

# Descrierea părții comune a proiectului; utilizarea resurselor procesorului AVR

M. Stanciu      februarie 2017 - rev. 7

## Cuprins

1. Procesorul și sursa de alimentare .....	1
2. Portul serial .....	4
2.1. Vizualizarea formelor de undă; depanarea portului serial .....	6
2.3 Folosirea unui convertor RS232-USB .....	6
2.4 Folosirea unui convertor TTL-USB; alimentarea (opțională) din USB.....	7
3. Pini ai procesorului de I/O de uz general .....	8
4. Convertorul Analog-Digital (ADC) .....	10
5. Senzori .....	14
6. Timere .....	15

## 1. Procesorul și sursa de alimentare

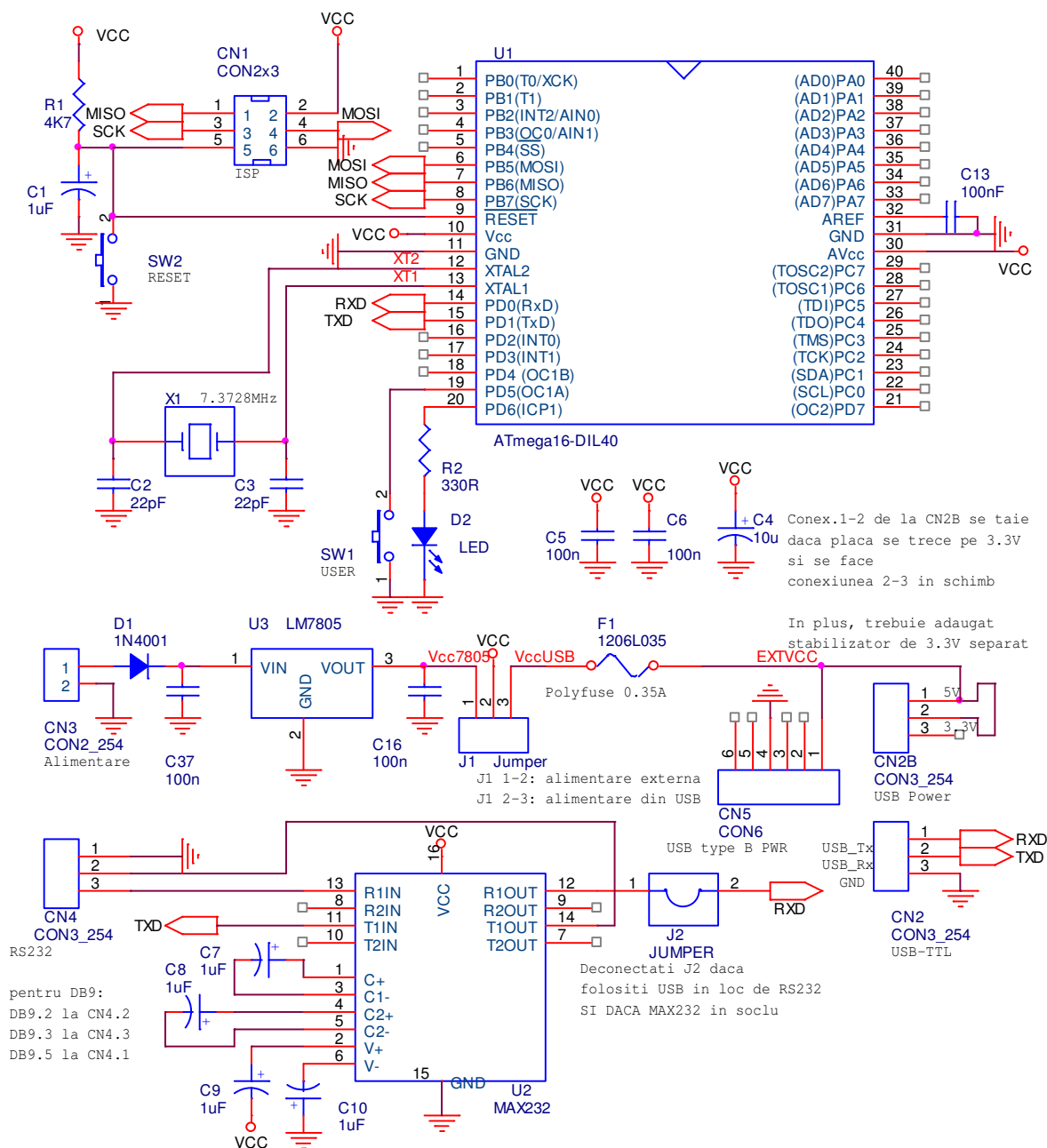


Fig. 1 Schema machetei

Schema din fig. 1 cuprinde componentele comune, care se vor regăsi pe toate machetele de proiect, indiferent de teme; la ele se vor adăuga conectorii și componentele specifice. R1, C1 formează circuitul de “power-on reset”. În momentul aplicării Vcc, C1 este descărcat și “trage” linia RESET\* în “0”, resetând procesorul. Ulterior, condensatorul se va încărca prin R1 și linia va sta în “1” (procesul este asimptotic, dar sfârșitul încărcării se consideră după un timp aproximativ egal cu  $5R_1C_1$ ). Acest reset este necesar pentru a asigura pornirea în bune condiții a procesorului; în lipsa lui, tensiunile tranzitorii care apar în momentul alimentării pot duce la ajungerea procesorului într-o stare incertă. Practic, linia RESET\* e ținută în “0” un timp semnificativ mai lung decât are nevoie sursa de alimentare să intre în regim staționar.

Condensatoarele C5 și C6 se află cât mai aproape de procesor, între pinii 10-11, respectiv 30-31. Ele sînt condensatoare de decuplare și sînt specifice alimentării oricărui circuit digital - asigură o “rezervă” de energie în momentul comutării, și astfel previne apariția zgomotului de comutare pe liniile de alimentare. Condensatorul electrolitic C4 nu este obligatoriu, dar este recomandat pentru reducerea riscului ca