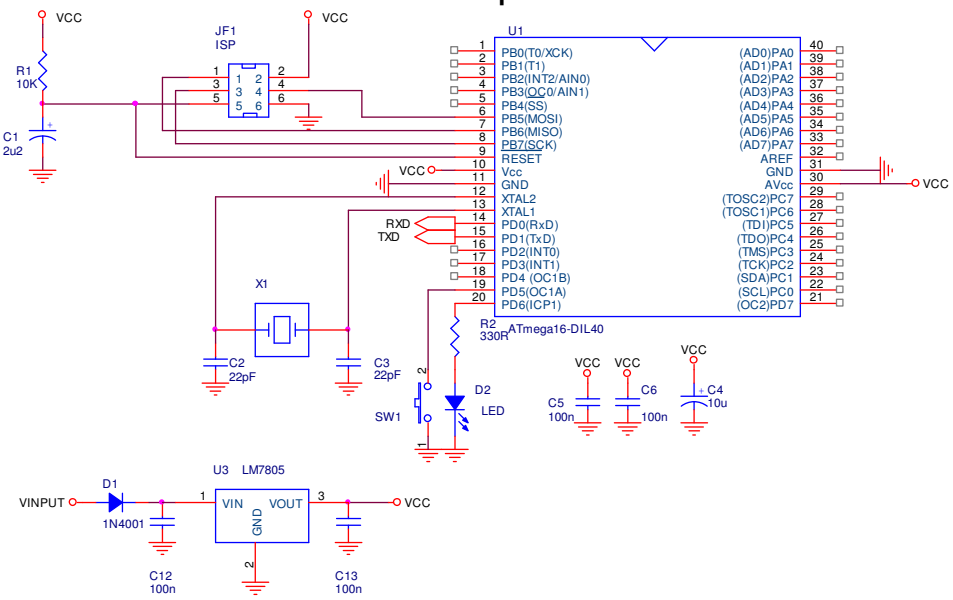
Aspecte practice privind utilizarea resurselor microcontrollerelor Atmel AVR

o mare parte din aceste materiale, inclusiv explicații în cuvinte, se află la:
http://ham.elcom.pub.ro → proiect 2 → documentație

Cel mai simplu circuit



- Ce face util: 1 pin de intrare (PD5), 1 pin de ieşire (PD6)

Pini de I/O

- *Pini de intrare:*
  - Initializare cu DDRX.Y = 0 *(Data Direction Register)*
  - Setează PORTX.Y = 1 pentru *pull-up resistor* intern
  - implicit PORTX.Y = 0 (fără *pull-ul resistor*)

  - Citeşte valoarea pinului X.Y folosind PINX.Y

  Exemplu:

```
If(PIND.5 == 0)    // citeşte switch conectat la D.5
  LED = 1
```

- *Pini de ieşire*:
  - Inițializare cu DDRX.Y = 1

  - Scrie valoarea pinului folosind PORTX.Y

  Exemplu:

```
PORTD.6 = 1    // aprinde LED conectat la D.6
```

- Notă: se pot accesa toți cei 8 pini simultan

```
PORTC = 0b11101011     // scriere
     sau
if(PINE == 0b10101011) {...}    // citire
```

