Descrierea părții comune a proiectului; utilizarea resurselor procesorului AVR

Cuprins

1. Procesorul şi sursa de alimentare

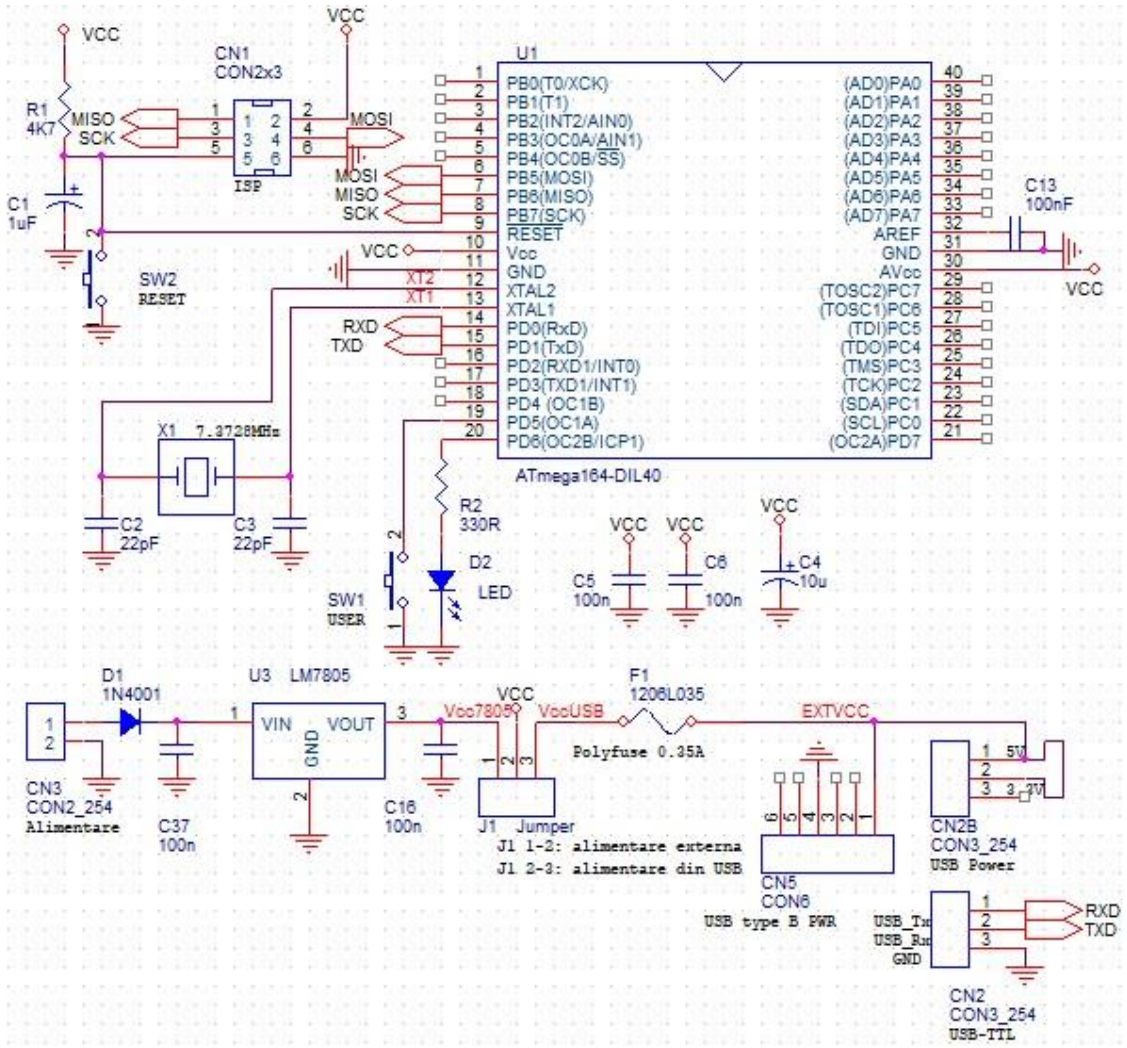


Fig. 1 Schema machetei

Schema din fig. 1 cuprinde componentele comune, care se vor regăsi pe toate machetele de proiect, indiferent de teme; la ele se vor adauga conectorii şi componentele specifice.
R1, C1 formeaza circuitul de “power-on reset”. În momentul aplicarii Vcc, C1 este descărcat şi “trage” linia

RESET* în “0”, resetînd procesorul. Ulterior, condensatorul se va incărca prin R1 şi linia va sta în “1” (procesul este asimptotic deci de durată infinită, dar convențional sfîrșitul încărcării se consideră dupa un timp aproximativ egal cu $5R_1C_1$). Acest reset este necesar pentru a asigura pornirea în bune condiţii a procesorului; în lipsa lui, tensiunile tranzitorii care apar în momentul alimentării pot duce la ajungerea procesorului intr-o stare incertă. Practic, linia RESET* e ţinută în “0” un timp semnificativ mai lung decît are nevoie sursa de alimentare să intre în regim staţionar.

Condensatoarele C5 şi C6 se află cît mai aproape de procesor, între pinii 10-11, respectiv 30-31. Ele sînt condensatoare de decuplare şi sînt specifice alimentarii oricărui circuit digital - asigură o “rezervă” de energie în momentul comutării, şi astfel previne apariţia zgomotului de comutare pe liniile de alimentare. Condensatorul electrolitic C4 nu este obligatoriu, dar este recomandat pentru reducerea riscului ca procesorul sa se reseteze din cauza unor glitch-uri pe alimentare.
Condensatorul C13 filtrează suplimentar tensiunea de referință AREF (V_{REF} ADC) și este necesar dacă se folosește convertorul analog-digital integrat.

Cristalul de cuarţ, împreuna cu condensatoarele C2, C3 şi cu amplificatorul intern de la bornele XTAL1,2 formeaza un oscilator cu cuarţ. Aceste componente trebuie să se afle, de asemenea, cît mai aproape de pinii respectivi ai procesorului.

Pentru alimentare se foloseste stabilizatorul U3 cu 3 terminale (7805), 2 condensatoare de 100nF necesare pentru stabilitatea functionării acestuia, şi dioda D1 care protejează la alimentarea inversă. Stabilizatorul preia o tensiune de 7..15V și scoate la ieșire Vcc=5V. Jumperul J1 va fi pe poziția 1-2 pentru folosirea lui 7805, și pe poziția 2-3 pentru scoaterea din circuit a lui 7805 (de exemplu dacă aplicăm Vcc extern de la USB, sau dacă dorim alimentare de la 3.3V).

Dioda LED este poziţionată astfel încît se aprinde când PD6 este pe “1” logic; rezistenţa R2 asigură limitarea curentului prin LED la cca. (5-1.6)/330= 10mA. De notat că în multe cazuri când se conectează un LED la ieșirea unui circuit digital, fără tranzistor de comanda, el se monteaza invers (între pinul circuitului și Vcc, evident cu anodul la Vcc), întrucît majoritatea circuitelor suportă un curent mai mare când pinul este în “0” logic decît atunci când e în “1”. Dezavantajul e ca LED-ul se aprinde pe “0”. În cazul nostru însă, procesorul AVR suportă curenti egali pe “0” şi pe “1” logic.

Butonul SW1 leagă PD5 la masa (0 logic) în momentul apăsării. Întrucît, atunci când nu este apăsat, starea pinului PD5 nu este definită, va trebui activată o rezistenţă de pull-up intern prin software.

CN1 este conectorul de programare ISP (*In-System Programming*); acesta permite programarea memoriei FLASH din procesor folosind un programator extern. De notat ca există mai multe standarde de conector (în principal 2x3 pini şi 2x5 pini), noi îl vom folosi numai pe acesta pentru compatibilitate. Deoarece procesorul pe care-l primiți este pre-programat de către noi cu un bootloader care permite programarea aplicației din FLASH pe serială, acest conector este opțional.

CN2 este conectorul care permite folosirea unui convertor TTL-USB pentru programare (cu bootloader) și comunicația serială.

Schema din fig. 1 este deja rutată pe un PCB ca în fig. 2. Fiecare pin liber al uC este adus la un pad adiacent, de unde se poate conecta, folosind un fir, la componente adiționale, care se pot lipi în zona de paduri libere (de tip placă de test). Spațierea padurilor libere este 100mils (2,54mm), permițînd plasarea oricăror piese și chiar circuite integrate DIP standard, și de asemenea sînt prevăzute amprente cu paduri spațiate la 50 mils (1,28mm), 25 mils (0,64mm) și 1mm, pentru eventuale circuit integrate în astfel de capsule.

