Hardware and software resources on the AVR family for the microcontroller project

1. Code Vision

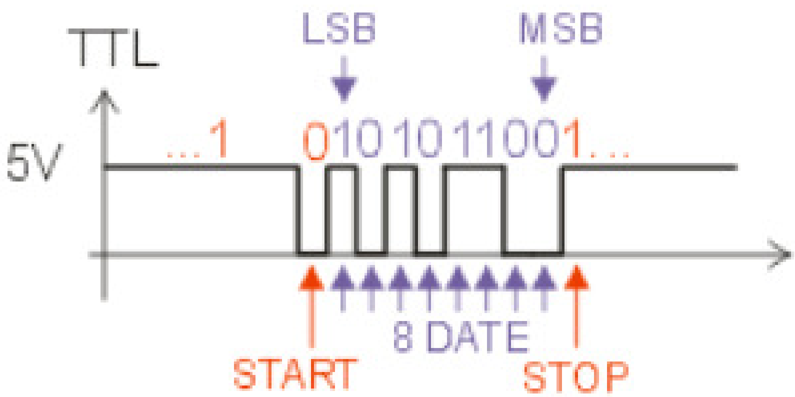
- The C Compiler you use: CodeVisionAVR (CVAVR)
- Where can you find it? a (limited) version is available free of charge at:

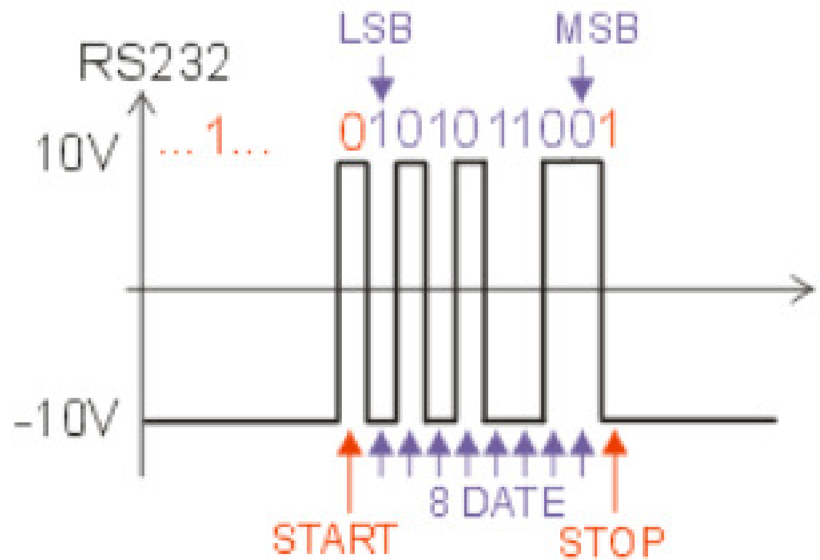
  http://www.hpinfotech.ro/html/download.htm

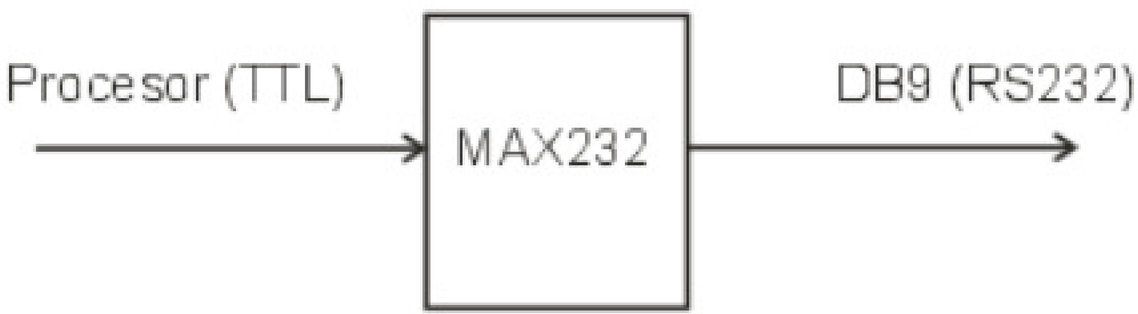
- See appnote (in English) at http://http://ham.elcom.pub.ro/proiect2/files/AtmelCVAVR.pdf

2. Serial ports

- the uC contains the “intelligence”
- it uses pins RxD, TxD
- MAX232, MAX202 etc: electrical level conversion (no “intelligence”)
- Logic Levels: “0” and “1” logic;
- Electrical Levels:
  - the uC uses TTL: “0” = 0V, “1” = 5V
  - the serial line uses RS232: “0” = +12V, “1”=“-12V”
- lines are kept at “1” while idle
- the “intelligence” means adding the start and stop bits and removing them upon reception

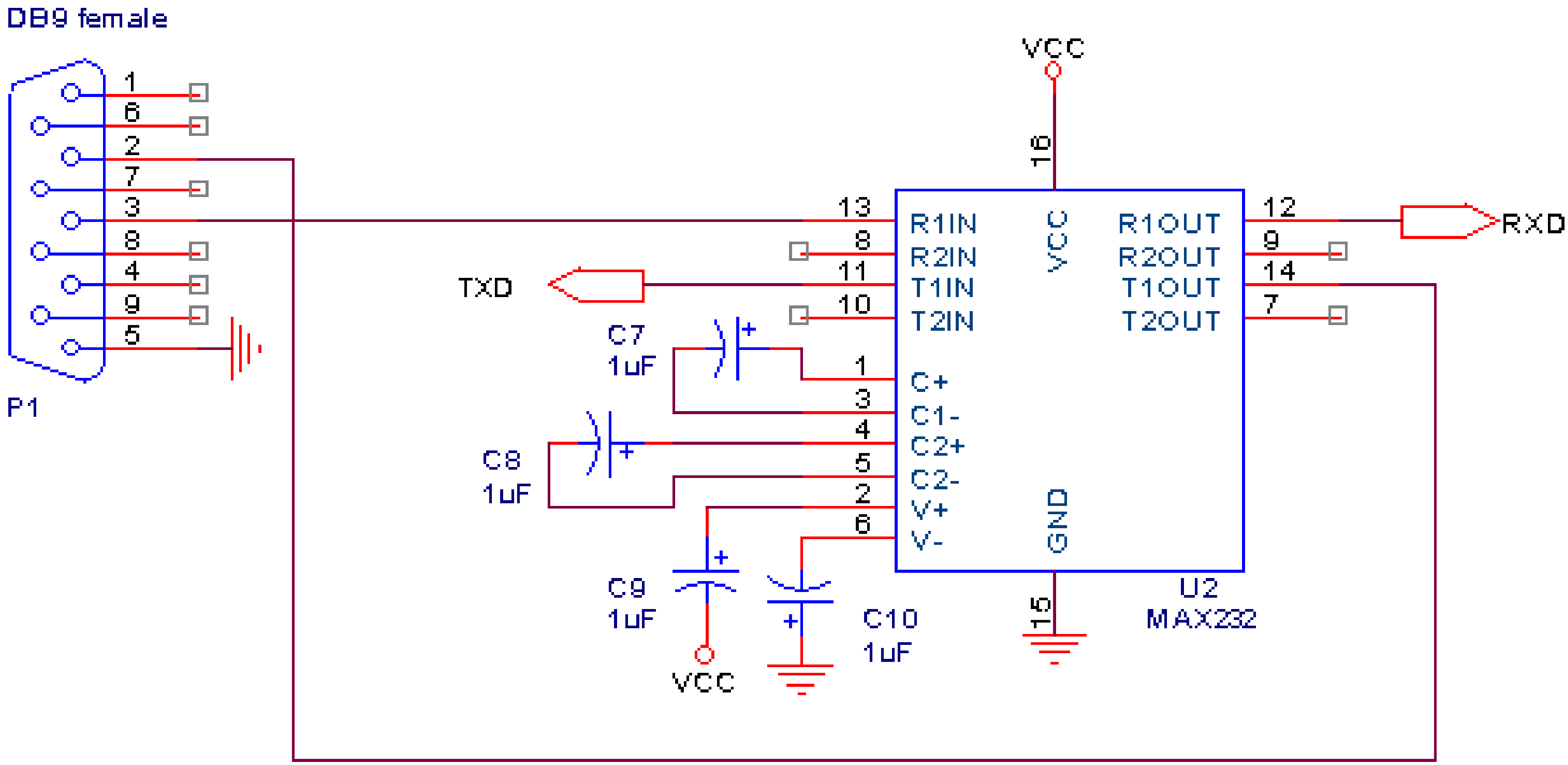






- one character structure: 1 start bit, 8 data bits (8D), 1 stop bit
- the 10 bits on the picture: START, 8D, STOP = 0101010101
- NOTE: LSB is transmitted first (so MSB is adjacent to the stop bit)
  so the 8 bit number must be read from right to left: 01010101