procesorul să 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 cuarț, împreună cu condensatoarele C2, C3 și cu amplificatorul intern de la bornele XTAL1,2 formează un oscilator cu cuarț. Aceste componente trebuie să se afle, de asemenea, cât mai aproape de pinii respectivi ai procesorului.

Pentru alimentare se folosește stabilizatorul U3 cu 3 terminale (7805), 2 condensatoare de 100nF necesare pentru stabilitatea funcționării acestuia, și 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 poziționată astfel încît se aprinde cînd PD6 este pe “1” logic; rezistența R2 asigură limitarea curentului prin LED la cca.  $(5-1.6)/330=10\text{mA}$ . De notat că în multe cazuri cînd se conectează un LED la ieșirea unui circuit digital, fără tranzistor de comanda, el se montează invers (între pinul circuitului și Vcc, evident cu anodul la Vcc), întrucît majoritatea circuitelor suportă un curent mai mare cînd pinul este în “0” logic decât atunci cînd e în “1”. Dezavantajul e ca LED-ul se aprinde pe “0”. În cazul nostru însă, procesorul AVR suportă curenti egali pe “0” și pe “1” logic.

Butonul SW1 leagă PD5 la masa (0 logic) în momentul apăsării. Întrucît, atunci cînd nu este apăsat, starea pinului PD5 nu este definită, va trebui activată o rezistență 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 că există mai multe standarde de conector (în principal 2x3 pini și 2x5 pini), noi îl vom folosi numai pe acesta pentru compatibilitate. Deoarece procesorul pe care-l primiți este pre-programat de către mine 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, în locul convertorului TTL-RS232 (MAX232).

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 liber, 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 locuri cu paduri spațiate la 50 mils (1,28mm) pentru eventuale circuite integrate în capsulă SO/.050, respectiv (pe spate) paduri pentru un circuit integrat cu terminale spațiate la 1mm.