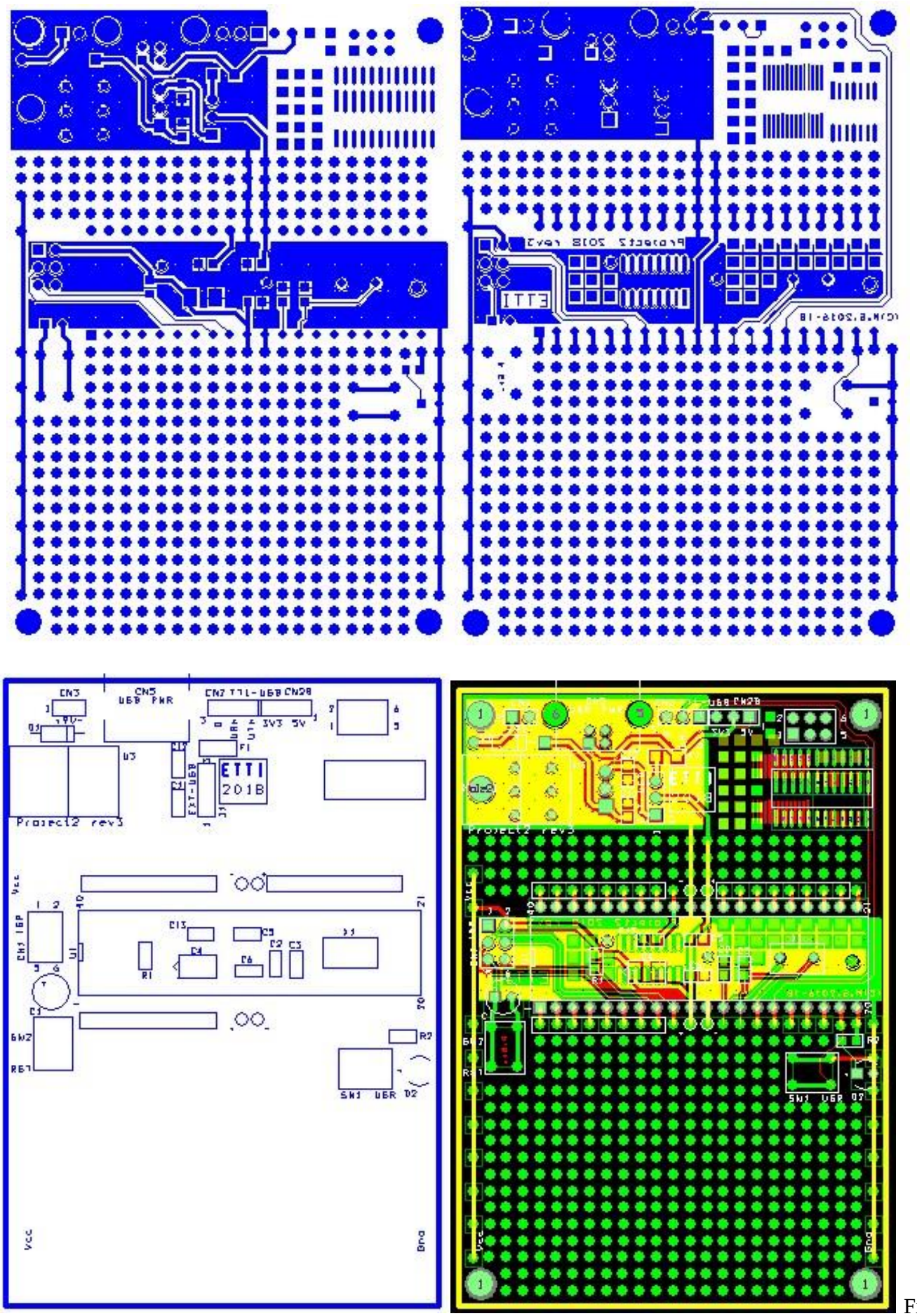


Fig. 2 PCB: Top, Bottom, Silkscreen Top și vedere generală

2. Portul serial

Microcontrollerele AVR includ 1 sau 2 porturi seriale UART de nivel TTL. Aceste nivele nu se folosesc uzual decît în interiorul plăcii; pentru comunicația cu un dispozitiv extern (calculator, etc) se pot folosi diverse variante:

- conversie la standardul de nivel RS232, folosit pe liniile de comunicații externe, prevede o tensiune între +6V..+15V față de masă pentru "0" logic și de -6V..-15V față de masă pentru "1" logic. Se observă că în RS232, logica este "inversată", și sînt necesare tensiuni negative, care nu sînt furnizate direct de către sursa de alimentare. În figura 3 este reprezentată forma de undă TTL şi respectiv RS232 pentru octetul "00110101". Există convertoare de nivel precum MAX232 care fac automat conversia bidirecțională între nivele
- conversie la standardul RS422/RS485; față de RS232, transmisia este diferențială pe doar 2 fire (fără masă) și este posibilă conectarea a mai mult de 2 dispozitive pe același cablu (bus). Există convertoare integrate de nivel RS485 precum SN75176.
- conversie la USB

Cei 8 biţi de date sînt încadraţi de un bit de start, care va fi întotdeauna "0", şi un bit de stop, care va fi întotdeauna "1". Practic, pe figura 3 se observă că bitul "1" de stop durează mult mai mult decît ceilalţi biţi, dar aceasta se întîmplă pentru că, în absenţa datelor, linia este ţinută în "1" logic, deci după ce se transmite acest bit "1" starea liniei nu se schimbă, până la transmiterea unui nou octet, care va începe cu un nou bit de start "0".

Atenţie! Folosind notaţia binară obişnuită b7b6b5b4b3b2b1b0, unde b7 este MSB şi b0 este LSB, biţii pe serială se transmit *LSB-first*, nu *MSB-first* cum ar fi mai intuitiv. Aşadar, după bitul de start (0, stînga) urmează LSB (b0), iar MSB (b7) este adiacent bitului de stop (1, dreapta)

Biţii de stop şi de start garantează faptul că la începutul fiecărui octet are loc o tranziţie "1 → 0", chiar şi în cazul în care octeţii transmişi reprezintă lungi şiruri de "1" sau "0". Dezavantajul este că la fiecare 8 biţi de date trebuie adăugaţi cei 2 biţi, adică eficienţa transmisiei este de doar 80%.

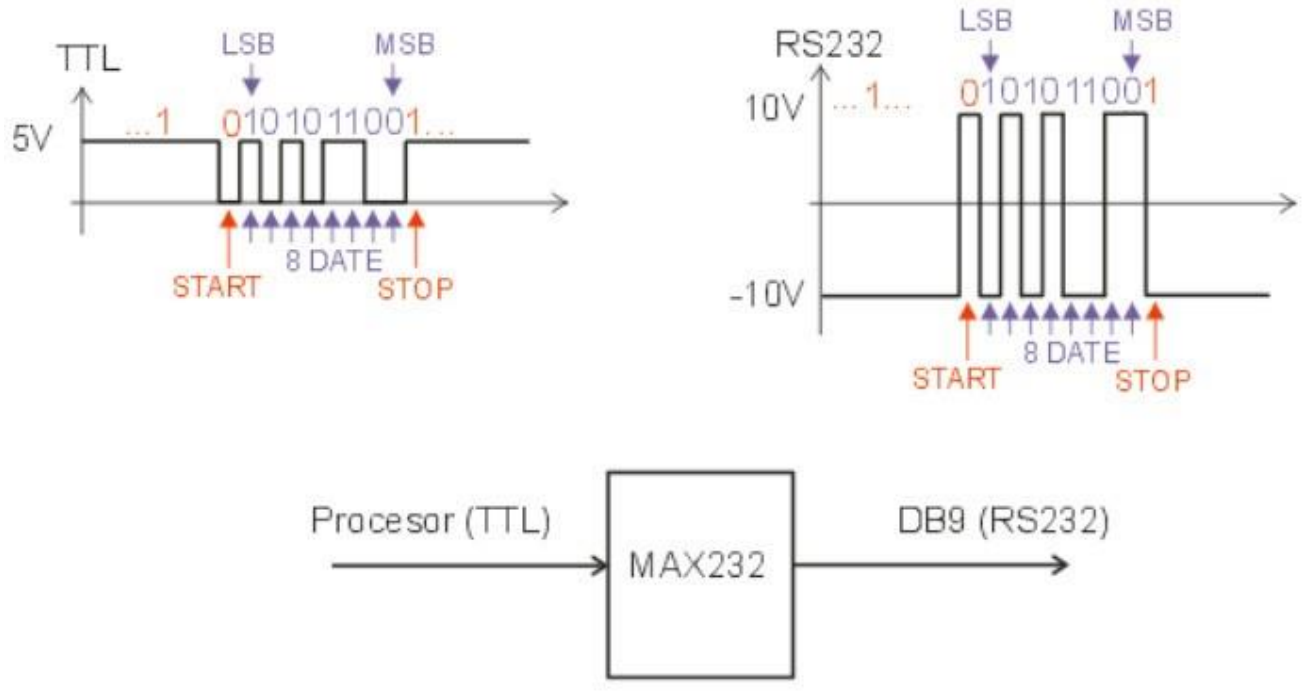


Fig. 3 Forme de undă la TTL şi RS232 pentru octetul de date 00110101. Formele de undă TTL sînt disponibile și pe intrarea convertorului TTL-USB.