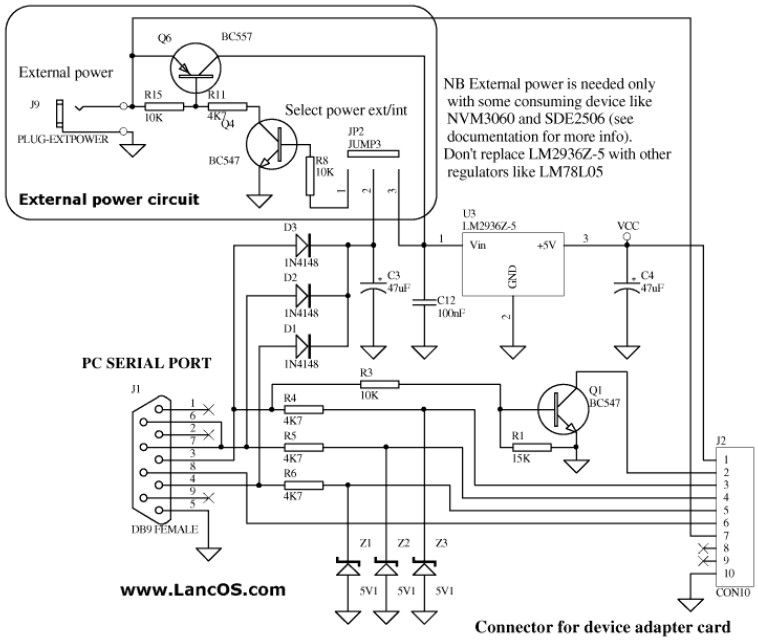
Programarea uC

- fişiere de intrare: *.c, *.h, etc
- fişier de ieşire: *.hex (formatul s.n. Intel HEX dar nu este specific doar Intel)

1. programarea clasică: uC scos din circuit şi introdus într-un programator
2. programarea *in-system*: uC rămîne în circuit în timpul programării; foloseşte conectorul dedicat ISP
3. programarea folosind *bootloader:* soft scris prima dată în uC, care preia programul pe un port disponibil în sistem, ne-dedicat (serial, USB, etc); *bootloader*-ul este un OS super-minimal

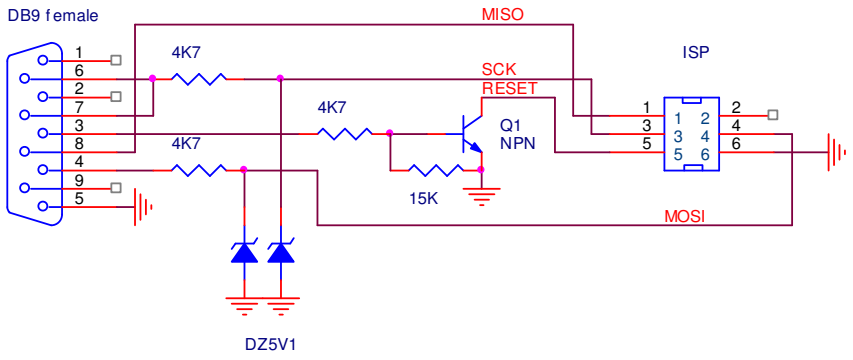
- urmează 2 scheme de programatorare ISP
- *bootloader:* după ce vedem portul serial