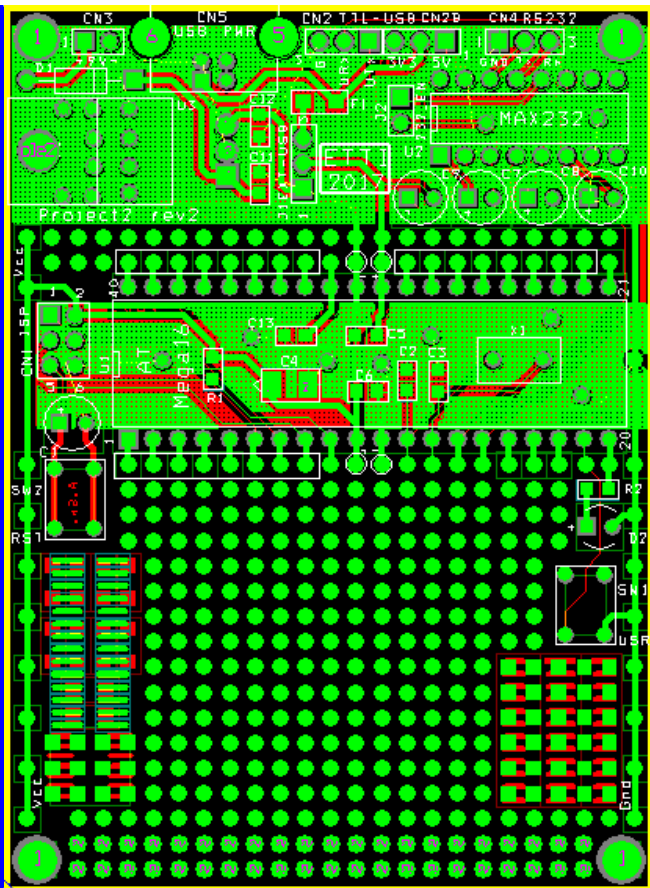
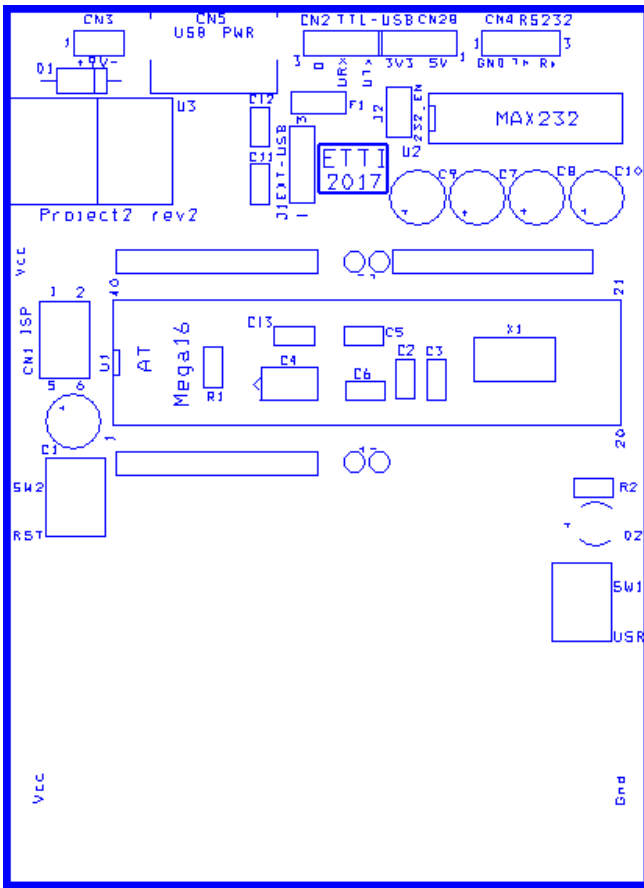
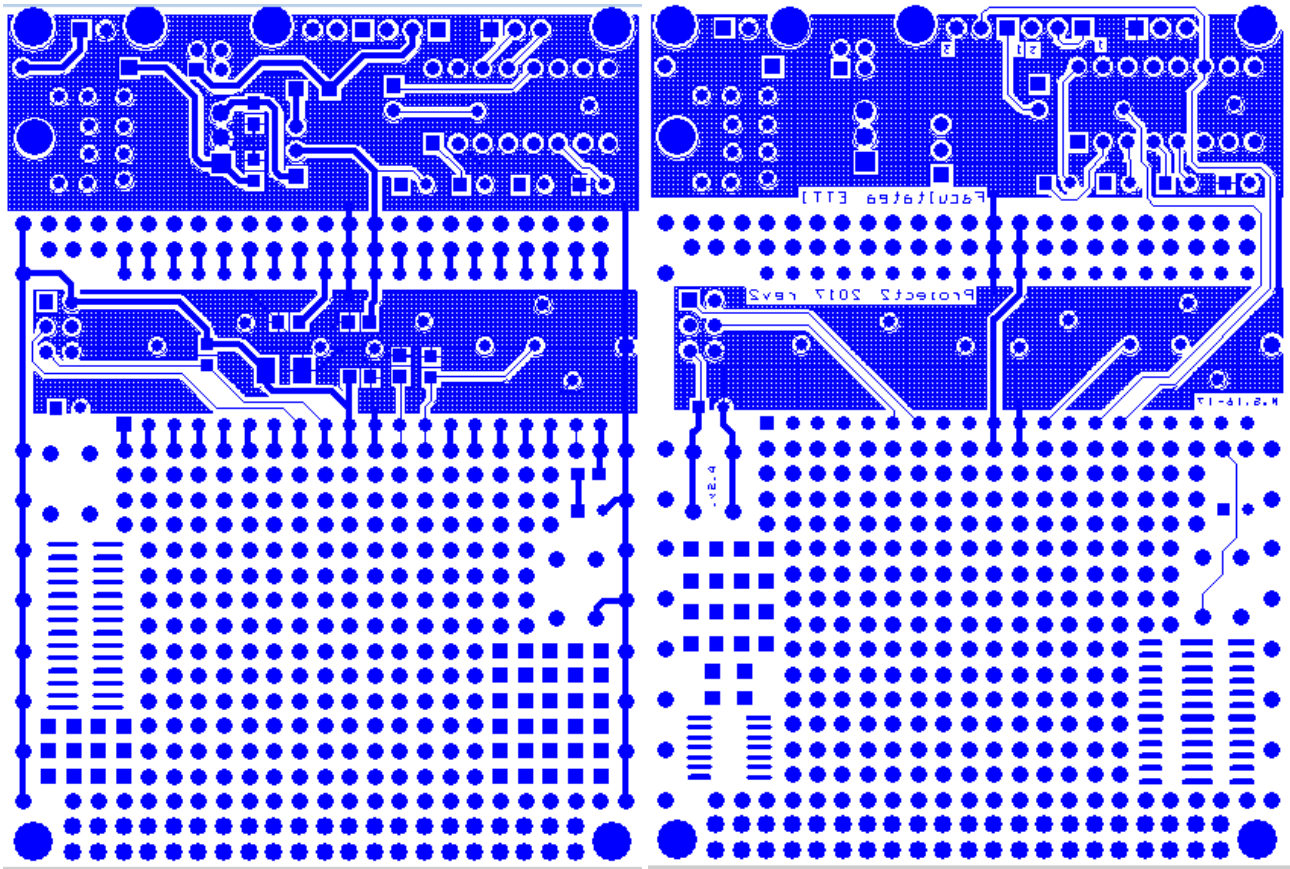


