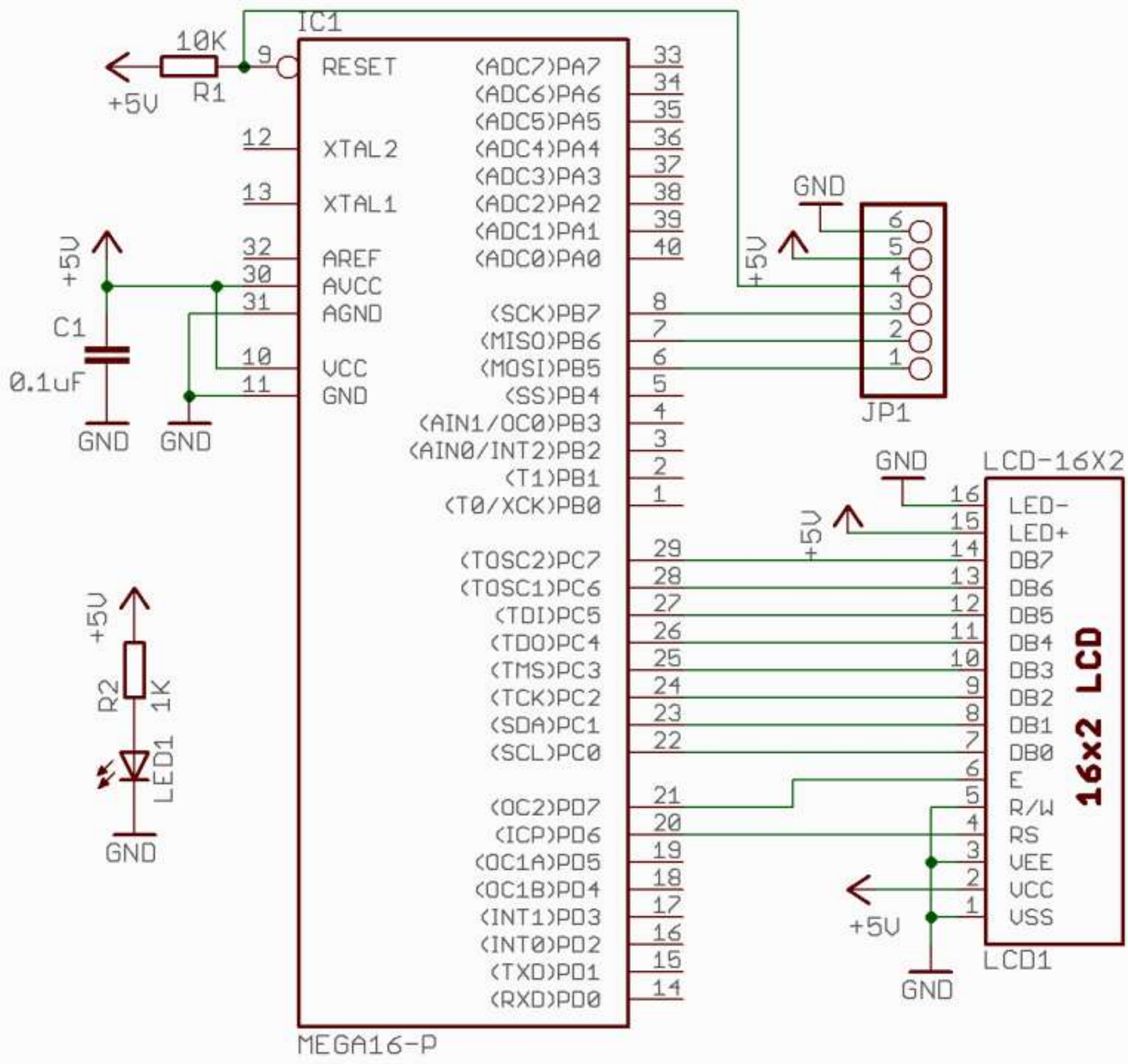
Liquid Crystal Display (LCD)

- Alpha-numeric display with backlight
- 16 columns x 2 rows (larger sizes available too)
- 8-bit data interface, 3 control signals
- For pin-starved applications : 4-bit data interface
- +5V supply, separate power for backlight
- Readymade libraries available
- Working with multiple files