RS232 (+/-10V) - TTL (0-5V) conversion

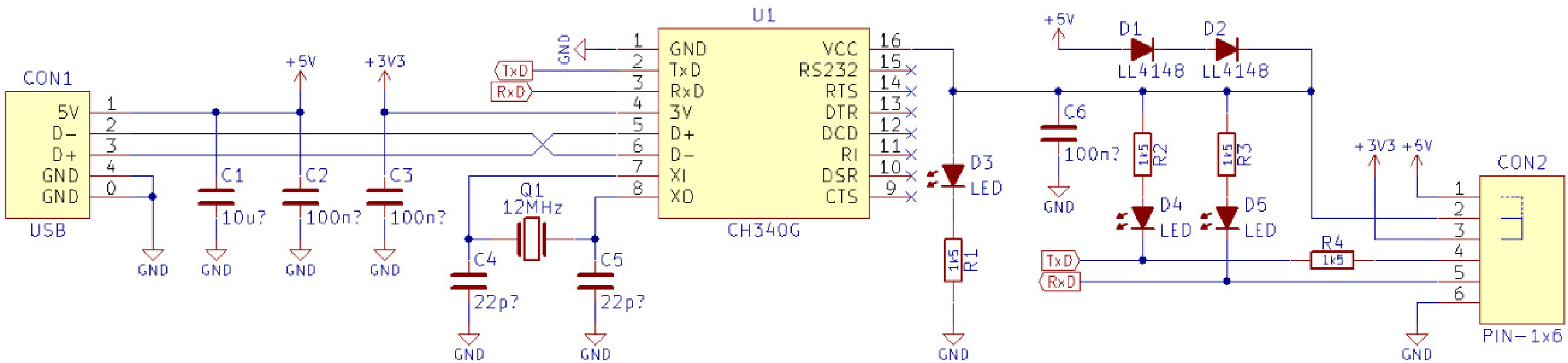


USB- TTL (0-5V) conversion

uP	USB
RXD	TXD
TXD	RXD
GND	GND

- The USB protocol is MUCH more complex than TTL/RS232 !
- The USB-TTL conversion is NOT just a logic level conversion !
- Waveforms on the USB lines use a different speed, include USB host/device arbitration, multiple devices on the same bus etc
- The USB-TTL chip (CH340) is a complex integrated circuit
- You should ONLY view the TTL waveforms on the RXD, TXD pins.

USB- TTL adapter



- The schematic of the USB-TTL adaptar is shown for reference
- D1,D2 are used to lower the 5V voltage to 3.3V (5-0.7-0.7V)
- The jumper on pins 1-2 shorts these diodes so the full 5V is supplied on pin 2
- The 3.3V – 5V jumper should be placed on 5V UNLESS you explicitly modify your own board for 3.3V operation
- Plug the USB-TTL adapter into the PC – a new COM Port (called a „virtual” COM port) will appear (usually COM3 or higher)

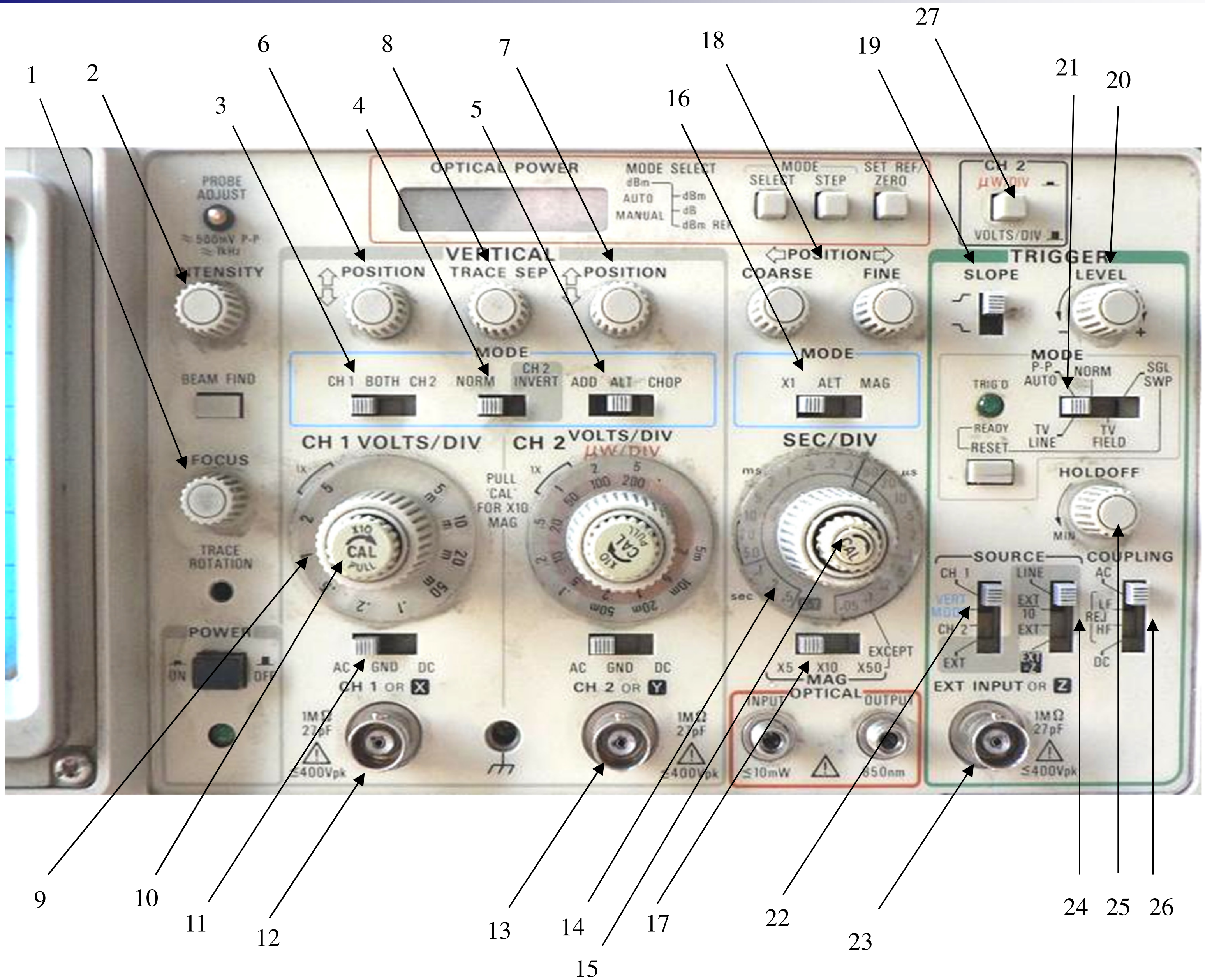
Waveform viewing on the scope

- worth 10% of your grade !

- 10 bits · 1/9600 sec/bit ≈ 1ms
- 10 divisions on the x axis → 1 bit = 1 div (horizontal)
- 1 bit/div → C_X = 0.1 ms/div

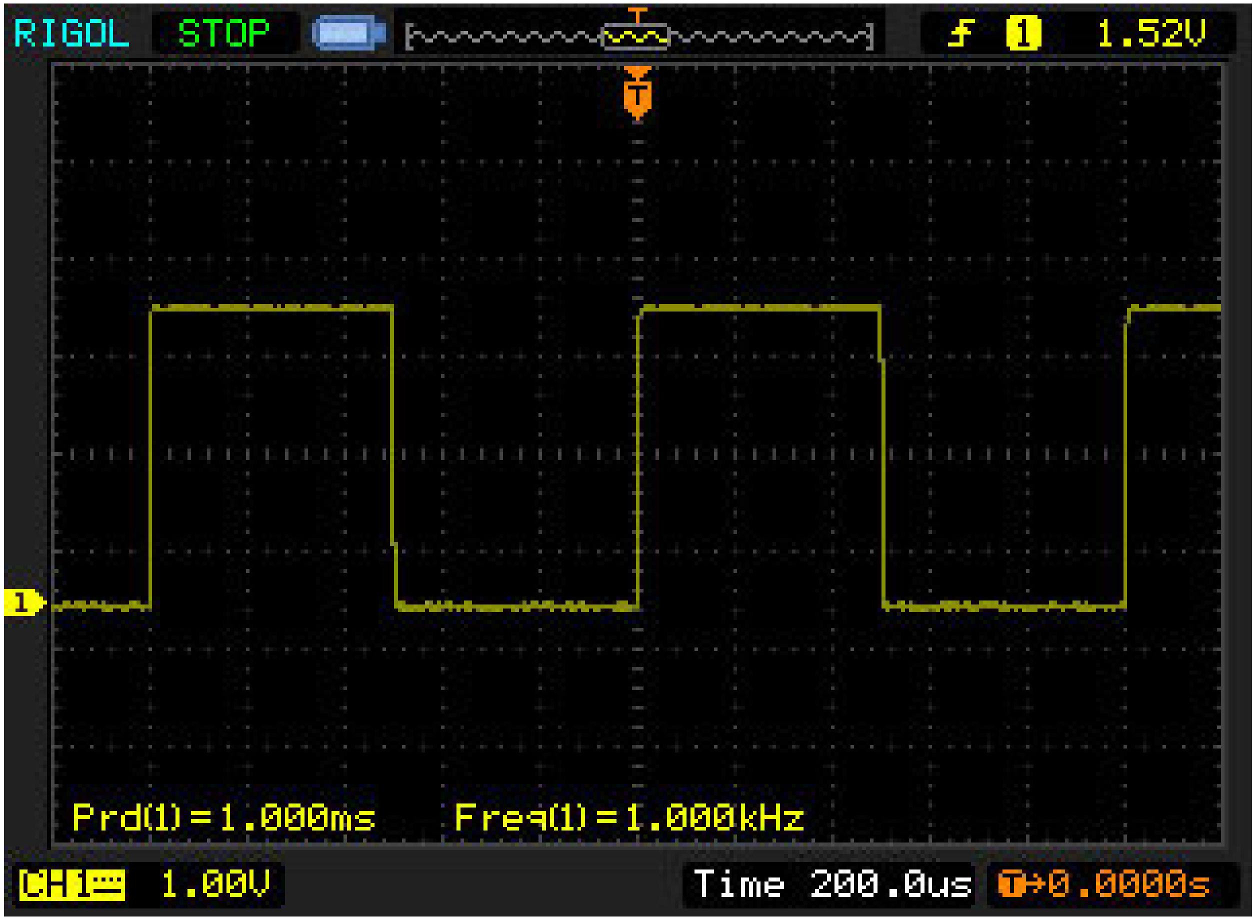
- C_y = 5V/div = 1 div (vertical) for TTL
- Trigger slope = falling slope  for TTL
- Why ? *see the TTL waveform on the previous page*