2.1. Vizualizarea formelor de undă; depanarea portului serial

Se va folosi un osciloscop cu care să se vizualizeze formele de undă în mai multe puncte. Pentru 9600bps un caracter are 10 biţi deci durează:

$$10 \text{ biţi} \cdot 1/9600 \text{ sec/bit} \approx 1\text{ms}$$

Pentru ca un caracter (10 biţi) să ocupe toate cele 10 diviziuni pe ecran, determinăm C_X :

$$T_X = N_X C_X \;\rightarrow\; C_X = T_X / N_X = 1\text{ms} / 10 \text{ div} = 0.1 \text{ ms/div}$$

Atenţie la reglajul de sincronizare! Bitul de start este ca în fig. 3: un front negativ de la 5V la 0V pe nivele TTL (deci *slope= falling*)

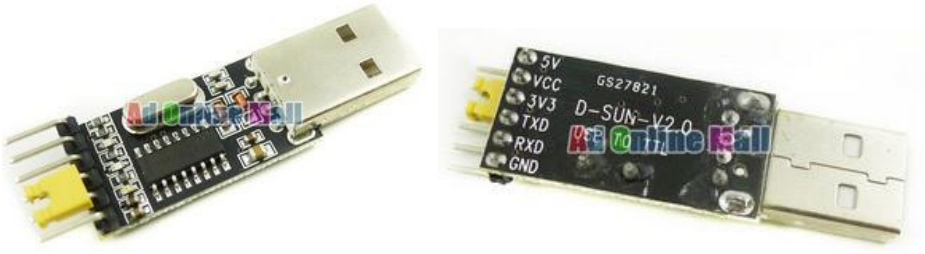
Cu procesorul avînd programul de test montat pe placă, se va ţine apăsată o tastă pe tastatura PC-ului (după ce se porneşte programul de terminal). Folosind softul de test, macheta trebuie să trimită înapoi caracterul următor: dacă primeşte “a” trimite “b”, etc.

Se urmăreşte pe osciloscop ordinea evenimentelor (crocodilul de masă al osciloscopului se poate conecta la aripioara metalică a lui LM7805):

1. de la PC, folosind terminalul, se emite un caracter (RS232) care
2. ajunge la procesor (uP pin 14) – vizualizați codul ASCII pe osciloscop !
3. acesta trimite caracterul cu codul imediat următor (uP pin 15) – vizualizați și acest cod !
4. caracterul+1 ajunge înapoi la PC.

2.2. Folosirea unui convertor TTL-USB; alimentarea (opțională) din USB

Se conectează convertorul TTL-USB la pinii conectorului CN2. Atenție la respectarea ordinii pinilor RxD, TxD, GND, de obicei convertorul TTL-USB are pinii denumiți după convenția DTE, deci RxD, TxD de la USB trebuie conectați respectiv la Tx, Rx de la uP (conectare *Cross*). De asemenea, trebuie instalat pe PC driverul convertorului CH340 și folosit portul COM virtual (COM3, etc). Uneori driverul îl instalează ca COM9 sau alt număr mare, pe care unele programe nu îl văd; în acest caz, se vor deschide proprietățile driverului (în *Device Manager*) și se va seta un număr de COM între 1 și 4, ca să fie accesibil.



uP	*USB*
RXD	TXD
TXD	RXD
GND	GND

Fig. 4 Convertoare USB-TTL

La convertoarele care au selecția tensiunii de 5V sau 3V3, vom selecta 5V, pentru a fi compatibil cu nivelele de tensiune ale uP alimentat la 5V.

Când se folosește convertorul USB-TTL, alimentarea plăcii se poate face în două moduri (vezi și fig. 1):

- Alimentare din sursa externă conectată la CN3. ***Jumperul J1 de pe placă se pune pe poziția 1-2 (EXT).***

  Se conectează convertorul la PCB cu doar 3 fire (USB_Tx, USB_Rx și GND), se folosește doar conectorul CN2, iar pe convertor se pune jumperul între pinii „5V” și „Vcc”. În acest caz, din USB de alimentează doar logica din interiorul convertorului, alimentarea PCB-ului fiind în continuare din 7805 și sursa externă. ***Nu lăsați placa fără alimentare externă !***
- Alimentarea din USB (5V). ***Jumperul J1 se pune pe poz. 2-3 (USB), iar la poziția F1 trebuie lipită o siguranță polimerică 1206 sau măcar o bucată de sîrmă subțire.*** Această variantă se poate folosi dacă nu aveți consumatori mari pe placă (motoare, becuri cu filament etc) și dacă nu vă trebuie tensiuni mai mari de 5V.

  Se conectează toți cei 6 pini, la conectorii CN2, respectiv CN2B care au fost plasați pe PCB cu terminalele în ordine identică cu cea de la convertorul TTL-USB. În acest caz, nu mai e loc de jumper, de aceea pe PCB este deja făcută conexiunea între „Vcc” și „5V” (pinii 1,2 ai CN2B), în locul jumperului de pe convertor, ca în figura 5a

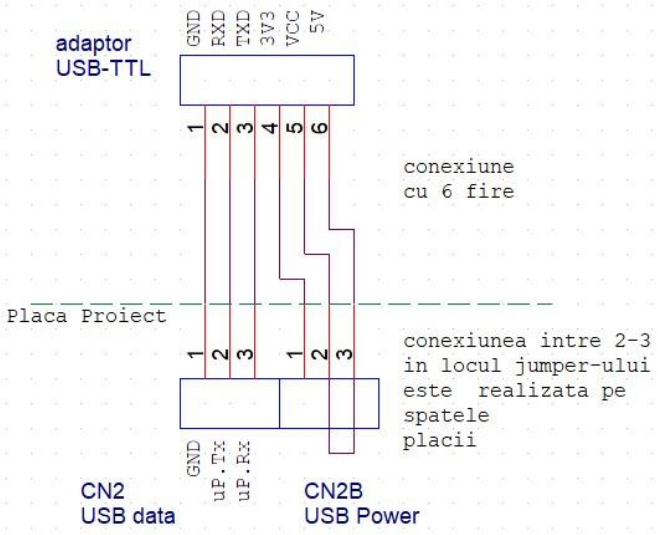


Fig. 5a conexiunea cu 6 fire pentru alimentarea plăcii din portul USB

Schema convertorului (bazată pe cipul CH340) este dată în fig. 5b:

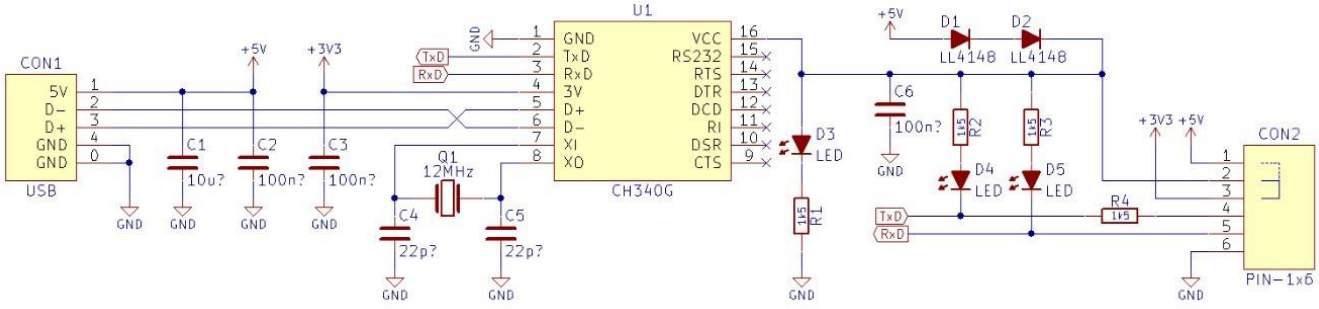


Fig. 5b Schema convertorului USB-TTL cu CH340

Se observă că reducerea tensiunii de la 5V la 3.3V se face prin înserierea a 2 diode (D1, D2), pe fiecare căzînd cca. 0.6-0.7V. Dacă jumperul este între pinii 1-2 aceste 2 diode sînt scurtcircuitate și întreaga tensiune de 5V de la USB ajunge pe pinul Vcc al cipului. Pinul V3 este o intrare și se leagă fie la 3V3 (când Vcc=3.3V), fie se lasă neconectat (când Vcc=5V).