  1. main.c – the main code
  2. lcdroutines.c – has functions for LCD interfacing
  3. lcdroutines.h – defines connections and function prototypes
- In main.c, `#include”lcdroutines.h”`
- In Makefile,

```
TARGET = main
SRC = $(TARGET).c lcdroutines.c
```

- For more information, consult the datasheet of HD44870


IC1
10K
R1
+5V
9
RESET
12
XTAL2
13
XTAL1
32
AREF
30
AVCC
31
AGND
10
VCC
11
GND
C1
0.1uF
GND
GND
(ADC7)PA7 33
(ADC6)PA6 34
(ADC5)PA5 35
(ADC4)PA4 36
(ADC3)PA3 37
(ADC2)PA2 38
(ADC1)PA1 39
(ADC0)PA0 40
(SCK)PB7 8
(MISO)PB6 7
(MOSI)PB5 6
(SS)PB4 5
(AIN1/OC0)PB3 4
(AIN0/INT2)PB2 3
(T1)PB1 2
(T0/XCK)PB0 1
(TOSC2)PC7 29
(TOSC1)PC6 28
(TDI)PC5 27
(TDO)PC4 26
(TMS)PC3 25
(TCK)PC2 24
(SDA)PC1 23
(SCL)PC0 22
(OC2)PD7 21
(ICP)PD6 20
(OC1A)PD5 19
(OC1B)PD4 18
(INT1)PD3 17
(INT0)PD2 16
(TXD)PD1 15
(RXD)PD0 14
MEGA16-P
GND
+5V
6
5
4
3
2
1
JP1
+5V
R2
1K
LED1
GND
GND
LCD-16X2
16 LED-
15 LED+
14 DB7
13 DB6
12 DB5
11 DB4
10 DB3
9 DB2
8 DB1
7 DB0
6 E
5 R/W
4 RS
3 VEE
2 VCC
1 VSS
+5V
+5V
GND
LCD1
16x2 LCD

On the bread-board

lcdroutines.h

```
// Connections between uC and LCD
#define DATA_DDR        DDRC
#define DATA_PORT       PORTC

#define CONTROL_DDR     DDRD
#define CONTROL_PORT    PORTD
#define RS              PD6
#define E               PD7

/* Function prototypes for interfacing to a 16x2 LCD
   Actual functions in lcdroutines.c */
void lcd_init(void);   // Initialize the LCD
void lcd_clear(void);  // Clear LCD and send cursor to first char
void lcd_home(void);   // Send cursor to first character
void lcd_command(unsigned char command);  // Send command to LCD

void display_char(unsigned char data);  // Display ASCII character
void display_byte(unsigned char num);   // Display number 0 - 255
void display_int(unsigned int num);     // Display number 0 - 65535

void move_to(unsigned char x, unsigned char y);  // Move cursor
```

Codes:

1. Hello World - lcd1.c
2. A better Hello World - lcd2.c
3. Animated display - lcd3.c
4. Digital Watch - lcd4.c
5. Voltmeter - lcd5.c

lcd1.c

```
#include <avr/io.h>
#include "lcdroutines.h"

int main(void)
{
  unsigned char a[] = {"Hello World!"};

  lcd_init();
  for(unsigned char i =0;i<sizeof(a)-1;i++) display_char(a[i]);

  while(1);
  return 0;
}
```

Interrupts

- Suppose you are making an obstacle avoiding robot.
- Drive motors continuously, and stop immediately when obstacle is detected.

```
while(1)
{
  drive_motors();
  if(obstacle_detected()) stop_motors();
}
```

- No obstacle detection when driving motors. Might miss the obstacle and crash!