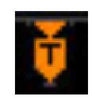
- Look for the C_y, C_x, trigger settings on the scope !



C_y [V/div] = setting 10; C_x[sec/div] = setting 14; Trigger Slope=setting 19

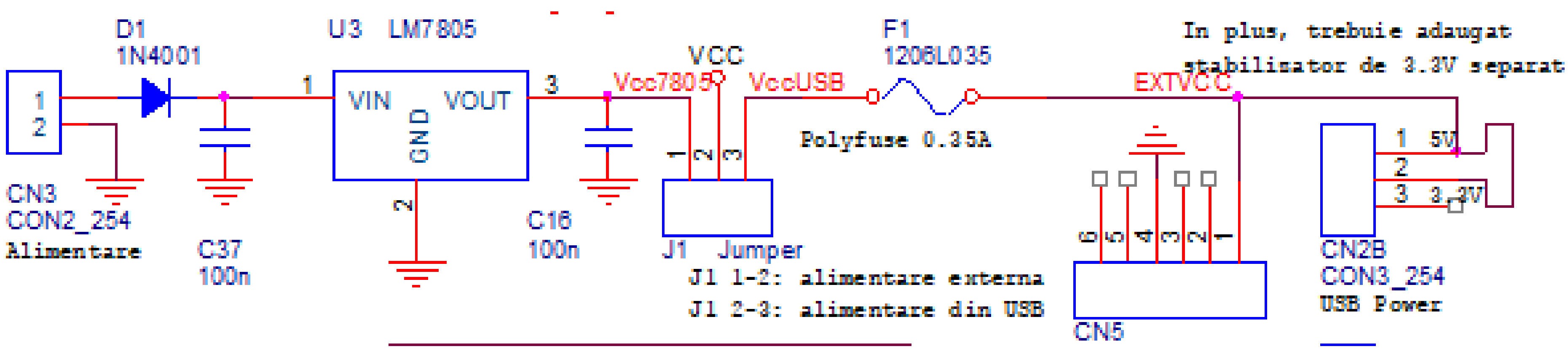
T = Trigger moment, by default on the CENTER of the screen



- Using the *Horizontal Position* knob you should move the trigger moment  to the *left* of the screen in order to see the bits as described
- Note some scopes have more than 10 horiz. divisions (above: 12 divisions)

Powering the board via USB

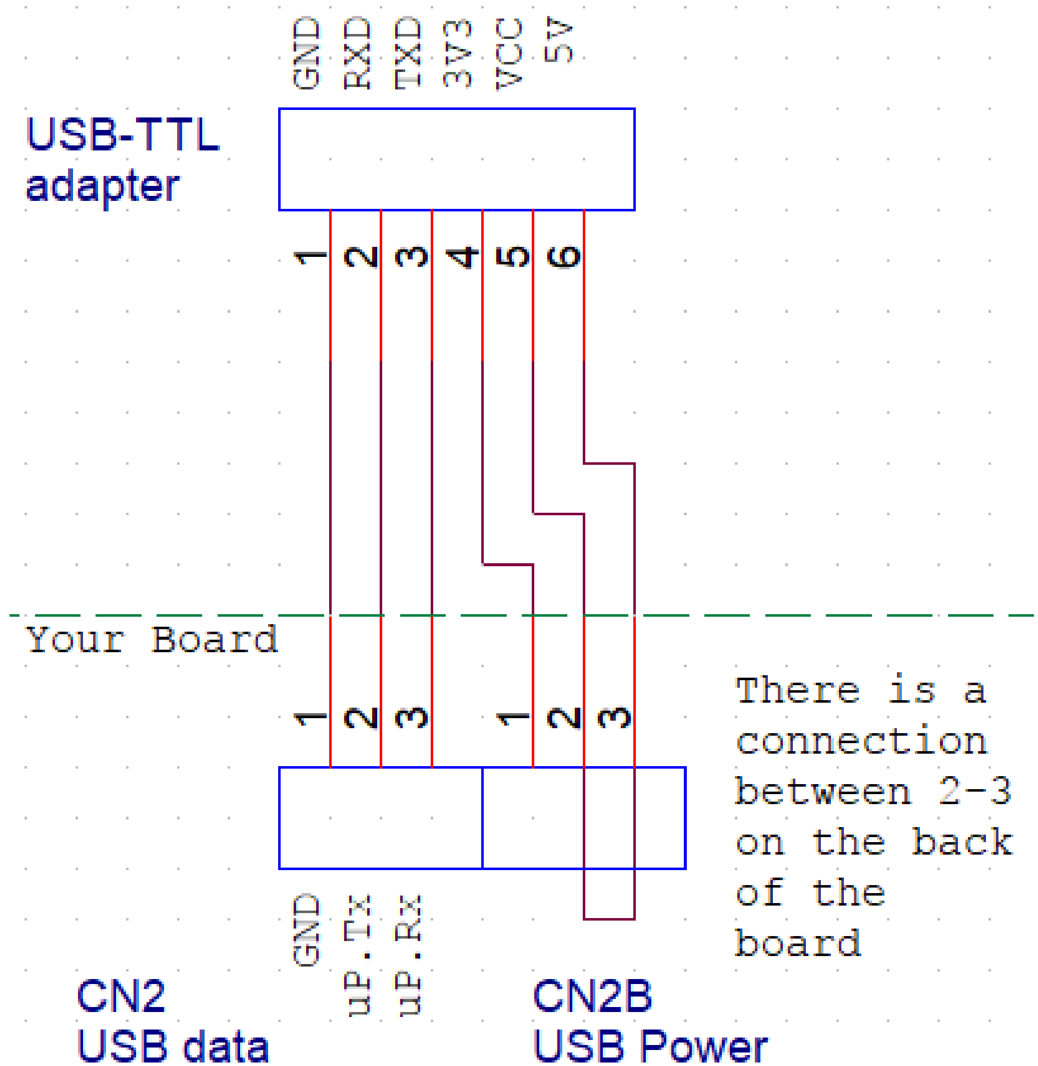
- You can power via USB and eliminate the need for an external power source on CN3; for this, you must properly select J1



- The schematic of the power section of your board is above; Vcc is the power pin of the uP and all other components
  - J1 = 1-2 : external power
  - J1 = 2-3 : USB power
- F1 must pe soldered (either a 1206 Polyfuse or a simple wire can be used)
- Either CN2B going to the USB-TTL adapter must be used, OR the „USB Type B" connector CN5.