Observație alimentarea se poate face și printr-un cablu USB de tip A-B, conectat la CN5. Observați pe figura 1 că la acest conector sînt lipite doar terminalele de alimentare și masă, nu și cele de date, deci pentru comunicație trebuie folosit, în continuare, convertorul USB-TTL.

2.3. Alimentarea la 3.3V (opțional)

Procesoarele AVR suportă alimentare la tensiuni mult mai mici de 5V. De ce am dori să alimentăm la mai puțin? în cazul acestui proiect, motivul principal este conectarea unui circuit periferic de 2.7 - 3.3V, cu multe linii de date (este posibilă și conectarea când cipul adițional este la 3V și uP la 5V, folosind circuite de adaptare pe fiecare linie de date; vezi documentația; soluția este nepractică când sînt multe astfel de linii).
Pentru a trece alimentarea plăcii pe 3.3V se procedează astfel:

1. jumperul J1 de pe PCB se scoate de tot
2. se adaugă un stabilizator de 3.3V pe placă (de exemplu, LF33)
3. pinul central al J1 (pinul 2), sau orice punct marcat Vcc pe schemă, se conectează printr-un fir la ieșirea noului stabilizator
4. pentru folosirea convertorului USB-TTL, se *taie* cu un cutter traseul de pe PCB dintre pinii CN2B 1-2 (fig. 5a) și se adaugă o conexiune între 2-3. Alternativ, dacă nu se folosește CN2B, se plasează un jumper direct între pinii VCC și 3V3 ai adaptorului.
5. alimentarea la 3.3V direct din convertorul USB-TTL este teoretic posibilă, dar nu este recomandabilă; așa cum se vede pe schema convertorului din fig. 5b, se obțin aprox. 3.3V din 5V prin înserierea a 2 diode (D1 și D2), tensiunea nefiind foarte precisă și, mai ales, curentul printr-o diodă 4148 este max 100mA, deci nu se poate folosi decît la consumuri mici ! dacă *totuși* se dorește acest lucru (pe răspunderea voastră; dacă vă stricați convertorul USB-TTL trebuie să vă procurați singuri unul nou), se va efectua doar punctul 4 de mai sus, și se va pune jumperul J1 pe poz. 2-3 pentru a uni linia de "3.3" cu Vcc-ul schemei.

2.4. Vizualizarea formei de undă; depanarea comunicației seriale în cazul adaptorului USB-TTL

În cazul folosirii acestui convertor, formele de undă se vor urmări numai pe nivele TTL. Formele de undă pe ieșirea USB sînt mult mai complexe, nu este o simplă translație de nivel ca în cazul TTL-RS232; spre deosebire de MAX232 care lucrează doar la nivelul fizic, convertorul la USB este un circuit care implementează protocolul USB. Acesta este și motivul pentru care folosim port serial și nu USB nativ: majoritatea uC au pe cip doar port serial, nu și USB, datorită complexității acestuia din urmă.

Depanarea comunicației seriale USB-TTL: pentru a vedea că funcționează, se va proceda astfel:

- se conectează convertorul USB-TTL la PC (fără a-l conecta și la PCB-ul nostru), se verifică în Windows → Device Manager sau în lista de porturi din CodeVision Settings → Terminal că apare un port serial adițional, de ex. COM3. Dacă nu apare, poate fi necesară instalarea manuală a driverului de CH340G

- se pornește terminalul și se setează din Settings → Terminal portul COM corespunzător, precum și opțiunea „Echo Transmitted Characters" pentru a vedea ce se transmite. Atenție! nu setați COM1 (dacă există), acela este portul serial fizic RS232 existent pentru majoritatea PC-urilor Desktop.

- se apasă o tastă oarecare în terminal; LED-ul de Rx de pe adaptorul USB-TTL trebuie să clipească, indicând faptul că acesta primește caracterele de la terminal

- se scoate jumperul de pe convertor și se pune direct între pinii Rx, Tx ai acestuia a.î. caracterele primite sînt întoarse înapoi; în acest moment trebuie să clipească ambele LED-uri (Tx și Rx) și caracterele transmise să apară dublate. Observație: acesta e motivul pentru care, în softul de test, am ales să transmitem *caracter+1* în loc de *caracter*; în acest mod, știm sigur că este un caracter nou generat de uP, nu un „ecou" al caracterului primit de la PC.

- se pune la loc jumperul de pe convertor pe poziția 5V și se conectează acesta cu cablu la placa rulînd softul de test. Caracterele trimise trebuie să se întoarcă sub forma char+1.

3. Pini ai procesorului de I/O de uz general

Cu excepția pinilor de alimentare și de cuarț, toți pinii procesorului pot fi folosiți în mod independent unul de altul ca pini *digitali* de **intrare** (la care se pot lega butoane, senzori etc) sau de **ieșire** (la care se pot conecta LED-uri sau alte elemente de acționare).
Observație: în acest paragraf sînt prezentați pinii de tip *digital* care pot fi doar „0" sau „1"; pragurile exacte de tensiune care corespund acestor nivele logice depind de tensiunea de alimentare și pot fi găsite în *datasheet*-ul procesorului. Cazul pinilor *analogici* de intrare este tratat separat în secțiunea despre convertorul analogdigital (în acest caz uC-ul poate citi și cuantiza orice valoare de tensiune analogică, nu doar „0" și „1").

Pinii de I/O sînt grupați în 4 porturi PORTA, PORTB, PORTC, PORTD de cîte 8 pini, aceștia fiind numiți PORTA.0 .. PORTA.7 etc. Dacă este nevoie în aplicație de toți cei 8 pini ai unui port (de exemplu PORTA să fie un „bus" paralel de 8 biți între uP și un alt dispozitiv), valorile celor 8 pini pot fi scrise/citite simultan folosind variabila PORTA (pt. scriere) sau PINA (pt. citire) – acestea sînt denumiri ale registrelor, care pot fi folosite ca variabile în limbajul C în compilatorul CodeVisionAVR.
Dar, oricare pin de la A.0 la A.7 poate fi scris/sau citit ca un singur bit, folosind variabila de tip „bit" PORTA.0 .. PORTA.7, respectiv PINA.0 .. PINA.7 (similar pt. port B, C, D).

În interiorul uC, fiecare pin are atît interfața electrică de intrare, cît și ieșire (schemele se găsesc în *datasheet*), dar în mod evident, doar una poate fi activă la un moment dat; utilizatorul trebuie să aleagă una din ele prin alegerea **direcției** pinului, în funcție de ce anume a conectat fizic la pinul respectiv. Direcția se inițializează ca mai jos pe baza registrelor DDRA, DDRB, ca mai jos:

1) Pini de intrare:
- Initializare cu DDRX.Y = 0 (Data Direction Register) - Citeşte valoarea pinului X.Y folosind PINX.Y
Exemplu:

```
if(PIND.5 == 0)     // citeşte switch conectat la D.5
     ...
```

Evident, unui pin de intrare nu are sens să-i setăm o valoare, căci atunci ar fi contradicție cu direcția de intrare. Totuși, prin convenție, la uC AVR scrierea unui „1" în PORTX.Y configurat cu direcția de intrare leagă în circuit rezistorul de *pull-up* intern, dacă utilizatorul dorește.
Observație: la AVR există rezistor intern de *pull-up*, dar nu și de *pull-down* disponibil; dacă este necesar un rezistor de *pull-down*, acesta se va monta separat pe machetă, între pin și masă. Reamintim (de la CID) că aceste rezistoare sînt de valori mari (tipic, peste 10K) și țin intrarea într-o stare precizată (1 sau 0) atunci cînd aceasta nu este conectată la nimic în exterior. De exemplu, în schema din figura 1, PIND.5 va fi „1" datorită *pull-up* cînd butonul nu este apăsat, căci în acest caz pinul este „în aer", și fără *pull-up* starea ar fi neprecizată.

```
DDRD = 0x00;// pini de intrare
PORTD.5 =1  // pentru pull-up resistor intern, switch legat la masa
            // implicit PORTX.Y = 0 (fără pull-ul resistor)
```

2) Pini de ieşire:
- Iniţializare cu DDRX.Y = 1

- Scrie valoarea pinului folosind PORTX.Y Exemplu:

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

Observație: direcția se poate schimba de oricîte ori se dorește, nu doar la pornirea programului; de exemplu, dacă un port e folosit ca bus (magistrală), direcția va fi alternată în 1/0 pentru a scrie, respectiv a citi bus-ul.