Fig. 2 PCB: Top, Bottom, Silkscreen Top și vedere generală

## Alimentarea la 3.3V

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 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 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 adaptorului USB-TTL, se *taie* cu un cutter traseul de pe PCB dintre pinii CN2B 1-2 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 adaptorul USB-TTL este teoretic posibilă, dar nu este recomandabilă; așa cum se vede pe schema adaptorului din fig. 5, 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 adaptorul 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. Portul serial

Microcontrollerele AVR includ 1 sau 2 porturi seriale UART de nivel TTL. 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 și respectiv RS232 pentru octetul "00110101".

Cei 8 biți de date sînt încadrați de un bit de start, care va fi întotdeauna "0", și 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 ceilalți biți, dar aceasta se întîmplă pentru că, în absența datelor, linia este ținută în "1" logic, deci după ce se transmite acest bit "1" starea liniei nu se schimbă, pînă la transmiterea unui nou octet, care va începe cu un nou bit de start "0".

**Atenție!** Folosind notația binară obișnuită b7b6b5b4b3b2b1b0, unde b7 este MSB și b0 este LSB, biții pe serială se transmit *LSB-first*, nu *MSB-first* cum ar fi mai intuitiv. Așadar, după bitul de start (0, stînga) urmează LSB (b0), iar MSB (b7) este adiacent bitului de stop (1, dreapta)

Biții de stop și de start garantează faptul că la începutul fiecărui octet are loc o tranziție "1 → 0", chiar și în cazul în care octeții transmiși reprezintă lungi șiruri de "1" sau "0". Dezavantajul este că la fiecare 8 biți de date trebuie adăugați cei 2 biți, adică eficiența transmisiei este de doar 80%.

Pinii RxD, TxD ai procesorului se conectează la pinii cu același nume ai convertorului de nivel realizat cu circuitul MAX202, MAX232 sau echivalent. Rolul acestuia este să convertească nivelurile TTL (0..5V) în niveluri RS232, și viceversa; practic, tensiunile +10V și -10V se obțin prin dublarea tensiunii de alimentare a circuitului, folosind un convertor intern cu capacități comutate, care folosește condensatoarele externe C7-C10.

Funcționarea pe scurt este următoarea: în interiorul circuitului MAX232 există un grup de comutatoare și o logică de comandă, prin care condensatoarele se încarcă mai întîi la 5V și apoi se pun în serie. Astfel se obține tensiunea de 10V. Prin conectarea inversă se obține -10V (bornele la care sînt legate C7, C8 sînt "flotante" deci comutatoarele interne le pot conecta în orice sens). Schema este dată în figura 3.

**Observație:** condensatoarele de 1uF pot fi înlocuite cu 100nF în cazul în care se folosește MAX202 sau ST232. În plus, condensatoarele de 100nF sînt nepolarizate, în timp ce la cele de 1uF trebuie avut grijă la polaritate.