Powering the board via USB using the USB-TTL adapter

- You must use 6 wires for CN2 and CN2B, not just 3 wires for data !
- You connect 6 wires to all 6 pins of the USB-TTL adapter
- Now you don't have space to put the yellow jumper on the USB-TTL adapter !
- To solve this, there is already a connection between 2 and 3 of CN2B on your board



- If you decide to power your board with 3V, you must **cut the connection** between 2-3 and solder a wire between 1-2 of CN2B

Serial port communications

- 2 useful applications of the serial port
  - bootloader, for loading the application (see later)
  - debugging using the serial port: on the PC, use a *Terminal* program (e.g. Windows HyperTerminal, or the terminal in CodeVision) which becomes a terminal for your PCB (extends the PC's keyboard and screen as if it were your PCB's keyboard and screen)

Programing an application

- input files: *.c, *.h, etc
- output file: *.hex (the format is called Intel HEX but is not used only on Intel)

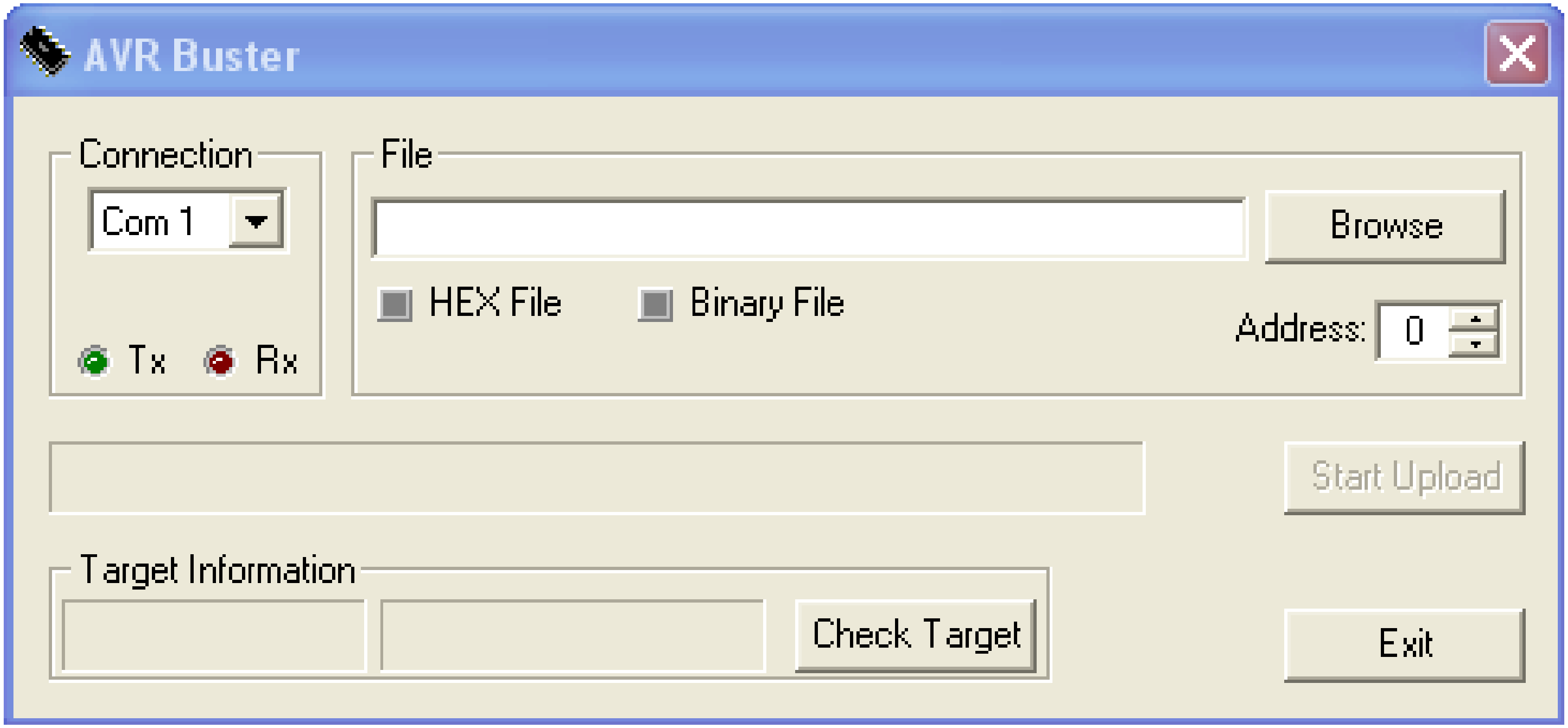
1. classical programming: the uC is taken off the board and plugged in a dedicated programmer
2. *in-system* programming: the uC remains in its socket and the ISP connector is used to connect to an external programmer
3. *bootloader* programming: the boot loader is a special program, similar to an operating system, preloaded in the uC (using method 1 or 2 but only once), which accepts the application program via a serial port (or USB, ethernet, etc)

Bootloader disadvantages: written for a specific processor and clock; needs a button on PORTD.5; doesn’t run on Linux.

3. Program loading into the uC using bootloader/PCLoader

- Boot Loader = a program already loaded into the uC, before I give you the chip
- PC Loader = runs on the PC (Windows)
- they use the serial port for communication
- Boot Loader = equivalent of a micro-OS
  - only function: application loading
  - Boot Loader is loaded at the top of the memory and cannot be overwritten by the application
- the uC's Flash memory contains the **Boot Loader** and the **application program**
  - no multitasking → the 2 do not run simultaneously
- the Boot Loader runs at power-up (or Reset) if the button is pressed; else the application runs.

PC Loader = AVR Buster

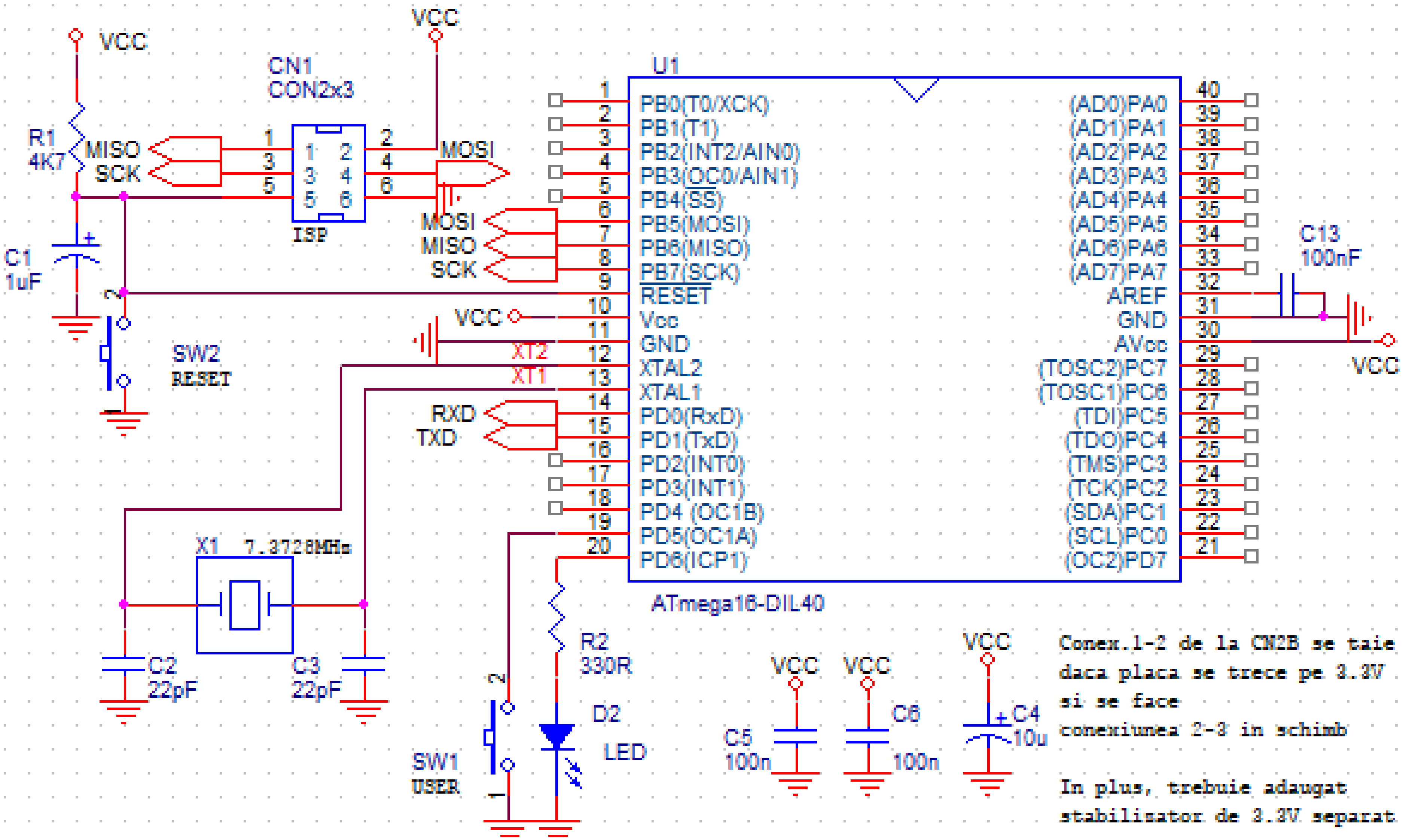


- Select the COM port
- Only COM1 to COM4 can be used
- If using the USB-TTL, the virtual COM can be sometimes greater than 4
- To solve this, use Windows Device Manager, serial port, Advanced properties and select a lower COM (even if it says „in use")