Notă: se pot accesa toți cei 8 pini simultan:

```
PORTC = 0b11101011     // scriere
```

sau

```
if(PINB == 0b10101011) {...}    // citire
```

Atunci când se dorește scrierea/citirea a mai mult de 1 bit, dar mai puțin de 8, se pot *masca* ceilalți, pentru a nu fi afectați, folosind valori numerice binare și funcțiile artimetice AND (&), OR (|). Se pornește de la următoarele relații, valabile pt un bit x oarecare:

- x & 1 = x (& cu 1 nu modifică bitul x, oricare ar fi acela)
- x | 0 = x (| cu 0 nu modifică bitul x, oricare ar fi acela)
- x & 0 = 0 (& cu 0 șterge bitul, indiferent de vechea sa valoare)
- x | 1 = 1 (| cu 1 setează bitul pe 1, indiferent de vechea sa valoare)

Reamintim că în C sintaxa `X &= Y` este sinonimă cu `X = X & Y` și la fel pentru alți operatori.

Exemplul 1: setăm pe 1 cei mai semnificativi 3 biți din portul C, fără a-i modifica pe ceilalți:

```
PORTC |= 0b111 00000;
```

Exemplul 2: dacă primii 5 biți ai portului A sînt egali cu 10101, execută o operație

```
if(PINA & 0b11111000 == 0b10101000) ... // ultimii 3 biți vor fi citiți
                                        // 0, primii 5 se vor păstra
```

Exemplul 3: setăm direcția „intrare” (0) pentru pinul B.4 și „ieșire” (1) pentru B.3, lăsînd pe loc ceilalți biți

```
DDRB &= 0b111 0 1111;       // B.4 = 0
DDRB |= 0b0000 1 000;       // B.3 = 1
```

Exemplul 4: dorim să scriem valoarea V pe 4 biți în biții 3,2,1,0 ai portului B, fără să modificăm ceilalți biți ai portului

```
unsigned char V;    // 8 biți
V = ...             // asignare la un moment dat
V &= 0b0000 1111; // ne asigurăm că biții 7,6,5,4 ai lui V sînt 0, dacă
                    // nu știm sigur acest lucru
PORTB &= 0b1111 0000;      // ștergem biții 3,2,1,0 din port, fără a umbla
                         // la biții 7,6,5,4
PORTB |= V;              // copiem biții de „1” pe pozițiile 3,2,1,0
                         // din V în port
```

4. Convertorul Analog-Digital (ADC)

Procesorul AVR include un ADC cu aproximații succesive de 10 biți cu 8 canale independente, care ocupă cei 8 pini ai portului A. Când se activează (folosind registrul ADMUX) citirea unei valori analogice de pe unul din pini, respectivul pin este conectat la ADC și eventualele setări de direcție și valoare digitală dată de registrele DDRA și PORTA se ignoră (setarea registrelor ADC are prioritare față de registrele DDR și PORT); pinii sînt independenți și pot fi folosiți o parte ca ADC și altă parte ca I/O digitali, în orice combinație.
ADC-ul folosește o tensiune de referință care poate avea valorile de 2.56V sau 1.1V (obținute dintr-o referință internă precisă de tip *bandgap*) , VCC (tensiunea de alimentare) sau AREF (valoarea aplicată pe pinul cu acest nume, care poate fi diferită de VCC).
Specificațiile ADC sînt următoarele:

Features

- 10-bit Resolution
- 0.5 LSB Integral Non-Linearity
- ±2 LSB Absolute Accuracy
- 13 - 260µs Conversion Time
- Up to 15kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- Differential mode with selectable gain at 1x, 10x or 200x(1)
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- 2.7V - V_{CC} Differential ADC Voltage Range
- Selectable 2.56V or 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

Reamintim că pentru un ADC unipolar (*engl. single-ended*) cu 10 biți, numărul corespunzător unei tensiuni U este:

$$N = [\ 1024 \bullet U/U_{REF}\ ]$$

unde prin [ ] am notat operația de trunchiere.
Registrele de control ale ADC sînt ADMUX și ADCSRA, descrise mai jos:

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	

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Internal 1.1V Voltage Reference with external capacitor at AREF pin
11	Internal 2.56V Voltage Reference with external capacitor at AREF pin

MUX[4:0]	Single Ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

Fig. 6 setarea registrului ADMUX al ADC

ADMUX permite setarea referinței și a canalului pe care se măsoară, la o măsurătoare putîndu-se citi valoarea unui singur canal la un moment dat. Condensatorul extern din tabel este C13 de pe fig. 1.
Cei 10 biți ai rezultatului sînt stocați în 2 registre de 2 biți ADCH, ADCL (sau se pot citi împreună accesînd registrul ADCW) ; întrucît 10b < 16b, rezultatul scris *b9b8b7b6b5b4b3b2b1b0* poate fi aliniat în două moduri:

	ADCH	ADCL
ADLAR = 0	*b9b8*	*b7b6b5b4b3b2b1b0*
ADLAR = 1	*b9b8b7b6b5b4b3b2*	*b1b0*

În cazul ADLAR=1 (*AD Left Adjust Result*), se observă că ADCH conține cei mai semnificativi 8 biți ai rezultatului; dacă dorim să folosim doar 8 biți, setăm deci ADLAR=1 și citim doar ADCH, ignorînd ADCL. Pentru multe aplicații, ultimii 2 biți oricum corespund unor valori prea apropiate de nivelul zgomotului din circuit, de aceea se consideră că 8 biți sînt suficienți, ducând la o rezoluție:

$$V_{LSB\ 8b} = 2.56V/256 = 10mV$$

Observație: atunci când MUX3, MUX4 au alte valori în loc de 00 (vezi datasheet), este posibilă măsurarea diferențială, adică diferența dintre tensiunile de pe două canale ADC, în loc de cazul *single-ended* când oricare canal se măsoară față de masă. Întrucît în modul diferențial tensiunile pot fi ***negative***, rezultatul este reprezentat în ***complement față de doi***. Reamintim formula: numărul N pe *n* biți este reprezentat în complement față de 2 sub forma:

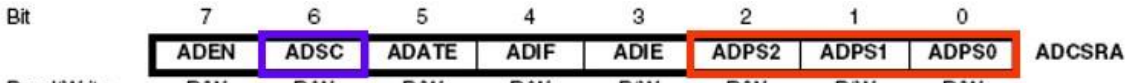
$$C2(N) = 2^n - N$$

ceea ce corespunde faptului că bitul MSB devine bit de semn, toate numerele cu MSB=1 fiind negative, conform tabelului:

Număr binar	Număr hexazecimal	Semnificație în cazul unipolar	Semnificație în cazul bipolar (complement față de 2)
0000000000	0x000	0	0
0000000001	0x001	1	1
...	...	...	...
0111111110	0x1FE	510	510
0111111111	0x1FF	511	511

1000000000	0X200	512	-512
1000000001	0x201	513	-511
...	...	...	...
1111111110	0x3FE	1022	-2
1111111111	0x3FF	1023	-1

În limbajul C, compilatorul va folosi automat reprezentarea în C2 pentru tipuri de date *signed char, signed int* etc. Se observă că domeniul numerelor disponibile este redus la jumătate: [0,N) devine [-N/2, N/2) unde $N=2^n$



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	

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

Fig. 7 setarea registrului ADCSRA al ADC

Descrierea biților ADCSRA:

- 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 când rezultatul e gata; în modul *Free Running* se pune pe 1 la început
- ADATE = ADC Auto Trigger Enable; se folosește în conjuncție cu registrul SFIOR și permite conversia continuă (*Free Running*), caz în care sfîrșitul unei conversii determină începutul următoarei conversii.
- ADIF = ADC Interrupt Flag; devine 1 când o conversie e gata şi se lucrează pe întreruperi; este automat trecut în 0 când se execută întreruperea ADC
- ADIE = ADC Interrupt Enable; în plus, trebuie setat bitul "I" în SREG. La terminarea conversiei va fi apelată funcția ISR (*Interrupt Service Routine*) a procesorului.
- ADPS 2:0 = cu cît se divizează XTAL pentru a obține ceasul ADC; se va alege o valoare în funcție de cuarțul disponibil, a.î. să nu se depășească viteza maximă de lucru a ADC-ului (vezi *datasheet*).

Exemplu de proiectare hard + soft folosind ADC:

Se dorește măsurarea unei tensiuni U_x de max. 24V pe canalul ADC0. La intrarea canalului respectiv se introduce un divizor, în cazul nostru de minim 1/10, care permite măsurarea tensiunilor de cel puțin 10 ori mai mari decît tensiunea de referință aleasă (2.56V). Alegem referința internă de 2.56V pentru că e mult mai

precisă (mai stabilă cu timpul, cu temperatura și în valoare absolută) decît tensiunea de alimentare de 5V a machetei.