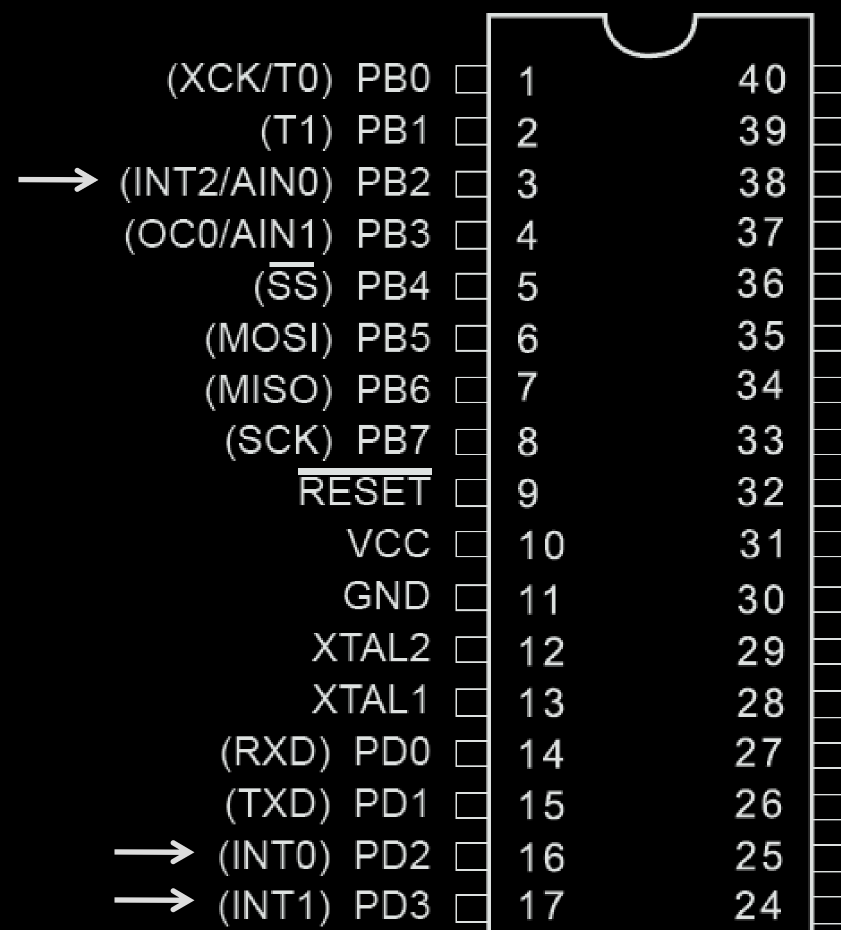
```
while(1)
{
  drive_motors();
  while(!obstacle_detected()) {};
  stop_motors();
}
```

- No motor driving when waiting for obstacle!

The solution - interrupts

- Run motor-driving routine in the main loop. Interrupt it when obstacle is detected.

```
ISR(vector_name)
{
  stop_motors();
}

int main(void)
{
  initialize_interrupt();
  while(1)
  {
    drive_motors();
  }
}
```