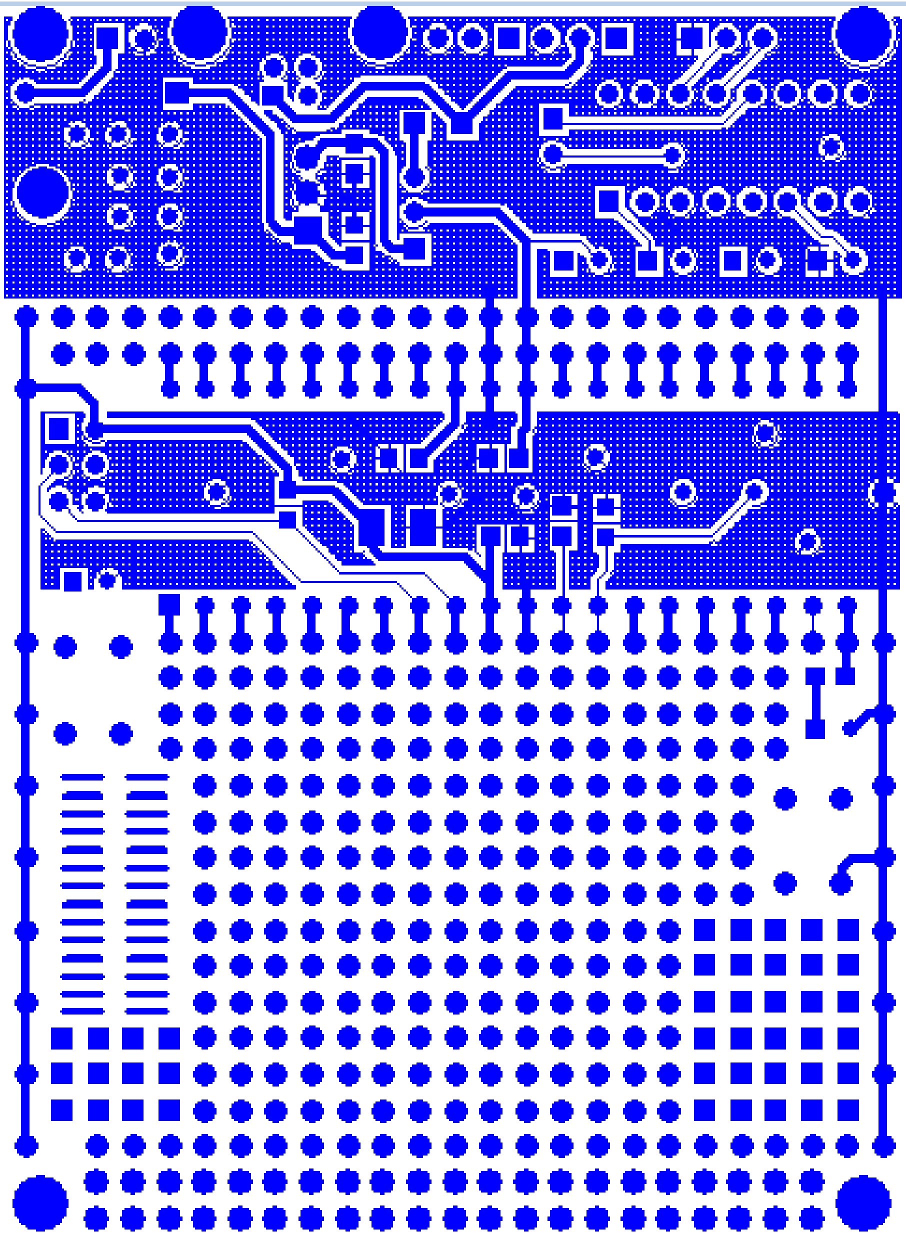
Use of the AVR Buster

1. Select COM1 or COM2 for RS232 ports (physical ports on the back of the PC)
2. Select COM3 or higher for USB virtual ports
3. Using *Browse* load the .HEX file to be uploaded (do NOT load a .c or .h or .prj file – only HEX are executables!)
4. *Start Upload*
5. power up the board while holding the button (OR reset the board while holding the button); the LED will not blink, since this is an application function

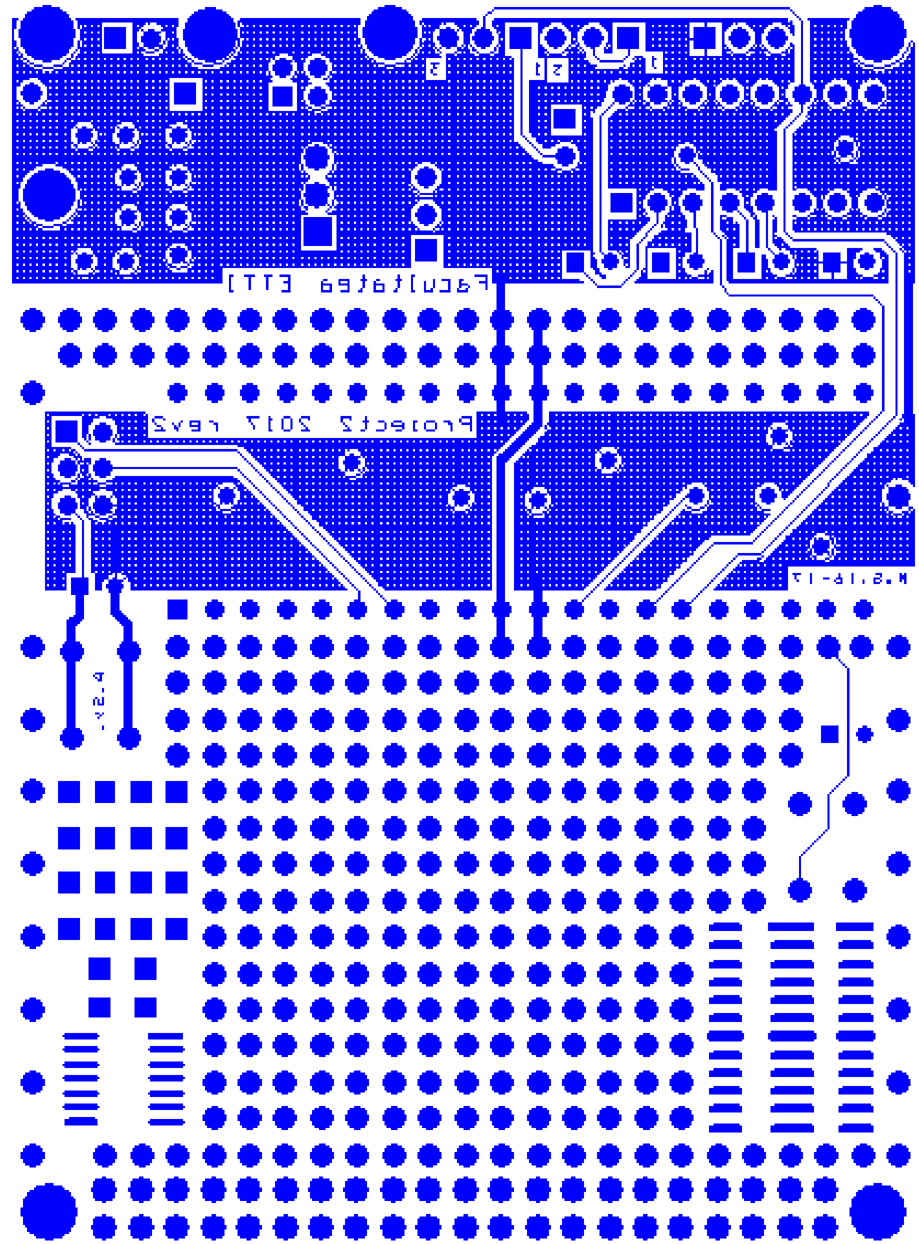
6. Error message “Error Accessing COM Port” = another program (typically, Code Vision) is using the port; use *Disconnect* in the Code Vision *Terminal*



- the power section was shown on a previous slide and it provides Vcc
- the ISP connector is for an optional external programmer (not needed since we have the boot loader)



Top layer



Bottom Layer

- Input pins:
  - Initialize with DDRX.Y = 0
  - Set PORTX.Y = 1 to enable the internal pull-up resistor
  - By default, set PORTX.Y = 0 (no pull-ul resistor)