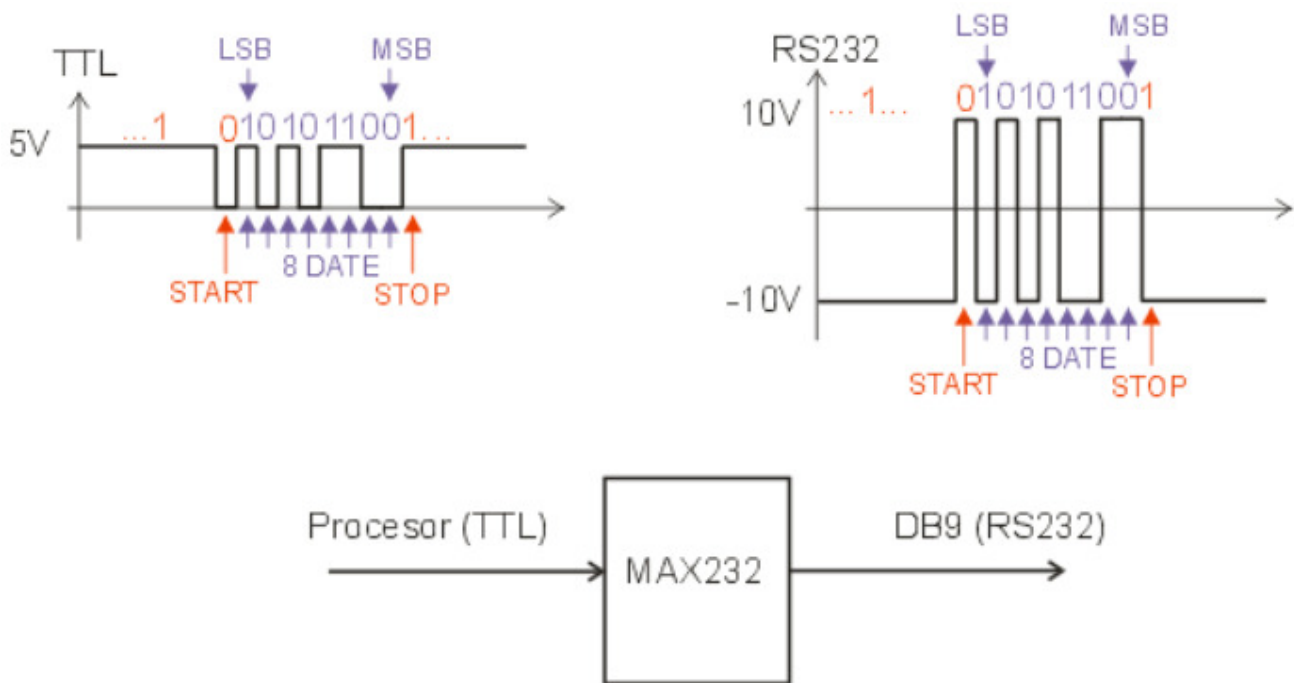


Fig. 3 Forme de undă la intrarea (TTL) și la ieșirea (RS232) convertorului MAX232 pentru octetul de date 00110101. Formele de undă TTL sînt disponibile și pe intrarea convertorului TTL-USB.

**Verificarea funcționării acestui circuit în regim static** (fără a transmite date) se face măsurînd cu voltmetrul tensiunile de pe pinii 2 și 6, față de masă. Trebuie să se obțină o tensiune de cca. +8..+10V pe pinul 2 și -8..-10V pe pinul 6. Practic, dublarea tensiunii la valoarea de 10V se obtine numai în cazul ideal și anume în gol; în sarcină această tensiune este mai mică, dar suficientă pentru a respecta standardul. Pe pinii de date, în pauzele dintre transmisii, trebuie să se măsoare “1”, adică +5V pe TTL și circa -10V pe RS232 (logică inversată).

Mufa DB9 de tip mamă este cablată după standardul DCE (*Data Communications Equipment*):

Pin mufa DB9 DCE	Descriere
2	Tx (transmisie; sens de ieșire, de la machetă către PC)
3	Rx (recepție; sens de intrare în machetă)
5	GND

Aceasta înseamnă ca se va folosi un cablu serial direct (1-la-1) pentru conectarea la PC, care este echipat cu o mufa de tip tată, cablată după standardul DTE (*Data Terminal Equipment*), la care pinii 2 și 3 sînt inversați. De notat ca dacă s-ar dori conectarea a 2 echipamente cu același standard (DCE cu DCE sau DTE cu DTE; de exemplu, 2 PC-uri legate între ele pe serială), ar trebui un cablu de tip “crossover” (pinii 2 și 3 inversați la unul din capete). De obicei, toate cablurile disponibile comercial cu mufe mamă-tată sînt de tip direct, iar cele mamă-mamă sînt de tip crossover.

Observație: denumirile DTE/DCE se folosesc și pentru alte standarde de comunicații, de exemplu în cazul Ethernet, un PC este DTE și un switch este DCE; așadar, similar cu cazul de mai sus, conectarea unui PC la switch se face cu cablu direct, iar conectarea PC-PC se face cu cablu *crossover*.