Se vor alege valori de rezistențe disponibile în mod uzual; se vor folosi rezistențe de cel mult 1% toleranță și de valoare suficient de mare a.î. puterea disipată să fie sub 10% din puterea nominală (tipic 0.125W, 0.25W sau 0.5W pentru rezistențele comune), în caz contrar rezistențele se vor încălzi și valoarea va fi afectată din această cauză. Pentru cazul nostru, alegem divizorul cu rezistențe uzuale, obținînd factorul K < 1:

$$K = 2.2/ (22+2.2) = 0.0909$$

astfel încît la intrarea canalului 0 vom citi o tensiune $U_0 = KU_x$ care, reprezentată pe un număr N de 10 biți față de o referință de 2.56V, satisface relația:

$$U_0 / 2.56V = N / 1024$$

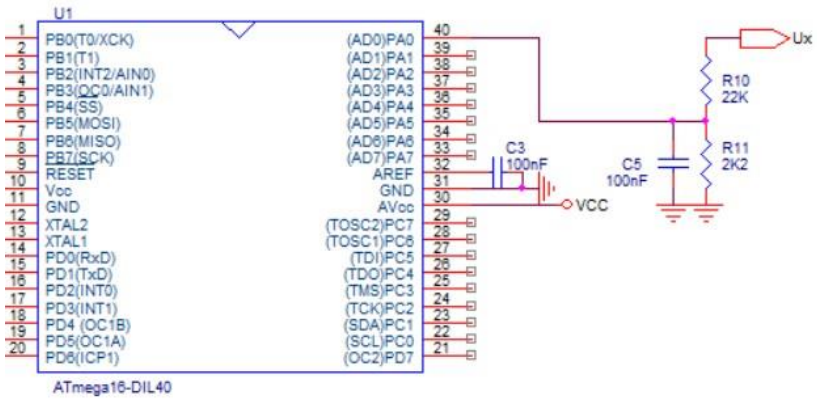


Fig. 8 Modul de conectare al unui canal ADC cu divizor 1/11

Rezultă deci relația de legătură dintre N și U_x :

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

Condensatorul C3 (C13 pe fig. 1) de pe pinul AREF (unde se poate măsura tensiunea de 2.56V) este necesar pentru stabilitatea și reducerea zgomotului tensiunii de referință, iar condensatorul C5 împreună cu R10 formează un FTJ care reduce zgomotul tensiunii măsurate. Valoarea C5 va depinde de viteza cu care variază semnalul de măsură, valori mari fiind potrivite pentru semnale lente (sau aproape staționare).
Mai jos este un exemplu de cod ce folosește ADC în modul *single conversion*, pentru oricare din cele 8 canale 0..7 (*single ended*). Nu se folosesc întreruperile, deci se așteaptă rezultatul conversiei până când bitul ADSC devine 0. Pentru conversii dese sau continue, pentru a nu ocupa procesorul, se recomandă lucrul pe întreruperi, citirea registrului ADCH sau ADCW făcându-se în rutina de întrerupere a ADC.

```
// initializare, se va apela inainte de prima conversie
// 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 0b11000000
// ADMUX initialization
```

```
// 11 = VREF=2.56V
// 0 = ADLAR=0 (no left adjust – use 10b )
// the rest: channel selection

// functia de citire a canalului 0..7;  float
read_voltage(unsigned char channel)  {
channel &= 0b00000111;          // max 111, orice alți biți tb. puși pe 0 ADMUX =
ADMUX_NOCHANNEL | channel;  // selectare canal 0 .. 7 (000 .. 111)
ADCSRA |= 0b01000000;                    // start conversie setînd bit ADSC=1
while (ADCSRA & 0b01000000);             // așteaptă rezult; ADSC=1 în timpul
                                         conv.
ADCSRA |= 0b00010000;                    // șterge flag ADIF
return (float)(ADCW) * 0.0275;           // calibrați cf. divizorului vostru  !!!
}
```

5. Senzori

La pinii de intrare ai uC se pot conecta două categorii de senzori: Senzori cu ieşire digitală (TTL):

- stări: LO şi HI;
- se citesc pe un pin de intrare (PINX.y, nu PORTX.y care e pt. ieșire)
- variantă: asigură doar starea LO, starea HI fiind implicită şi dată de o rezistenţă de pull-up; exemplu: tastă/microswitch conectată la masă (vezi primul circuit)
- pull-up intern: se activează cu PORTX.y=1 când direcţia e de intrare (DDRX.y=0)
- pot fi şi senzori analogici (de lumină, cîmp magnetic etc) conectați la un port digital, la care se detectează trecerea peste o valoare de "prag" (vezi *Datasheet* pentru tensiunea minimă care este interpretată ca „1" logic, în funcție de tensiunea de alimentare).

Senzori cu ieşire analogică:

- se folosește convertorul A/D intern
- maxim 8 canale = 8 senzori
- mai mulți senzori se pot citi dacă adăugăm un multiplexor extern (de ex. 74HC4051).

Exemplu de senzor de lumină folosind o fotodiodă:

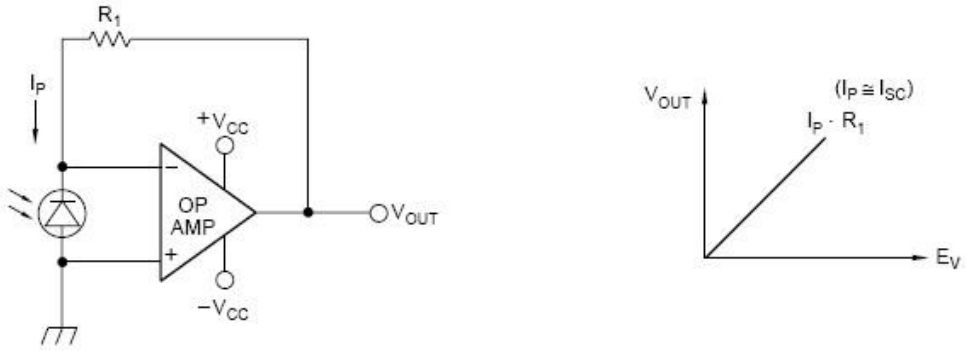


Fig. 9 Folosirea unei fotodiode ca senzor de lumină

AO = 1/2 LM358 (-Vcc = 0V, +Vcc = +5V)
Fotodioda e polarizată invers, pentru sensibilitate maximă !

R1 = zeci.. sute KΩ (exemplu: semireglabil de 1M Ω), întrucît dacă dorim sensibilitate mare, curentul I_P este mic (la întuneric sau în condiții de lumină foarte redusă), și dorim o tensiune de ieșire de ordinul V. Tensiunea V_{out} este:

$$V_{out} = I_p R_1$$

Pentru aplicații unde nu este nevoie de sensibilitate mare, putem folosi și alte fotoelemente (de exemplu fototranzistoare, fotorezistențe). O fotorezistență poate fi conectată într-un divizor rezistiv cu o rezistență obișnută, tensiunea obținută prin divizare fiind astfel dependentă de lumină, fără folosirea unui AO sau a altui element activ. Fiecare fotoelement are avantajele și dezavantajele sale; de exemplu, fotorezistența e relativ puțin sensibilă, dar funcționează în bandă mai largă și este mai ieftină decît fototranzistorul.

6. Timere

Timerele sînt folosite pentru generarea sau măsurarea unor intervale de timp. Procesorul nostru are Timerele 0,1,2; alte uP au alt număr de timere (aceasta este principala diferență dintre diferitele variante de uC-uri AVR, AtMegaXXX: diferă numărul de periferice)

Timerele pot fi pe 8 biţi sau 16 biţi. Un timer de 16 biți are registrul de numărare de 16 biți (valoarea maximă 65535) deci poate genera durate mai lungi de timp, la același ceas, decît unul de 8 biți.
Sursa: clock intern, clock intern cu prescaler, clock extern (doar pentru timer 2), pin special de intrare, etc (vezi *datasheet*).
Prescalerul este un divizor opțional care permite reducerea f_{CLOCK} (frecvenţe mai mici → perioade mai mari de temporizare)

Timerele au multe moduri de lucru, trebuie citit *datasheet*-ul pentru o descriere completă.

Obs: pt generarea unui interval de timp (pauză), se poate folosi şi funcţia delay_ms(valoare) dar ţine procesorul blocat (nu lucrează pe intreruperi).