  - Read value using PINX.Y

  Example:

```
If(PIND.5 == 0)    // read switch connected on D.5
  LED = 1
```

- Output pins:
  - Initialize with DDRX.Y = 1

  - Write value using PORTX.Y

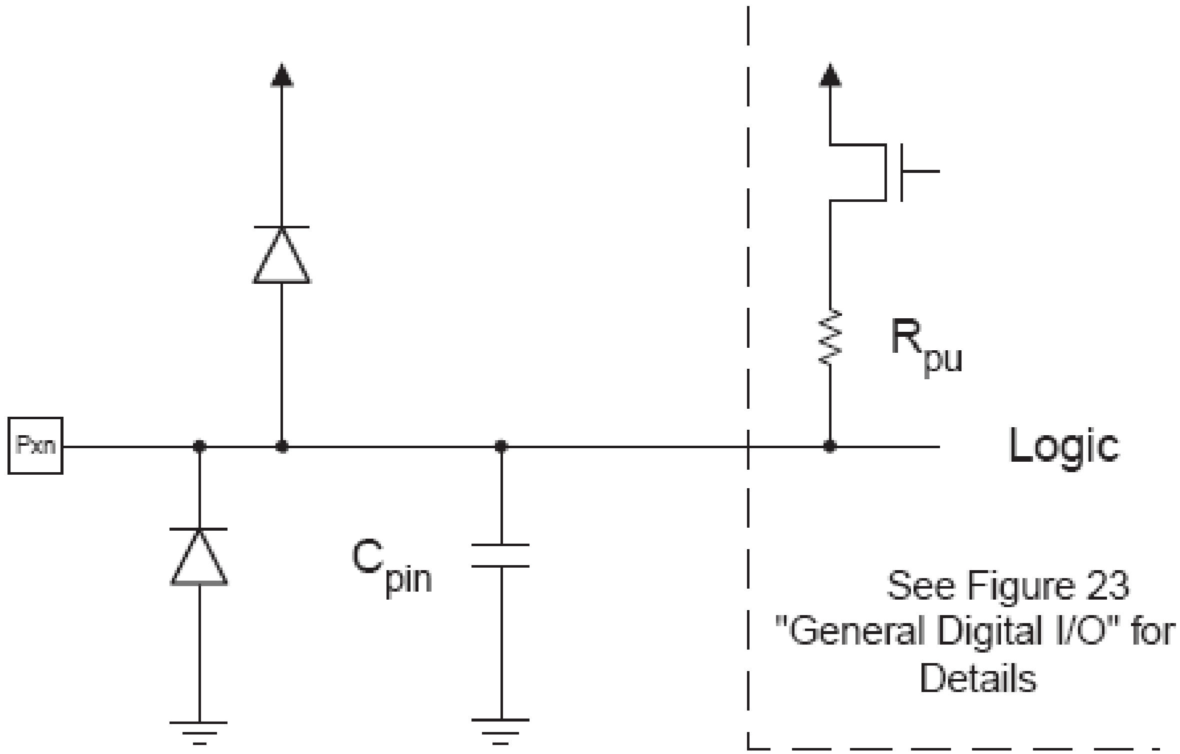
  Example:

```
PORTD.6 = 1    // light up LED connected on D.6
```

- Note: you can access all 8 pins of a port at a time:

```
PORTD = 0b11101011
```

I/O Port schematic for input



- A port pin used as an input
- You can enable the pull-up resistor (R_{pu}) in software
- Example: PORT D.5:
  - DDRD = 0b00000000 // Direction register; 0 = input
  - PORTD = 0b00100000 // 1 on an input pin = pull-up enabled

Interrupts

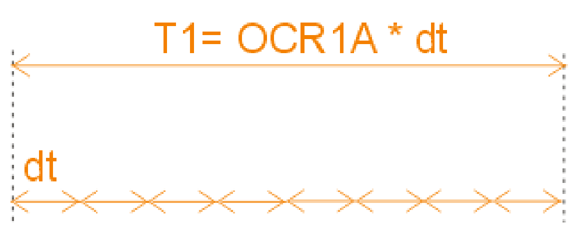
- External interrups: the are called when:
  - a certain pin becomes 0 or 1
  - a character is received on the serial port, etc

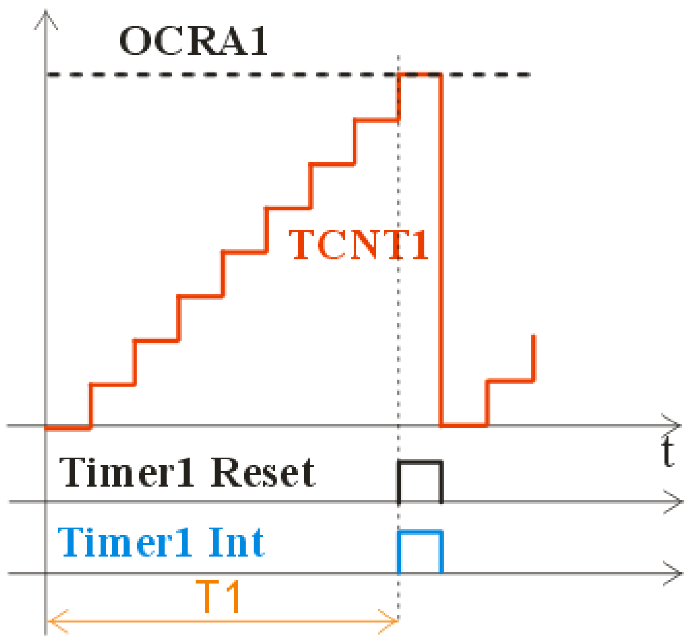
- Internal interrupts:
  - a timer register reaches a certain value (a certain time is reached)
  - an A/D conversion is ready, etc

- See *datasheet* for a complete list for the AT Mega 16
- in the software, an interrupt is *serviced* by a C function called ISR (*Interrupt Service Routine*)

- See the test program for an example using the timer interrupt

- Timer 0,1,2
- 8 bits or 16 bits
- source: internal or external clock, with or without prescaler
- Many operating modes, see the *datasheet* for full details
- example: Timer1 in CTC mode (*Clear Timer on Compare Match*)
  - the selected clock source increments the timer
  - the current value is held in TCNT1 (starts at 0)
  - when TCNT1 = OCRA1, an interrupt is issued and the timer is reset
  - by choosing OCR1A and the clock frequency, the timer can be programmed for any time interval
  - *dt* is the clock period divided by the prescaler you choose





Timer calculations

- How do I set the value of a control register ?
- The next tables are taken from the datasheet.

- RTFM ! (Read The *Fine* Manual) - the At Mega 16 datasheet, available either on Atmel's site or at:

  http://ham.elcom.pub.ro/proiect2/files/atmega16.pdf

- *Prescaler*: frequency divider, having a fixed set of values (e.g. 8, 64, 256, 1024); setting the prescaler changes the *dt* (basic timer interval, equal to the minimum amount of time).

- example: we want to program a 1-second interval using Timer1: look at the blue arrow:

Registers for Timer/Counter 1

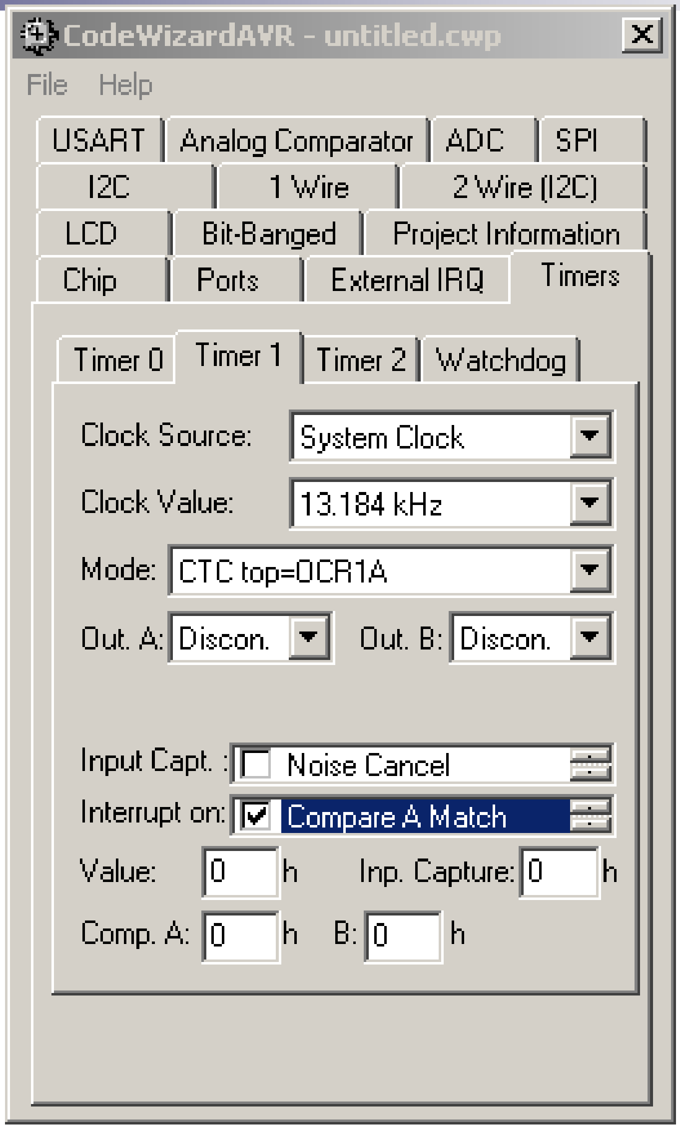
	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
	R/W	R/W	R/W	R/W	W	W	R/W	R/W	

	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
→ 4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