## 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 biți deci durează:

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

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

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

Atenție la reglajul de sincronizare! Sincronizarea trebuie setată diferit în funcție de punctul unde se măsoară. Bitul de start este ca în fig. 3:

- un front negativ de la 5V la 0V pe nivele TTL (deci *slope*= *falling*)
- un front pozitiv de la -10V la +10V pentru RS232 (deci *slope*= *rising*)

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

Se urmărește 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 pe DB9.2 (nivel RS232), apoi la intrarea în MAX232 (pin 13),
3. este convertit de MAX232 în nivel TTL și scos la ieșire (pin 12),
4. ajunge la procesor (uP pin 14),
5. acesta trimite caracterul cu codul imediat următor (uP pin 15), ajunge la intrarea MAX232 (pin 11, nivel TTL),
6. este convertit în nivel RS232 (MAX232 pin 14) și
7. ajunge pe DB9.3 către PC.

*Obs: în cazul folosirii convertorului USB-TTL, se pot verifica doar pașii 4,5.*

În cazul în care nu funcționează comunicația machetă-PC (nu se primește înapoi în terminal caracterul următor), depanarea se face cu osciloscopul, determinînd în care dintre punctele de mai sus nu se mai observă forma de undă (se întrerupe "lanțul").

Cele mai frecvente defecte:

- cipul MAX232 nu funcționează; verificați prezența tensiunilor de aprox. +8..10V și -8..-10V pe pinii 2, respectiv 6, în caz contrar probabil că cel puțin unul din condensatorii electrolitici este lipit pe dos.
- softul de test nu rulează pe machetă, din cauza unui defect la altceva decît portul serial (faptul că softul rulează se vede prin clipirea LED-ului);
- este alt soft programat în procesor
- problemă de setare: viteza nu e setată la 9600, N, 8, 1, sau terminalul de la PC setat pe COM1 și cablul de la machetă înfipt în COM2, atunci cînd PC-ul are 2 porturi seriale COM. În CodeVision, se poate alege COM-ul și parametrii din Settings -> Terminal
- dacă nu se vede în terminal decît caracterul recepționat, nu și cel transmis (se apasă „a” și pe ecran apare doar „b” în loc de „ab”) înseamnă că în Settings -> Terminal s-a omis bifarea opțiunii „local echo”.

## 2.3 Folosirea unui convertor RS232-USB

În cazul în care PC-ul nu dispune de un port serial, se pot folosi convertoare RS232-USB, care au la un capăt mufă DB9 tată (precum PC-ul) și la celălalt USB. Prin instalarea driverului acestui convertor pe PC, apare un port COM virtual care poate fi folosit în terminal sau programul PC-loader ca și cînd ar fi un port RS232

nativ. Atenție la numărul COM-ului, 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.

## 2.4 Folosirea unui convertor TTL-USB; alimentarea (opțională) din USB

Varianta convertorului RS232-USB nu este economică, căci practic pe lanțul uP – PC avem conversii TTL – RS232 urmat de RS232-USB și înapoi; mai simplu este să se facă conversie direct TTL-USB.

În acest caz **cipul MAX232 nu se montează în soclu**<sup>1</sup>, sau se scoate jumperul J2, și în schimb 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 și folosit portul COM virtual.



Fig. 4 Converteare 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. Selecția se poate face în două moduri:

- dacă 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.
- dacă se dorește și alimentarea din USB (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, pe PCB este deja făcută conexiunea între „Vcc” și „5V” (pinii 1,2 ai CN2B), în locul jumperului de pe convertor.

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

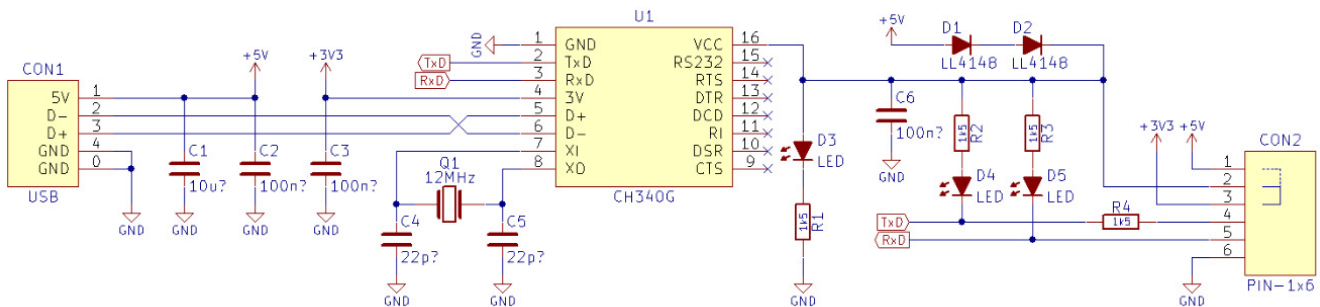


Fig. 5 Schema convertorului USB-TTL cu CH340

Se observă că reducerea tensiunii de la 5V la 3.3V se face prin inserierea 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 (cînd Vcc=3.3V), fie se lasă neconectat (cînd Vcc=5V).