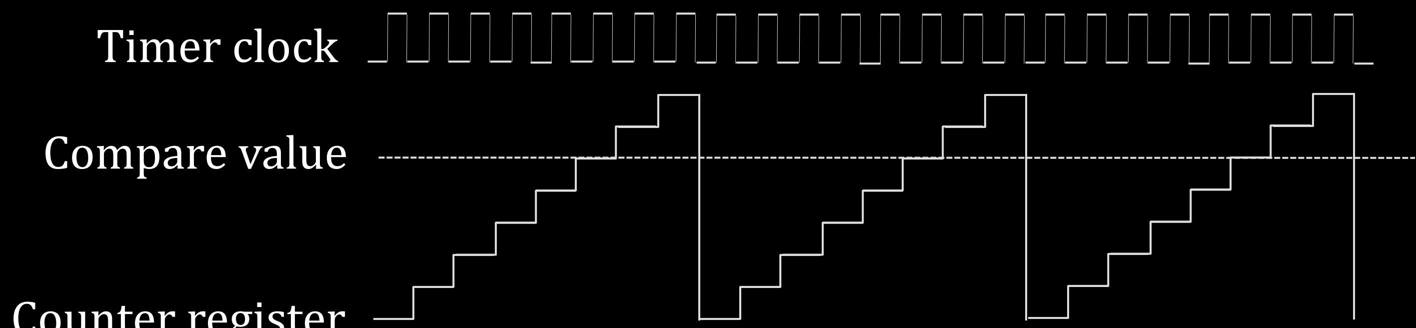
Interrupts explained

- Interrupt is a special "function" which gets called when a specific hardware condition is met.
- When condition is met, an interrupt flag is set in a specific register.
- If interrupt is enabled, the interrupt flag signals the muC to stop executing main program and jump to an interrupt service routine (ISR).
- After ISR is done, CPU resumes main program from where it left.
- Possible conditions that can cause interrupts -
  - Voltage change on pins INT0, INT1 and INT2. (External interrupt)
  - ADC conversion complete, Timer, UART, etc. (Internal interrupt)
- In ISR(vector name), vector name depends on which interrupt condition is being used.
- Get the vector name from avr-libc manual : WinAVR_installation_path/doc/avr-libc/avr-libc-user-manual/group__avr__interrupts.html

```
#include <avr/io.h>
#include <avr/interrupt.h>  // Need to include this for interrupts
#include <util/delay.h>
volatile unsigned char i=0; /* Declare variables being used in ISR
                               as global and voltatile */
ISR(INT0_vect)
{
  PORTC |= _BV(PC6);
  PORTC &= ~(_BV(PC0));
  i++;
}
int main(void)
{
  DDRD = ~(_BV(PD2));      // Set PortD2 (INT0) as input
  PORTD = _BV(PD2);        // Enable pullup on INT0
  DDRC = _BV(PC7) | _BV(PC6); // LED outputs
  MCUCR &= ~(_BV(ISC00) | _BV(ISC01)); // Low level on INT0 generates interrupt
  GICR  |= _BV(INT0);      // Enable INT0 interrupt
  sei();                   // Enable global interrupts. Else no interrupt works.
  while (1)
  {
    _delay_ms(100);
    PORTC ^= _BV(PC7);
    PRTC &= ~(_BV(PC1));
  }
  return 0;
```

Blink LED with a timer

- In blink.c, CPU does nothing useful for 250ms
- Instead, let a timer run in the background
- Interrupt the CPU every 250ms and toggle LED
- How a timer/counter works
  - Normally, counter register increments every timer clock pulse (resets to zero when it reaches maximum value)
  - Timer clock frequency = Main clock / prescaler
  - When counter value equals compare value, a compare match interrupt flag is set



Timer/counters in ATmega16

- ATmega16 has 3 timer/counters : 0, 1 and 2
- 0 and 2 are 8-bit counters, 1 is a 16-bit counter
- Each T/C has different modes of operation : normal, CTC and PWM
- Special waveform generation options : variable frequency pulses using CTC, variable duty-cycle pulses using PWM

Initializing a timer and interrupts

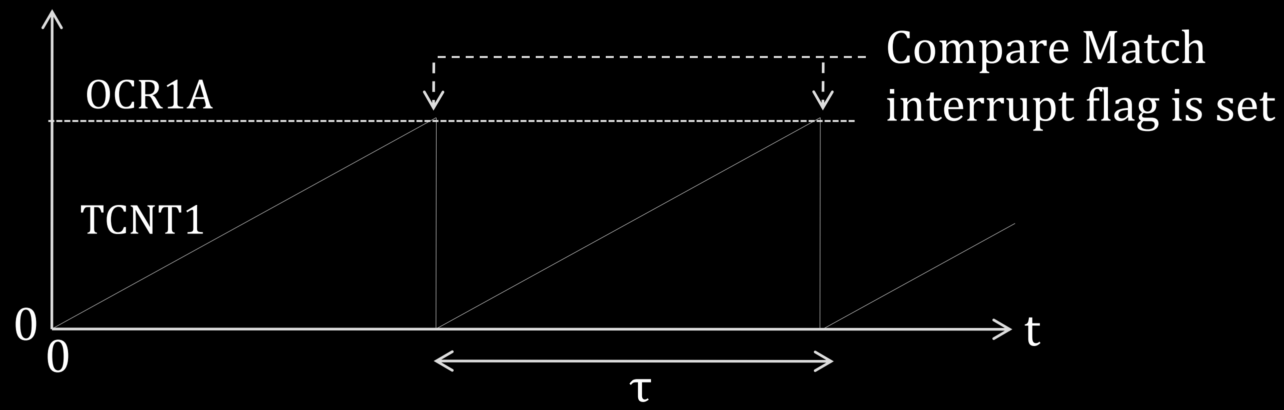
- Select mode of operation : Normal
- Select prescaler
- Select the event which causes the interrupt
- Set the time delay to interrupt every 250ms
- Relevant registers for Timer 1:
  - TCNT1 : 16-bit count register
  - TCCR1A : Mode of operation and other settings
  - TCCR1B : Mode of operation, prescaler and other settings
  - OCR1A : 16-bit compare A register
  - OCR1B : 16-bit compare B register
  - TIMSK : Interrupt mask register