Programator extern, serial



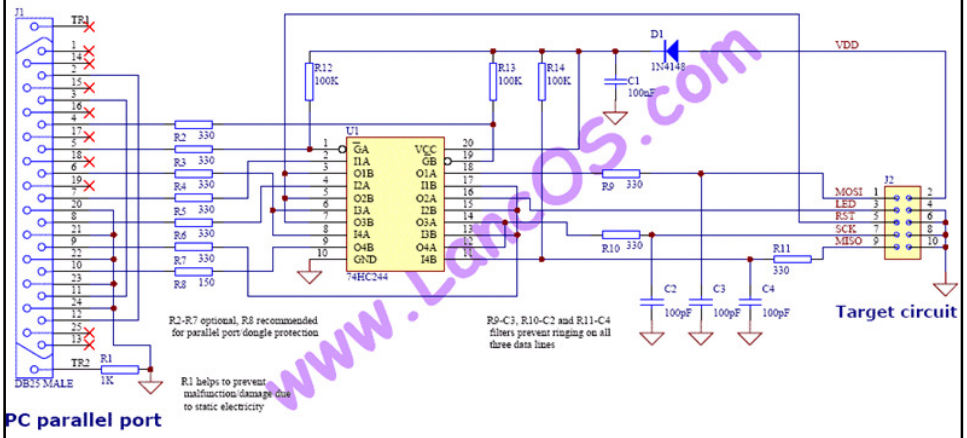
software:
PonyProg

Programator serial simplificat



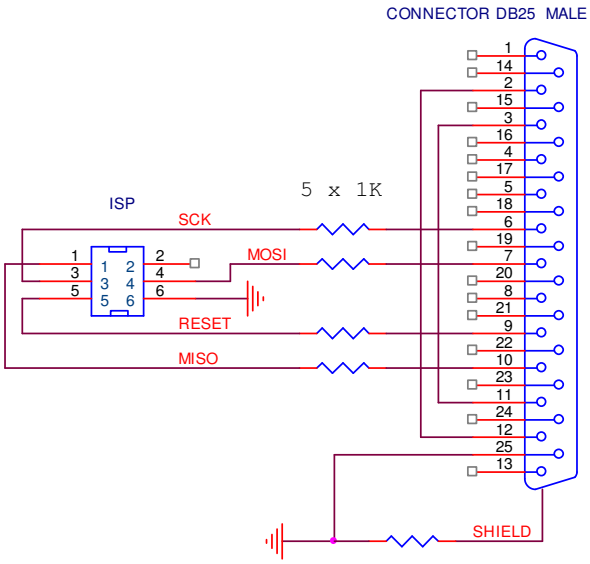
- schema precedentă, adaptată pentru plăcuța noastră (conector ISP 2x3, fără alimentare externă, căci plăcuța cu uC are deja alimentare)
- acelaşi pinout de conector ISP ca în schema cu uC

Programator extern, paralel



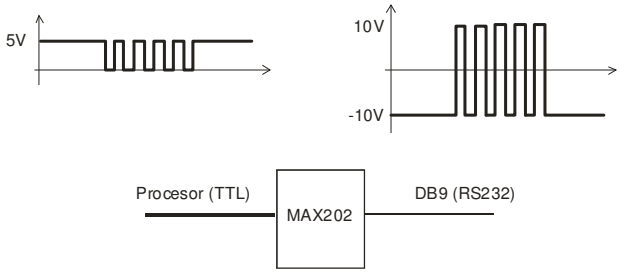
- programatorul paralel e mai rapid
- dezavantaje:
  - portul paralel de la PC e mai “fragil” decît cel serial
  - în lipsa sa, un convertor USB-paralel de obicei *nu merge cu această schemă*

Programator paralel simplificat



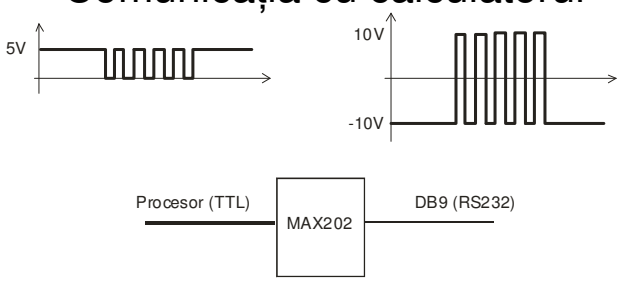
- s-au eliminat componentele de protecție !

Comunicația cu calculatorul



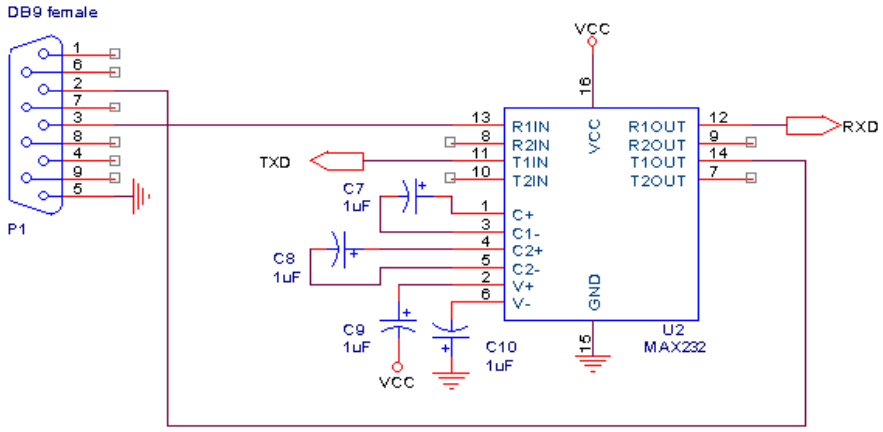
- uC conține deja un port serial (TTL) compatibil cu PC-ul
- trebuie doar un translator de *nivele electrice* (TTL ↔ RS232)
- utilitate:
  - debugging pe robotul *real*: printf() în programul uC nu scrie pe ecran, ci pe portul serial; se pot scrie valorile citite de la senzori (pentru calibrare), se pot trimite de la PC parametri modificați, comenzi, etc, *în timp real*
  - bootloader, pentru a nu mai folosi programator !