<sup>1</sup> de ce trebuie scos MAX232 în acest caz ? pentru că nu putem avea simultan conectate la pinul RxD al uP două surse de semnal: pinul de la MAX232 și pinul de la convertorul USB-TTL. Este posibil doar invers, ca o sursă de semnal să fie conectată la 2 receptoare.



**Observație 1:** dacă dorim alimentarea plăcuței din USB (la 5V), *trebuie mutat jumperul J1* de pe fig. 1 în poziția 2-3, pentru a nu mai avea stabilizatorul 7805 în circuit, întrucât un stabilizator nu este proiectat să primească tensiune din exterior pe pinul 3 (de ieșire din stabilizator) și se poate defecta.

**Observație 2:** 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, fie convertorul USB-TTL, fie MAX232.

În cazul folosirii acestui convertor, formele de undă se vor urmări numai pe nivelele 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 adaptorul USB-TTL la PC (fără a-l conecta și la PCB-ul nostru), se verifică în Windows → Device Manager 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
- se apasă o tastă oarecare în terminal; LED-ul de Rx de pe adaptorul USB-TTL trebuie să clipească, indicînd faptul că acesta primește caracterele de la terminal
- se scoate jumperul de pe adaptor și se pune direct între pinii Rx, Tx, a.î. caracterele primite (sub formă 232 nivel TTL) 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.
- dacă pașii de mai sus funcționează, înseamnă că adaptorul USB-TTL este instalat și funcționează corect dpdv hard+driver; orice problemă mai departe este de la conexiunile către PCB sau pe PCB-ul însuși. Caracterele sub formă TTL pot fi vizualizate cu osciloscopul direct pe pinii adaptorului.

### 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 analog-digital (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:

- Inicializare cu `DDRX.Y = 0` (Data Direction Register)

- Citește valoarea pinului X.Y folosind `PINX.Y`

Exemplu:

```
if(PIND.5 == 0)      // citește 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 rezistor intern, switch legat la masa
           // implicit PORTX.Y = 0 (fără pull-ul rezistor)
```

### 2) Pini de ieșire:

- Inițializare 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 cînd 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 aritmetice 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 |= 0b11100000;
```

**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 citați
                                         // 0, primii 5 se vor păstra
```