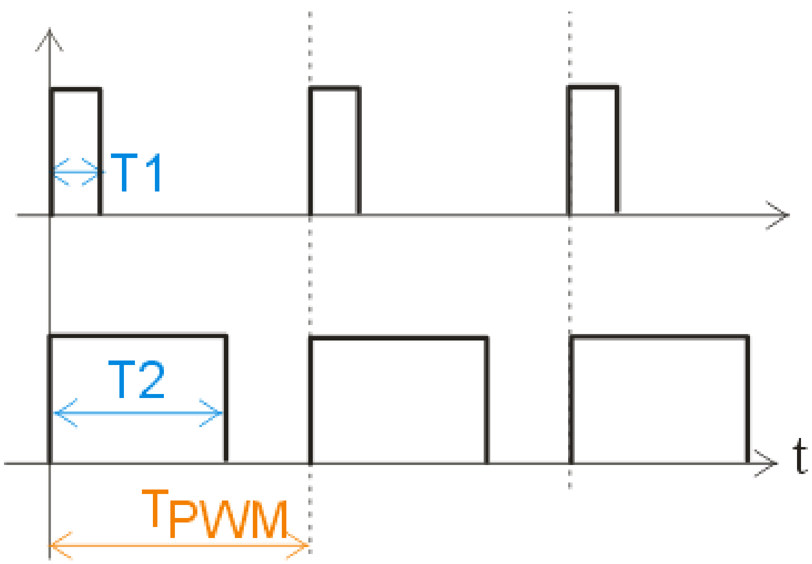
CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

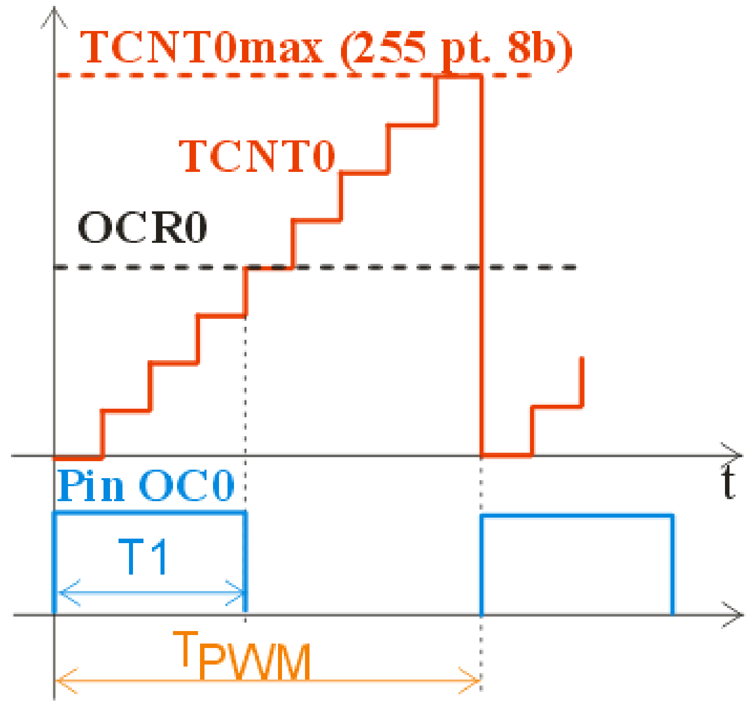
- we want 1 s = low frequency
- Example: $f_{Crystal}$ =13.5MHz→division by 13,500,000 > 65536 (16 bits) → impossible
- we need the prescaler to divide some more
- prescaler: max divisor = 1024; 13.5MHz / 1024 = 13.184KHz
- we want 1Hz: we div ide again by 13184 = 3380h
- OCR1AH = 33h, OCR1AL = 80h
- we select the CTC mode; let's set the remaining registers
- from the 2 previous tables: TCCR1A = 0 and
- TCCR1B= 00001101 = 0Dh



- All these calculations can be done using *CodeWizard*
- You still need to read the *datasheet* for the explanation of the different modes

- the PWM mode: useful for setting the speed of a motor or the light intensity of a light source
- example: use of the timer/counter0 in PWM mode to set the intensity level of a LED





  - TPWM is fixed; should be short enough to avoid flicker – if you choose a flicker-free frequency of 200Hz, then TPWM= 1/200Hz = 5ms
  - T1 < T2; the longer this interval, the longer you keep the LED on
  - connect the LED to pin OC0 so it is turned on automatically when TCNT0< OCR0 and turned off when TCNT0 >= OCR0
  - by changing the value OCR0, you change T1 and the intensity changes

Timer/Counter 0 Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

How to calculate the PWM frequency

- PWM → the frequency is constant, the duty cycle varies

- Example: assume $f_{crystal}$ = 13.5MHz
- We divide by:
  - prescaler: max 1024
  - maximum value for the 8 bit timer register: 256
  - we have f_{PWM} = 13500000/1024/256 = 51 Hz
  - Note: 51Hz is enough for light bulbs or motors, but a 51Hz flicker is visible on LEDs

  - we choose a lower prescaler: 256
  - f_{PWM} = 13500000/256/256 = 205 Hz

  - Prescaler=256 → CS02:00 = 100 (see previous table)

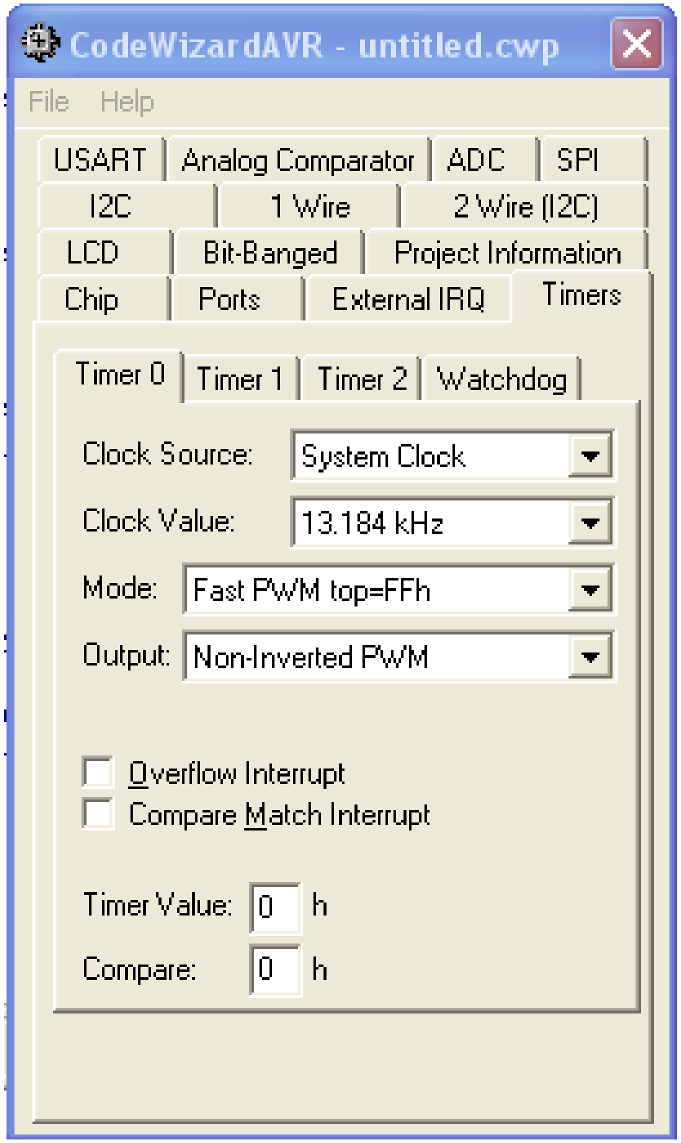
Timer/Counter 0 Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

- table COM 01:00 is for the Fast PWM mode
- we choose WGM 01:00 = 11, COM 01:00 = 10 CS 02:00 = 100
- the final value is: TCCR0 = 01101100 = 6Ch