timer1blink.c

```
#include <avr/io.h>
#include <avr/interrupt.h>

void timer1_init(void)      // Initialize timer
{
   TCCR1B = _BV(CS11);      // Normal mode, prescaler = 8
   TIMSK = _BV(OCIE1A);     // Enable T/C1 Compare Match A interrupt
}

ISR(TIMER1_COMPA_vect)      // ISR for T/C1 Compare Match A interrupt
{
   PORTB ^= _BV(PB0);        // Toggle pin
   OCR1A += 31250;          /* Increment Compare Match register by
                               250ms*1MHz/8 = 31250 */
   return;                   // return to main code
}

int main(void)
{
   DDRB = _BV(PB0);         // Set pin to output
   timer1_init();           // Initialize timer
   sei();                   // Enable global interrupts
   while(1) {};             // Do anything you want here
   return 0;
}
```

CTC mode

- Timer 1 counter register (TCNT1) increments every timer clock
- When TCNT1 reaches OCR1A, compare match interrupt flag is set
- TCNT1 is reset to zero

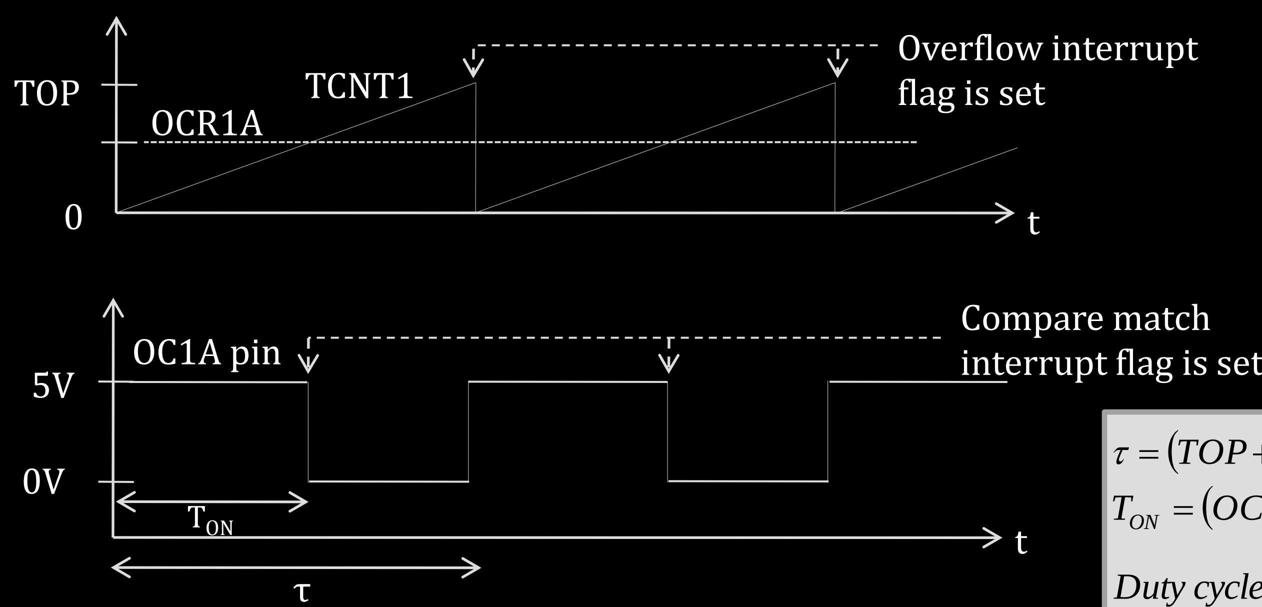


$$\tau = (OCR1A + 1) \cdot T_{TIMER1} = (OCR1A + 1) \cdot T_{CLK} \cdot prescaler$$

Try these out:
Use CTC instead of Normal mode to blink LED
Now control the blinking speed using potentiometer

Controlling the brightness of an LED using Pulse Width Modulation (PWM)