Aplicație 1: folosirea Timer1 pentru generarea unui eveniment periodic (OBS: se folosește Timer 1 pe 8 biți; verificați în datasheet dacă în cazul procesorului vostru, timerul are 8 sau 16 biți; în cazul cu 16 biți registrele se dublează, de ex OCR1A devine OCR1AH + OCR1AL)

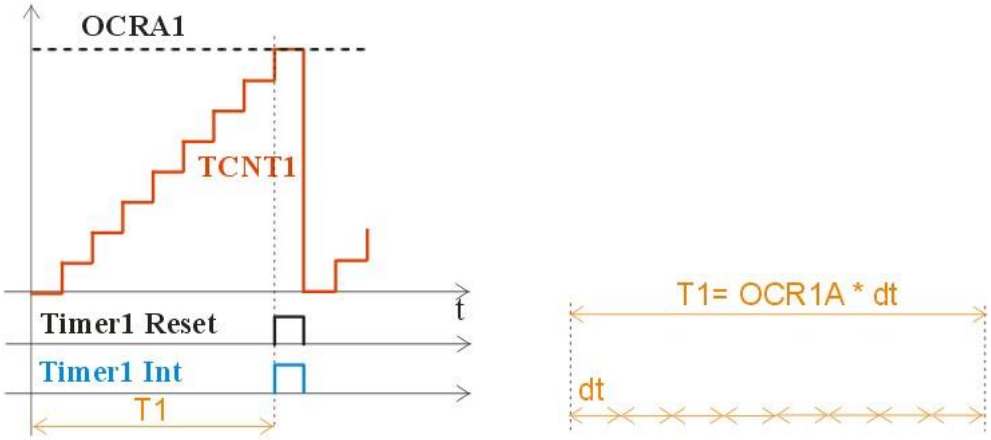


Fig. 10 Timerul în modul CTC

Vom folosi modul CTC (Clear Timer on Compare Match) care înseamnă că valoarea din registrul de numărate TCNT1 crește până la atingerea unei valori (*compare match*) stabilite de utilizator OCRA1, după care este adusă la 0 automat (*clear timer*) și procesul se repetă.
Conform figurii 9, timerul va genera perioade de timp T1; după fiecare T1 are loc o întrerupere (timer1 interrupt). Dacă de exemplu rutina de întrerupere aprinde (și apoi stinge un LED), între 2 aprinderi vom avea exact T1.

exemplu: timer 1 (16biți):

- timerul este incrementat de la 0 folosind ceasul selectat
- valoarea curentă este în TCNT1 (Timer Count) și este incrementată la fiecare ceas.
- când TCNT1 = OCRA1 (Output Compare Register), se generează o întrerupere şi se resetează timerul, după care incrementarea continuă de la 0
- alegînd OCRA1 şi frecvenţa clock-ului se poate genera orice perioadă

După cum se vede pe fig. 9, intervalul T1 este format din N subintervale *dt* unde:

- N = OCR1A
- *dt* = ceasul sistemului direct sau divizat printr-un prescaler (divizor de viteză mare, avînd cîteva valori fixe)
- dacă dorim T1 mare, trebuie N mare şi *dt* mare ○ N mare: registru de 16b, nu 8b; verificați în datasheet care timer e pe 16b !
    - ○ *dt* mare: prescaler cît mai mare deci frecvență cît mai mică.

Exemplu numeric:
Dorim o întrerupere de timer la exact 1 secundă folosind Timer 1

- Cum se calculează valoarea unui registru de control ? Toate tabelele următoare provin din *datasheet*.
- RTFM ! (*Read The **Fine** Manual*)
- Folosim modul CTC – alegem modul WGM13,12,11,10 = **0100** din primul tabel.
- În registre setăm biții pentru mod, valoarea prescalerului, întreruperi
- Apoi încărcăm valorile calculate în TCCR1A, TCCR1B, OCR1A (registru de 16b compus din OCR1AH și OCR1AL).

În tabelele următoare, culorile corespund biților selectați:

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.

Fig. 11 Setarea registrelor pentru Timer1 modul CTC

Detaliem calculul: dorim perioada de 1 s (frecvenţă mică)

- presupunem că dispunem de un cuarț de 13.5MHz
- $f_{cuarț}$ =13.5MHz → trebuie divizare cu 13500000 > 65536 (16 biţi) → nu se poate direct

→ trebuie prescaler (în general, cu cît ceasul e mai rapid și evenimentul dorit e mai lent, cu atît factorul total de divizare, dat de produsul dintre prescaler și registrul de comparare al timer-ului, trebuie să fie mai mare).

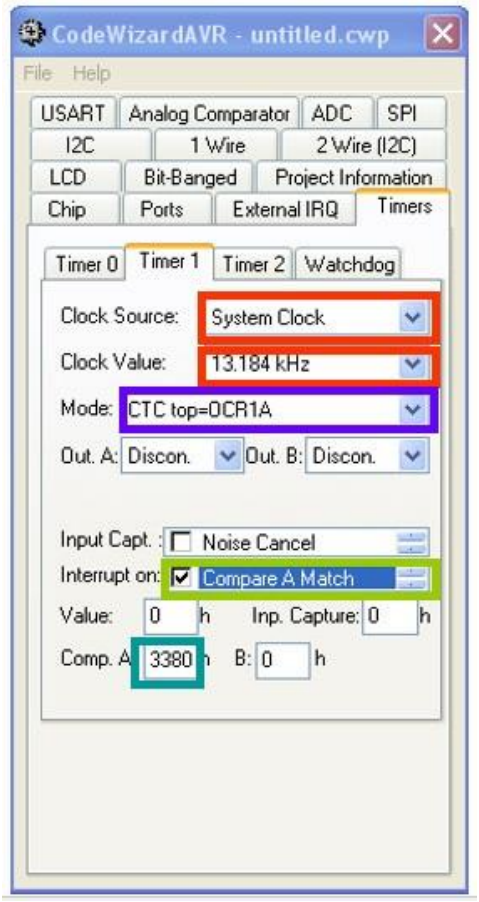
- alegem prescaler max = 1024 (**CS12:CS10 = 101**); 13.5MHz / 1024 = 13184Hz
- am ajuns la 13184Hz, dorim 1Hz: mai divizăm cu 13184 = 3380h

→ OCR1AH = 33h, OCR1AL = 80h

Mai rămîne să setăm divizorul şi modul de lucru; am ales CTC, iar din din tabelele anterioare rezultă:

- TCCR1A = 00000000 şi
- TCCR1B= 00001101 = 0Dh

Setarea registrelor poate fi făcută manual, sau folosind CodeWizard ca mai jos (culorile corespund calculelor precedente).



Folosind opțiunea „program preview” în CodeWizard cu valorile de mai sus, se obțin aceleași valori finale pentru TCCR1A, TCCR1B, OCR1A, în codul de mai jos:

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 13.184 kHz
// Mode: CTC top=OCR1A
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
```