Comunicația cu calculatorul



- Structura unui caracter:
  - 1 start bit
  - 8 data bits (8D)
  - 1 stop bit
- Comparați cele 2 forme de undă: TTL, RS232
  (cea mai “naturală” este TTL)
- Cei 10 biți pe f.u. TTL: START, 8D, STOP = 0101010101
- NOTĂ: LSB transmis primul (MSB lîngă stop bit)
  deci numărul pe 8 biți se citeşte de la dreapta: 01010101

Translatorul de nivele seriale



- OBS: MAX232: 4 x 1µF; MAX202: 4 x 100nF
- TxD, RxD sînt aceiaşi ca pe schema cu procesorul

Programarea cu *bootloader* pe portul serial

- Prima dată tot trebuie scris *bootloader*-ul folosind un programator extern
- Vi-l pot scrie eu; disponibil pentru AT Mega 8,16,32,128 cu cuarț de 7.37MHz (toate uC) şi 8 şi 13.5MHz (doar AT Mega16)
- avantaj:
  - nu mai necesită programator extern !
- dezavantaje:
  - personalizat pentru un anumit tip de procesor şi cuarț
  - necesită port serial

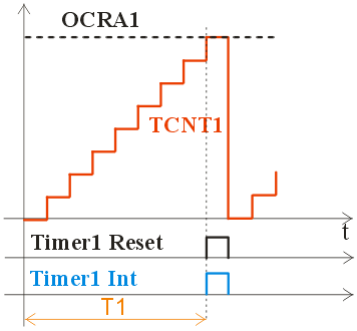
Întreruperi

- întreruperi *externe*: se activează la
  - recepționarea unui front/nivel pe anumiți pini
  - recepția unui caracter pe serială, etc
- întreruperi *interne*:
  - eveniment dat de un timer intern (ajunge la o anumită valoare, etc)
  - finalul unei conversii A/D, etc
- Vezi *datasheet* pentru lista întreruperilor *fiecărui* procesor
- în soft, o întrerupere e *deservită* (engl. *serviced*) de o funcție C numită ISR (ISR= *Interrupt Service Routine*)
- Vezi programul de test: întreruperea pentru Timer 1 face să clipească LED-ul de la PORTD.6 cu perioada de 1 sec.

Timere

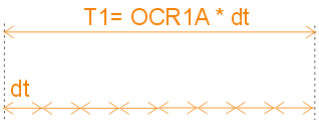
- Timer 0,1,2
- Alte uP au alt număr de timere
- 8 biți sau 16 biți
- sursa: clock intern, clock intern cu prescaler, clock extern (timer 2), pin special de intrare...
- prescalerul permite reducerea f_{CLOCK} → frecvențe mai mici → perioade mai mari
- multe moduri de lucru, trebuie citit *datasheet*-ul
- Folosite pentru *generarea* sau *măsurarea* unor intervale de timp
- Obs: pt generare, se poate folosi şi funcția *delay_ms(valoare)* dar ține procesorul blocat (nu lucrează pe intreruperi)

Aplicație 1: generarea unui eveniment periodic



- modul CTC (*Clear Timer on Compare Match*)
  - perioade de timp T1
  - după fiecare T1 are loc o întrerupere (*timer1 interrupt*)
  - codul scris de user în rutina întreruperii va fi executat periodic, cu perioada T1
- exemplu: timer 1 (16biți)
  - timerul este incrementat de la 0 folosind ceasul selectat
  - valoarea curentă este în TCNT1 (*Timer Count*)
  - cînd TCNT1 = OCRA1 (*Output Compare Register*), se generează o întrerupere şi se resetează timerul, după care incrementarea continuă de la 0
  - alegînd OCRA1 şi frecvența clock-ului se poate genera orice perioadă