- Apply a variable duty-cycle high frequency clock to the LED
- Duty-cycle decides the brightness of the LED
- Timer 1 in PWM mode
  - Special output pins – OC1A (PD5) and OC1B (PD4)
  - Function of output pins depends on the compare output mode : COM1A and COM1B bits in TCCR1A
  - No interrupt is needed



$$\tau = (TOP+1) \cdot T_{CLK} \cdot prescaler$$

$$T_{ON} = (OCR1A+1) \cdot T_{CLK} \cdot prescaler$$

$$Duty\ cycle = \frac{OCR1A+1}{TOP+1}$$

Fast PWM mode with non-inverted compare match output

- Initialize the timer
  - Set timer to Fast PWM 8-bit mode
  - Set compare output mode
  - Set prescaler
  - Set OC1A (PD5) pin to output
- Initialize the ADC
- Use ADC result to change OCR1A

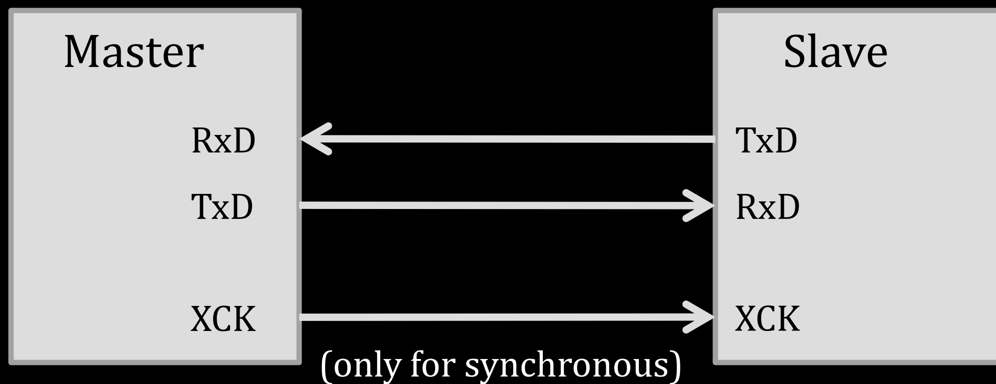
ledbrightness.c

```
#include <avr/io.h>

void timer1_init(void)
{
   TCCR1A = _BV(WGM10) | _BV(COM1A1); // Fast 8-bit non-inverting
   TCCR1B = _BV(WGM12) | _BV(CS11);  //  PWM with prescaler = 8
   OCR1AH = 0;
}

int main(void)
{
   DDRD |= _BV(PD5);    // Necessary to set DDR value for PD5
   timer1_init();
   adc_init();          // Get ADC functions from the ADC tutorial
   while(1) OCR1AL = adc_read(0);  // Set duty cycle
   return 0;
}
```

Try this out:
Use CTC mode instead of PWM mode and change blinking speed instead of brightness without using interrupts

USART

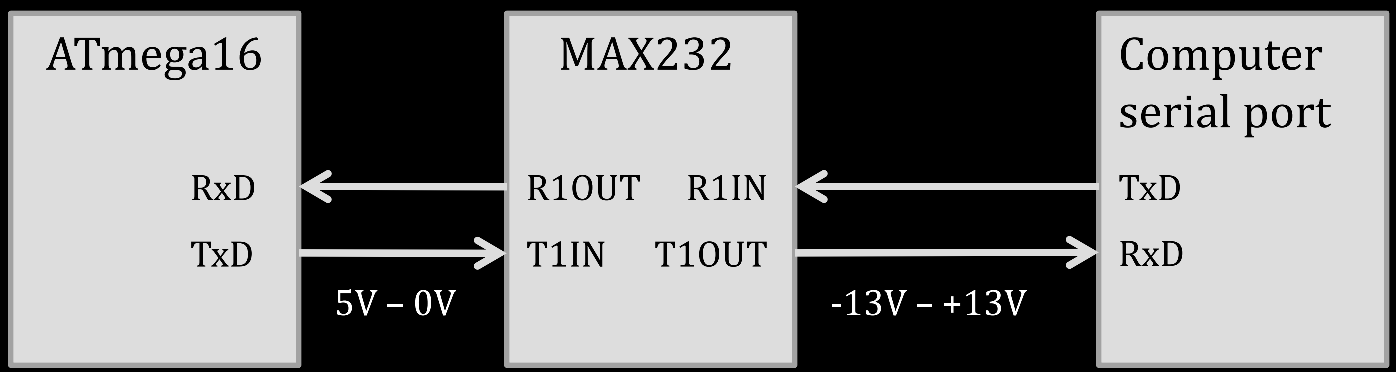
- Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Serial communication : one bit at a time.
- Communication speed : baud rate (number of bits per sec)