```
OCR1AH=0x33;
OCR1AL=0x80;
OCR1BH=0x00;
OCR1BL=0x00;
```

De notat că timerul 1 are 2 registre „Output Compare” numite OCR1A și OCR1B, deci poate genera 2 valori de temporizare.

Pentru întreruperea de timer, vezi codul generat de CodeWizard sau exemplul din softul de test. În general, întreruperea trebuie să conțină un minim de operații, pentru a dura cît mai puțin, mai ales dacă timerul este setat la o valoare mică (rapidă). De exemplu, la un program de ceas, în întrerupere doar se incrementează valorile unor variabile globale precum secunde, minute, ore etc, partea de afișare făcându-se în programul principal, citind valorile acestor variabile globale.

Aplicație 2: folosirea modului PWM al timerelor

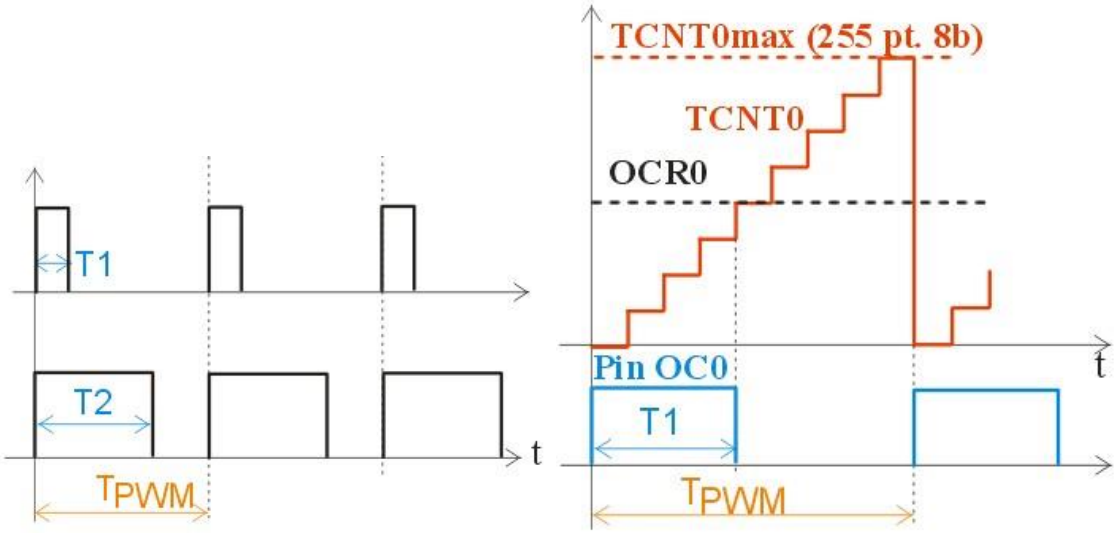


Fig. 12 Modul PWM al timerelor

După cum se știe, PWM (*Pulse Width Modulation*) înseamnă un semnal cu perioada constantă (T_{PWM} pe figura 12, stînga) și factorul de umplere variabil, cu aplicație directă în reglajul unei mărimi analogice de ieșire: intensitatea luminii unui LED, viteza unui motor, etc. Cu cît factorul de umplere este mai mare, cu atît *media* semnalului de la ieșire este mai mare, deci, de exemplu LED-ul luminează mai tare.

Observație: un factor de umplere de ½ înseamnă că media semnalului de amplitudine U este U/2 (deci 2.5V pentru cazul TTL), dar aceasta nu înseamnă că luminozitatea LED-ului va fi percepută ca jumătate din maxim, întrucît răspunsul ochiului uman nu este liniar. De asemenea, pentru aplicații unde contează puterea (de exemplu, reglăm la ½ comanda unui element de încălzire) trebuie luat în calcul că puterea depinde de pătratul tensiunii, deci vom avea un sfert din puterea aplicată.

Pe figura 11, cei 2 factori de umplere sînt $\eta_1 = T_1/T_{PWM}$, respectiv $\eta_1 = T_2/T_{PWM}$ cu $\eta_1 < \eta_2$. În modul PWM hardware numit “fast PWM”, timerul poate genera impulsurile de duratele dorite, și în plus poate comanda automat un pin (OC0 pentru Timer0, etc) să urmărească valoarea T_1 ca pe figura 12:

- cît timp $t < T_1$, OC0=1 deci LED aprins • când $T_1 < t < T_{PWM}$, OC0=0 deci LED stins
- apoi, când se atinge sfîrșitul perioadei T_{PWM}, LED-ul se stinge și ciclul se repetă.

Observație: un dezavantaj al modului PWM hardware este că funcționează *numai* pentru pinul OC corespunzător (OC0 pentru Timer0, OC1 pentru Timer1, etc). Pentru anumite aplicații, poate fi necesar să comandăm mai mulți pini în PWM. În acest caz, se implementează PWM în software, folosind un timer în modul obișnuit (CTC) de viteză mare, care incrementează un contor, și trecând în 1, respectiv 0, la anumite valori ale contorului, pinii doriți, totul în rutina de întrerupere a timerului.

Pentru ATMega164, OC0A (vezi fig. 1) este PB3 adică pinul 4 în capsula DIP40. Prin urmare, LEDul trebuie conectat la acest pin și direcția DDRB trebuie setată ca ieșire pentru B.3 (setînd timerul 0 în modul PWM, portul B.3 capătă funcția alternativă de OC0, dar direcția *nu* este setată automat).

Exemplu de calcul PWM – Timer 0 – pinul OC0:

La PWM → ***frecvenţa constantă, factorul de umplere variabil***; frecvența/perioada PWM o vom seta la început și va rămîne constantă, variind pe parcurs doar T_1 și implicit factorul de umplere. La orice factor de umplere diferit de 0 sau 100%, frecvența PWM se va regăsi în semnalul de ieșire, de formă dreptunghiulară (vezi figura 11), deci trebuie ca elementul de acționare să nu lase să treacă această frecvență (să aibă o natură de FTJ cu frecvența de tăiere mai mică decît frecvența PWM). De exemplu, pentru varierea luminii unui LED sau tub cu descărcare, ochiul uman are o frecvență de tăiere (dincolo de care nu mai percepe „pîlpîitul”) de ordinul a 70..100 Hz, în funcție de individ. Dacă se folosește un bec cu incandescență în loc de LED, acesta introduce propria frecvență de tăiere semnificativ mai mică (net sub 50Hz) datorită inerției termice a filamentului.

Etape:

- Exemplu: alegem frecvenţa pentru fcuarţ = 13.5MHz și Timer0 de 8 biți; alegem modul „Fast PWM”.
- alegerea factorului de divizare:
- presupunem prescaler de 1024
- “umplerea” registrului de timer de 8 biţi: 256
- obţinem f_{PWM} = 13500000/1024/256 = 51 Hz
- OBS: 51Hz poate fi acceptabil la becuri/motoare, datorită inerției, dar în cazul LED-urilor această frecvenţă este vizibilă ca un uşor “flicker”; prin urmare valoarea nu ne convine.

- alegem așadar un prescaler mai mic: 256, implicit fPWM va fi mai mare
- f_{PWM} = 13500000/256/256 = 205 Hz

- deci folosim prescaler de 256 → CS02:00 = 100 cf. tabelului din datasheet

Valorile corespunzătoare ale registrelor vor fi ca în figura 13:

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)

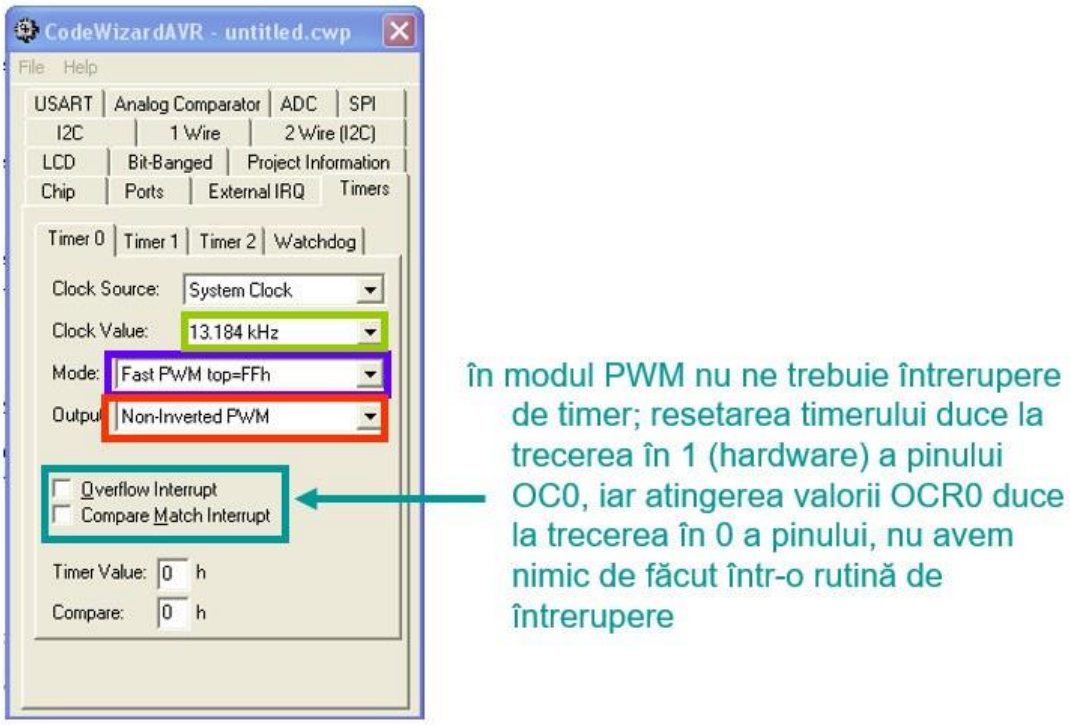
CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin.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

Fig. 13 Timerul în mod PWM

Rezultă de mai sus valorile WGM 01:00 = **11** COM 01:00 = 10 CS 02:00 = 100

în total, TCCR0 = 0**1**10**1**100 = 6Ch

Putem folosi CodeWizard pentru setarea valorilor registrului; culorile corespund grupurilor de biți de mai sus. În concluzie, odată inițializat TCCR0, tot ce trebuie să facem pentru a varia PWM în timp real este să modificăm registrul OCR0.



Un exemplu de program care folosește valorile de mai sus și variază intensitatea luminoasă a unui LED conectat la pinul OC0, în 4 pași, cu pauze de 1 secundă:

```
// 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 pași
// 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
 }
}
```