- *CodeWizard* again

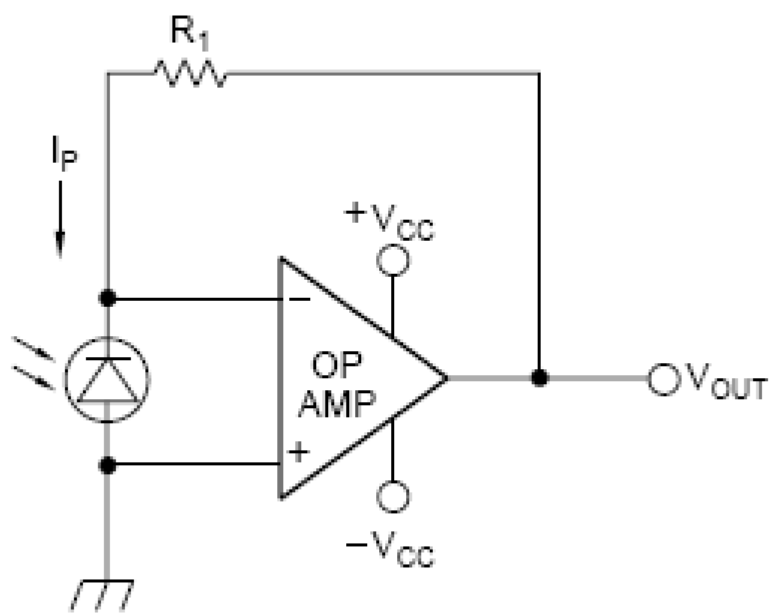
Sample program in PWM mode

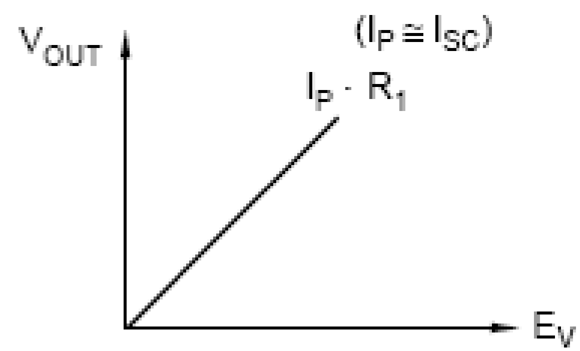
```
// timer0 init in PWM
// Clock source: System Clock/256, Clock value: 52734 Hz, Mode: Fast PWM top=FFh, OC0: Non-
    Inverted PWM
TCCR0=0x6C;
TCNT0=0x00;
OCR0=0x00;
// in PWM mode the OC0 pin is changed automatically so we don't need a timer interrupt !

// 4 different light intensities for LED, set using 4 different values of the OCR0 register
// pause 1 second between each intensity change
void main (void)
{
    while(TRUE)
    {
        OCR0 = 0; delay_ms(1000);        // no light
        OCR0 = 4; delay_ms(1000);        // little light
        OCR0 = 16; delay_ms(1000);       // medium light
        OCR0 = 253; delay_ms(1000);      // full light
    }
}
```

Sensors

- Digital sensors (TTL)
  - examples: contact switches, magnetic switches, optical switches, etc
  - states: LO and HI (only 2 values)
  - read on an input pin (PINX.y, not PORTX.y)
  - you may user a pull-up resistor so the HI state is default; pull LO by connecting the pin to ground → see the first circuit
  - internal pull-up: activate using PORTX.y=1 when the direction is set to "input" (DDRX.y=0)
  - use the same for analog sensors, when you need to detect the crossing of a treshold
- Analog sensors
  - many values (8 bits = 256 values; 10 bits = 1024 values)
  - use the internal A/D converter
  - 8 channels are built-in so you can read 8 separate inputs

Analog example: light sensor





- AO = 1/2 LM358 (-Vcc = 0V, +Vcc = +5V)
- The photodiode is reverse biased so we measure its dark current
- R1 = tens of KΩ up to 1M Ω

The Analog to Digital Converter (ADC)

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13 - 260 µs Conversion Time
- Up to 15 kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- 7 Differential Input Channels
- 2 Differential Input Channels with Optional Gain of 10x and 200x[1]
- Optional Left adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 2.56V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

- Specifications: Successive aproximations type
- kSPS = kilo Samples per Second
- Control registers: ADMUX, ADCSRA

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

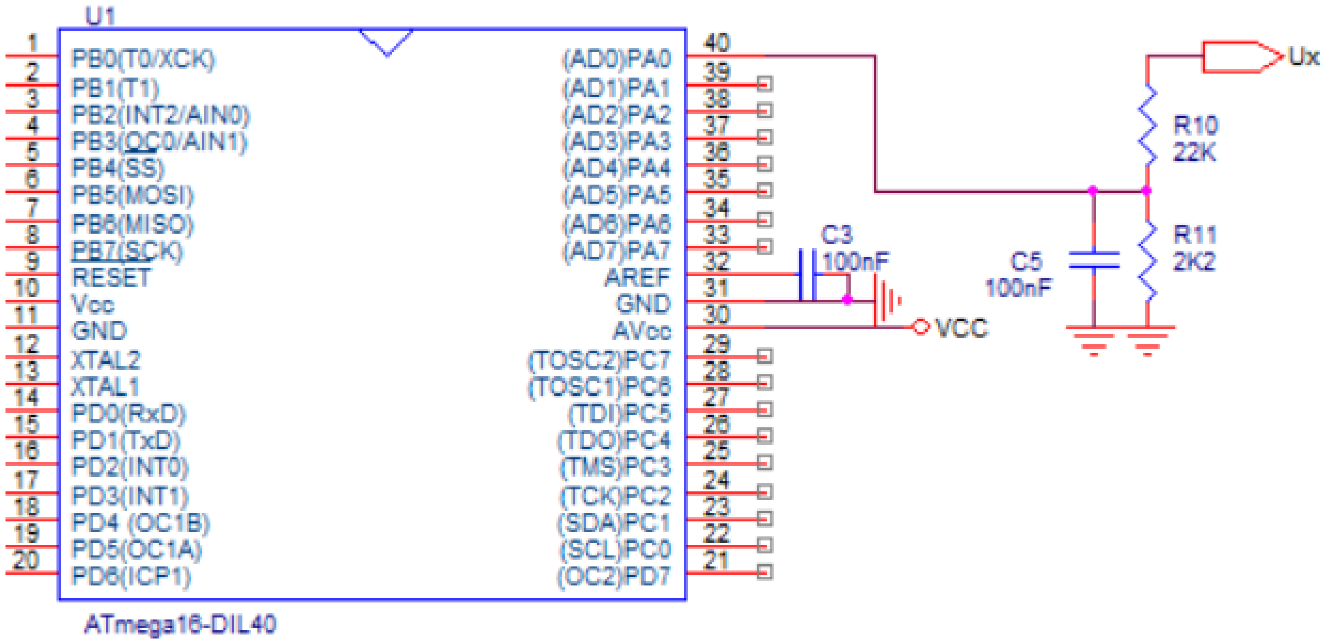
MUX4..0	Single Ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

- REFS: choose the reference
- differential modes also exist;
- ADLAR = AD *Left Adjust Result*
  - use ADLAR=1 if only 8 bits are needed; read only ADCH, containing the most significant 8 bits;
  - if you need 10b → ADLAR=0, read ADCH, ADCL
  - careful with the analog part if you want to use 10b !
- Input pins are AD0 to AD7 (on AT MEGA 16, pins 40 downto 33)

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- ADEN = ADC Enable
- ADSC = ADC Start Conversion; set to 1 to start a conversion in *Single Conversion* mode; in *Free Running* mode, set to 1 at the beginning
- ADATE = ADC Auto Trigger Enable; is used together with SFIOR
- ADIF = ADC Interrupt Flag; becomes 1 when the conversion is ready; automatically becomes 0 if the ADC ISR is executed (if ADC interrupts active)
- ADIE = ADC Interrupt Enable; also must set bit "I" in SREG
- ADPS 2:0 = prescaler for the ADC clock (= Crystal clock/prescaler)

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128



- Example: measuring a voltage larger than the reference voltage using the ADC
- R10, R11 form a divider which reduces Ux with the ratio K = 2.2/ (22+2.2) = 0.0909
- AD0 will read a voltage U0 corresponding to a number N (assuming 10 bits)

$$U0 / 2.56V = N / 1024$$

- (we assume that you select the Uref=2.56V which is more precise, using REFS0,1)
- thus, you calculate Ux in the software:

$$Ux = 2.56V / 1024 * N / K \quad \text{or} \quad Ux = 0.0275\ N\ \ [V]$$

- if you use only 8 bits:

$$Ux = 2.56V / 256 * N / K \quad \text{or} \quad Ux = 0.11\ N\ \ [V]$$

- C5 is optional, however it filters noise, by forming a LPF with R10.

```
#define ADMUX_NOCHANNEL 0b00100000            // see below ADMUX initialization

void init_adc(void)
{
// ADCSRA initialization; in order from MSB:
// 10 =  enable ADC, do not start a conversion yet
// 0 = disable free-running mode
// 10 = clear ADIF interrupt flag, disable ints
// 101 = ADC clock =XTAL/32
ADCSRA=0b10010101;
// ADMUX initialization
// 11 = internal VREF=2.56V ***OR*** 00=AREF= external reference on AREF pin
// 1 = ADLAR=1 (left adjust, use only 8 bits)
// the rest: channel selection
ADMUX=ADMUX_NOCHANNEL;                        // external AREF, ADLAR=1
}

// channel can be 0 to 7;
float read_voltage(byte channel)
{
channel &= 0b00000111;                        // 8 channels are possible
ADMUX = ADMUX_NOCHANNEL | channel;
ADCSRA |= 0b01000000;                         // start conversion
while (ADCSRA & 0b01000000);                  // wait for result in ADIF flag
ADCSRA |= 0b00010000;                         // clear ADIF flag
return 0.11 * (float)ADCH;                    // return value directly in volts
}
```