Calcul detaliat



- Intervalul T1 = N subintervale dt
  - N = OCR1A
  - dt = ceasul sistemului, eventual divizat printr-un prescaler (divizor de viteză mare, avînd cîteva valori fixe)
  - dacă dorim T1 mare, tb. N mare şi dt mare
    - N mare: registru de 16b, nu 8b → verif. care timer e pe 16b
    - dt mare: prescaler cît mai mare
- exemplu: dorim întrerupere de timer la 1s folosind Timer 1
  - Folosim modul CTC
  - În registrele următoare setăm biții pentru mod, valoarea prescalerului, întreruperi
  - Cum se calculează valoarea unui registru de control ?
  - Foaia de catalog e sfîntă ! toate tabelele provin de acolo.
  - RTFM ! (Read The *Fine* Manual)
  - Apoi încărcăm val. calculate în TCCR1A, TCCR1B, OCR1

Registre 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	

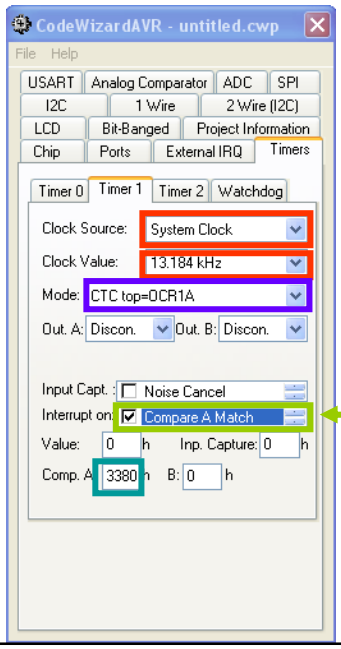
	Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
	0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
	1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
	2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
	3	0	0	1	1	PWM, Phase Correct, 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

Timer/Counter 1

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.

- Exemplu de calcul: dorim 1 s = frecvență mică
- $f_{cuarț}$ =13.5MHz→divizare cu 13,500,000 > 65536 (16 biți) → nu se poate
- trebuie prescaler
- prescaler max (**CS12:CS10 = 101**): 1024; 13.5MHz / 1024 = 13.184KHz
- avem 13184Hz, dorim 1Hz: mai divizăm cu 13184 = 3380h
- OCR1AH = 33h, OCR1AL = 80h
- mai rămîne să setăm divizorul şi modul de lucru; am ales CTC
  - din tabelul anterior şi acesta: TCCR1A = 0 şi
  - TCCR1B= 00001**101** = 0Dh

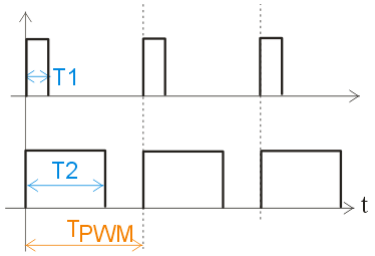
Good News !

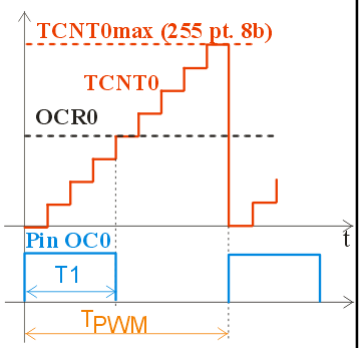


- Toate calculele pot fi făcute de catre *CodeWizard*
- culorile corespund calculelor precedente
- tot e necesar să citiți *datasheet*-ul pt. explicarea modurilor, a divizorilor etc

Compare A match → OCR1A
dacă era B → OCR1B

Aplicație 2: Modul PWM al timerelor





- modul PWM: util pentru varierea vitezei unui motor, etc
- T_{PWM} *fix*, factorul de umplere η *variabil* prin varierea T (T1,T2)
  - η1 = $T1/T_{PWM}$ mic → viteză mică
  - η2 = $T2/T_{PWM}$ mare → viteză mare

- La începutul unei perioade T_{PWM}, pinul OC0 setat pe 1
- TCNT0 incrementat automat pînă atinge maximul (255 pt 8biți)
- În momentul TCNT0= OCR0 pinul OC0 devine 0 pt restul perioadei T_{PWM}
- deci, pe pinul OC0 avem semnalul PWM din figură
- Pt fiecare timer (0,1,2...) avem un pin OC (*Output compare*) corespunzător