- Master and slave can be microcontrollers, computers or any electronic device with a USART interface
- Computer serial port uses UART protocol. Connection between uC and computer can be used for data-logging, sending commands, etc.
- Programs like HyperTerminal, Tera Term, Bray’s terminal are available for using serial port in Windows

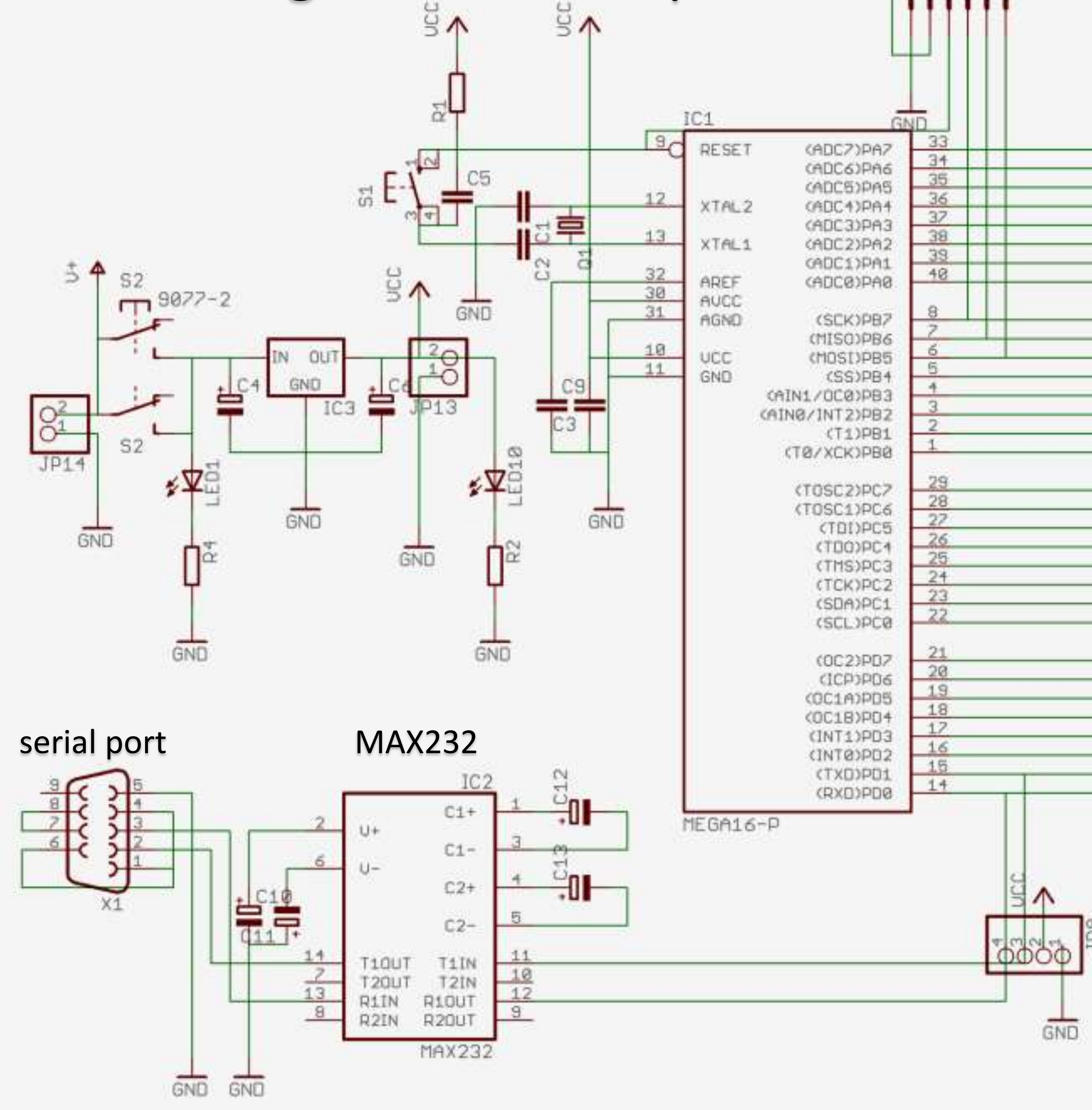
uC to computer

- Serial port voltage levels are different: +13V : 0 and -13V : 1
- Voltage converter IC :MAX232



- USB-serial converter can be used if no serial port is available
- Some USB-serial converters do not require MAX232 to be used. Their output is already 5V compliant and can be connected to the uC pins directly.
- Download Tera Term from http://ttssh2.sourceforge.jp/

Circuit diagram for serial port



Using the UART in AVR

- Relevant registers
  - Control and status registers – UCSRA, UCSRB, UCSRC
  - Baud rate registers – UBRRH, UBRRL
  - Data register – UDR (2 registers by the same name)
- Initialize UART
  - Enable transmitter and receiver
  - Set baud rate – typically 2400 bps
  - Frame length – 8bits
  - Other settings – 1 stop bit, no parity bits
- Transmit data
  - Check if data register is empty (UDRE bit in UCSRA)
  - Write data to data register UDR
- Receive data
  - Check if receive is complete (RXC bit in UCSRA)
  - Read UDR

UART functions

```
#include <avr/io.h>
void uart_init( unsigned int ubrrval )
// ubrrval depends on uC clock frequency and required baudrate
{
  UBRRH = (unsigned char) (ubrrval>>8);  // set baud rate
  UBRRL = (unsigned char) ubrrval;       // set baud rate
  UCSRB = _BV(RXEN) | _BV(TXEN);
  /* Enable UART receiver and transmitter, 8-bit data length,
     1 stop bit, no parity bits */
}

unsigned char uart_rx( void )
{
  while ( !(UCSRA &  _BV(RXC)) );    // wait until receive complete
  return UDR;
}

void uart_tx( unsigned char data )
{
  while ( !(UCSRA & _BV(UDRE)) );    // wait until UDR is empty
  UDR =  data;
}
```

If things aren't working as expected

Nothing works: Check the power

uC doesn't get programmed:

1. Check connections from USBasp to uC. You might have connected the 6-pin programming connector in the reverse order.
2. Check if correct jumpers are set in USBasp.
3. Did you write fusebits incorrectly before?
4. Check the Makefile.

LED doesn't blink:

1. Is the LED connected in the reverse? Is the series resistor correct?
2. Is it blinking too fast?
3. Is the uC pin damaged/shorted? Is the DDR bit for the pin set?

Some pins on PortC aren't working properly: Disable JTAG using fusebits.

Hyperterminal shows nothing/garbage:

1. Check the serial port and MAX232 using a loop-back test.
2. Do the settings on Hyperterminal and uC match?

ADC doesn't give correct result:

1. Is the reference set correctly?
2. Is your output format correct (left adjust/right adjust) ?
3. Try reducing the ADC clock speed by increasing prescaler.
4. Check if there is too much noise on the input and reference.
5. Keep pull-ups disabled (`PORTA = 0`)

Mathematical operations give incorrect results:

1. Check for overflow, integer round-off.
2. Use the correct data type.

Interrupts aren't working:

1. Are global interrupts enabled using `sei()`?
2. Is the interrupt mask set correctly?
3. Is the ISR vector name correct?

`for(int i= 0;i<50000;i++)` is an infinite loop since the maximum value of an int is 32767.

Read the Atmel application note on “Efficient C Coding for AVR”.

Important precautions:

1. Be extra careful when using USB power for powering the uC. Shorting the supply may damage the USB port. Use external 5V regulated power to be safe.
2. Connect power, programmer and other connectors in the correct location and correct polarity. Use matching male-female connectors as far as possible as they do not allow reverse connection.
3. Do not short output pins to any supply or other outputs.
4. AVR’s have some amount of ESD (electrostatic discharge) protection. But, still, do not touch them if you are charged, say by wearing a woolen sweater.

Resources on the web

- WinAVR : http://winavr.sourceforge.net
- USBasp: http://www.fischl.de/usbasp
- IITB Electronics Club: http://groups.google.com/group/elec-club
- Tutorials and sample codes:
  1. http://www.avrtutor.com
  2. http://winavr.scienceprog.com
  3. http://kartikmohta.com/tech/avr/tutorial
- Atmel application notes
  http://www.atmel.com/dyn/products/app_notes.asp?family_id=607
- Forums: http://www.avrfreaks.net
- Advanced projects:
  1. http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects
  2. http://www.obdev.at/products/vusb/projects.html