Calcul frecvență PWM timer0

- La PWM → frecvența constantă, factorul de umplere variabil
- Etape:
  - programăm la început frecvența constantă $f_{PWM} = 1/T_{PWM}$
  - în timpul funcționării variem fact. de umplere după dorință
  - varierea η → variem T1/T2 precedent → **variem OCR0**
- Exemplu: alegem frecvența pentru $f_{cuarț}$ = 13.5MHz
- factori de divizare:
  - presupunem prescaler de 1024
  - "umplerea" registrului de timer de 8 biți: 256
  - obținem f_{PWM} = 13500000/1024/256 = 51 Hz
  - OBS: 51Hz poate fi acceptabil la becuri/motoare, dar în cazul LED-urilor aceasta frecvență este vizibilă ca un uşor "flicker"
  - alegem un prescaler mai mic: 256, implicit f_{PWM} va fi mai mare
  - f_{PWM} = 13500000/256/256 = 205 Hz
  - deci folosim prescaler de 256 → CS02:00 = 100

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.

- programăm Timer0 în modul PWM
- alegem un prescaler de 256: CS02:CS00 = 100

Timer/Counter 0 Control Register

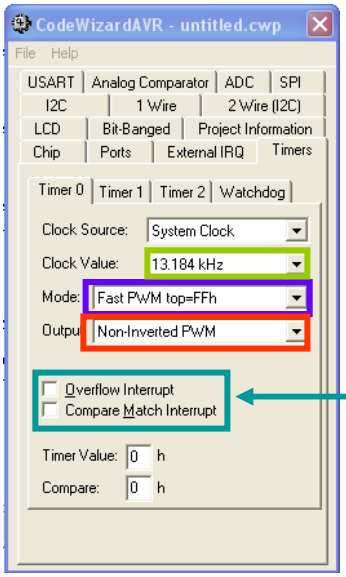
Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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)

- tabelul COM 01:00 valabil pentru modul Fast PWM
- alegem WGM 01:00 = 11, COM 01:00 = 10 CS 02:00 = 100
- în total: TCCR0 = 01101100 = 6Ch

Good news !



în modul PWM nu ne trebuie întrerupere de timer; resetarea timerului duce la trecerea în 1 (hardware) a pinului OC0, iar atingerea valorii OCR0 duce la trecerea în 0 a pinului, nu avem nimic de făcut într-o rutină de întrerupere

- *CodeWizard* din nou !

Programul în modul PWM

```
// intializam timerul 0 in modul PWM
// Clock source: System Clock/256, Clock value: 13500000/256=52734 Hz, Mode: Fast PWM
    top=FFh, OC0: Non-Inverted PWM
TCCR0=0x6C;   // valoarea din calculul precedent
TCNT0=0x00;
OCR0=0x00;

// programul variază intensitatea luminoasă a unui LED conectat la pinul OC0 în 4 paşi
// intensitatea se schimbă la cîte o secundă, schimbînd OCR0
void main (void)
{
    while(TRUE)
    {
            OCR0 = 0; delay_ms(1000);          // stins
            OCR0 = 4; delay_ms(1000);          // slab
            OCR0 = 16; delay_ms(1000);         // mediu
            OCR0 = 253; delay_ms(1000);        // tare
    }
}
```

PWM

- Avantaj PWM hardware (folosind modul PWM al timerelor): după inițializare, tot ce tb să facă programul e să scrimbe OCR în orice moment şi se schimbă automat factorul de umplere pe pinul OC corespunzător
- Dezavantaje:
  - uC nu are decît puține timere
  - dacă avem 3 timere: 3 canale PWM posibile, fiecare cu un pin OC corespunzător; dacă dorim să controlăm de ex. 10 motoare/becuri, nu putem
  - un timer PWM nu mai poate fi folosit la altceva (poate avem neoie de multe temporizări în aplicație)

PWM

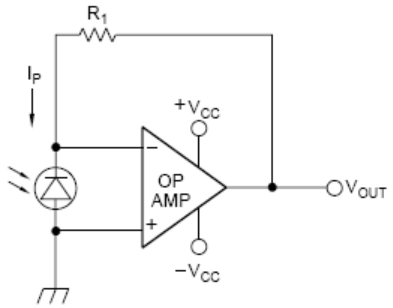
- Soluție: PWM software

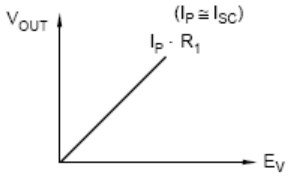
- Folosim un timer în modul CTC pt controlul *tuturor* celor 10 (sau oricîte) motoare, conectate la 10 pini de uz general
- Setăm perioada timerului la o valoare mică (ex: 1ms)
- în rutina ISR a timerului:
  - folosim un contor *i* pt a obține TPWM=ct; ex: i=50 → TPWM=50ms;
  - incrementăm *i;* cînd *i=50* facem i=0 pt a obține perioada de repetiție=ct
  - controlăm cîți pini externi (de uz general) dorim; cînd *i=0* îi setăm pe toți pe "1"
  - stabilim cîte praguri de comparație *i1,i2,...* dorim; cînd *i=i1* trecem pinul 1 pe "0", cînd *i=i2* trecem pinul 2 pe 0, etc
- *i1,i2,...* vor fi variabile globale; cînd programul principal le schimbă, automat se vor vedea şi în rutina ISR

Senzori

- Senzori cu ieşire digitală (TTL)
  - stări: LO şi HI
  - se citesc pe un pin de intrare (PINX.y, nu PORTX.y)
  - variantă: asigură doar starea LO, starea HI fiind implicită şi dată de o rezistență de pull-up; exemplu: tastă/microswitch conectată la masă (vezi primul circuit)
  - pull-up intern: se activează cu PORTX.y=1 cînd direcția e de intrare (DDRX.y=0)
  - pot fi şi senzori analogici (de lumină, cîmp magnetic etc) la care se detectează trecerea peste o valoare de "prag"
- Senzori cu ieşire analogică
  - se foloseşte convertorul A/D intern
  - maxim 8 canale = 8 senzori

Exemplu: senzor de lumină





- AO = 1/2 LM358 (-Vcc = 0V, +Vcc = +5V)
- Fotodioda e polarizată invers, pentru sensibilitate maximă !
- R1 = zeci.. sute KΩ (exemplu: semireglabil de 1M Ω)

Convertorul Analog-Numeric (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

- Specificații !
- kSPS = kilo Samples per Second
- Tip de conversie: aproximații succesive (AS); mai multe detalii → vezi cursul de IEM :-)
- Registre de control: ADMUX, ADCSRA

ADC

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: referința poate fi AVCC, AREF extern sau 2.56V (referință internă precisă)
- nu am inclus modurile diferențiale;
- ADLAR = AD *Left Adjust Result* :
  - dacă sînt suficienți 8 biți, se folosește ADLAR=1 şi se citeşte doar ADCH (conținînd cei mai semnificativi 8biți)
  - dacă se doresc 10b → ADLAR=0, se citesc ADCH, ADCL
  - atenție la execuția părții analogice dacă vreți să *vă bazați* pe 10b !

ADC

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; se pune pe 1 pentru a porni o conversie în modul *Single Conversion*; este 1 atîta timp cît conversia e în lucru, devine 0 cînd rezultatul e gata; în modul *Free Running* se pune pe 1 la început
- ADATE = ADC Auto Trigger Enable; se fol. în conjuncție cu registrul SFIOR
- ADIF = ADC Interrupt Flag; devine 1 cînd o conversie e gata şi se lucrează pe întreruperi; este automat trecut în 0 cînd se execută întreruperea ADC
- ADIE = ADC Interrupt Enable; în plus, trebuie setat bitul “I” în SREG
- ADPS 2:0 = cu cît se divizează XTAL pentru a obține ceasul ADC

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

Exemplu de folosire a ADC în modul *Sgl. conv.*

```
 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;

#define ADMUX_NOCHANNEL 0b00100000
// ADMUX initialization
// 00 = VREF=AREF
// 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= max value 111
ADMUX = ADMUX_NOCHANNEL | channel;      // select channel 0 to 7 (000 to 111)
ADCSRA |= 0b01000000;                   // start conversion by setting bit ADSC=1
while (ADCSRA & 0b01000000);            // wait for result; while working, ADSC is 1
ADCSRA |= 0b00010000;                   // clear ADIF flag
return (float)(ADCH) * 0.215;           // calibrate to whatever values are in the schematic
}
```