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

```
DDRB &= 0b111101111; // B.4 = 0
DDRB |= 0b00001000; // 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 &= 0b00001111; // ne asigurăm că biții 7,6,5,4 ai lui V sînt 0, dacă
// nu știm sigur acest lucru
PORTB &= 0b11110000; // 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 prioritate 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 (obținut 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:

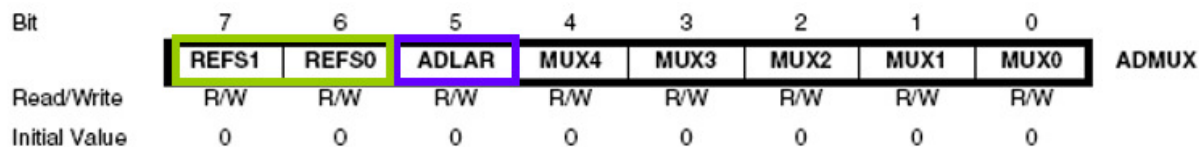
- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13 - 260 μs Conversion Time
- Up to 15 kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- 7 Differential Input Channels
- 2 Differential Input Channels with Optional Gain of 10x and 200x<sup>(1)</sup>
- Optional Left adjustment for ADC Result Readout
- 0 - V<sub>CC</sub> 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

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

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

unde prin [ ] am notat operația de trunchiere.

Registrele de control ale ADC sînt ADMUX și ADCSRA, descrise mai jos:



REFS1	REFS0	Voltage Reference Selection	MUX4..0	Single Ended Input
0	0	AREF, Internal Vref turned off	00000	ADC0
0	1	AVCC with external capacitor at AREF pin	00001	ADC1
1	0	Reserved	00010	ADC2
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin	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 putându-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	<i>b9b8</i>	<i>b7b6b5b4b3b2b1b0</i>
ADLAR = 1	<i>b9b8b7b6b5b4b3b2</i>	<i>b1b0</i>

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, ducînd la o rezoluție:

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

**Observație:** atunci cînd MUX3, MUX4 au alte valori în loc de 00, este posibilă măsurarea diferențială, adică diferența dintre tensiunile de pe două canale ADC, în loc de cazul *single-ended* cînd 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	Semnificație în cazul unipolar	Semnificație în cazul bipolar (complement față de 2)
000000000	0	0
000000001	1	1
...	...	...
011111110	510	510

011111111	511	511
100000000	512	-512
100000001	513	-511
...	...	...
111111110	1022	-2
111111111	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  $N=2^n$

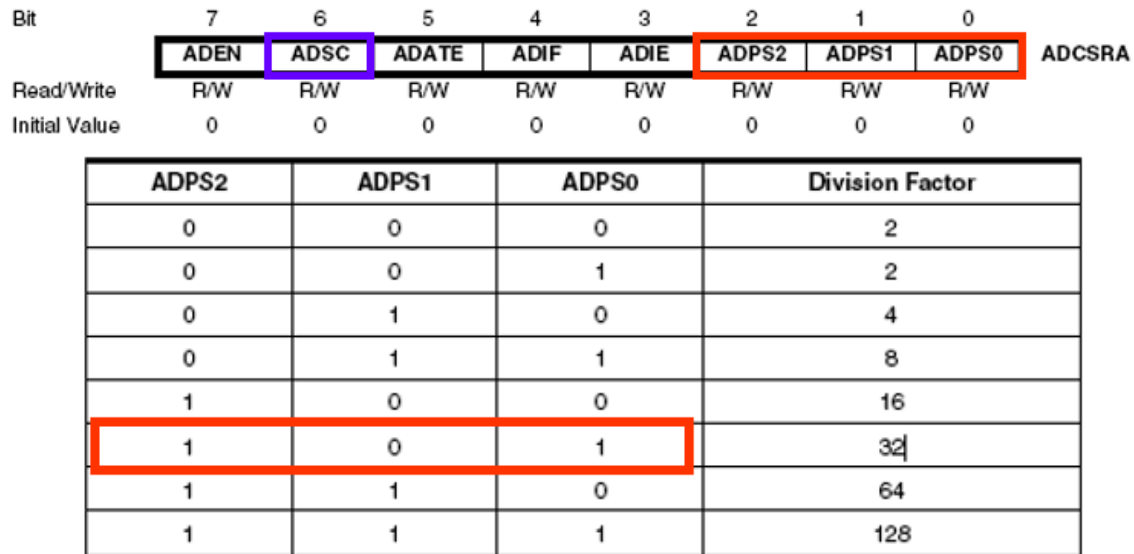


Fig. 7 setarea registrului ADCSRA al ADC

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 atâta timp cât 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 sfârșitul unei conversii determină începutul următoarei conversii.
- ADIF = ADC Interrupt Flag; devine 1 când o conversie e gata și 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 cât se dividează 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 decât 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) decât 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, obținând factorul  $K < 1$ :

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

astfel încât 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:

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

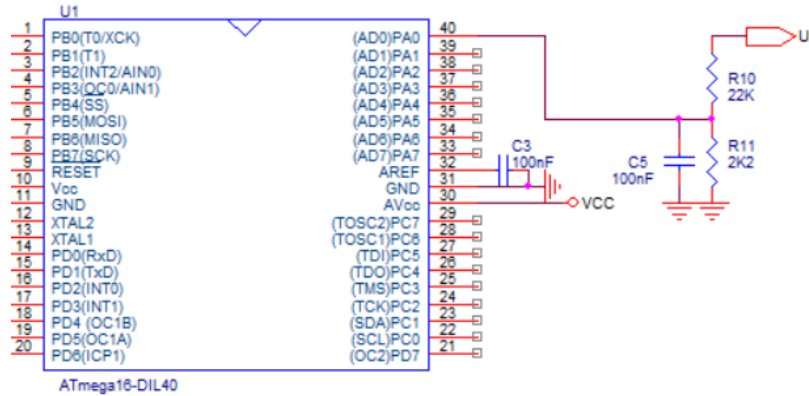


Fig. 8 Modul de conectare al unui canal ADC cu divizor 1/11

Rezultă deci relația de legătură dintre  $N$  și  $U_x$ :

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

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 pînă cînd bitul ADSC devine 0. Pentru conversii dese sau continue, pentru a nu ocupa procesorul, se recomandă lucrul pe întreruperi, citirea registrului ADCH sau ADCW făcîndu-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=0b10010**101**;

#define ADMUX\_NOCHANNEL 0b**11**000000

*// 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 rezultat; 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 ieșire digitală (TTL):

- stări: LO și 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 și 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 și 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 ieșire 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 :

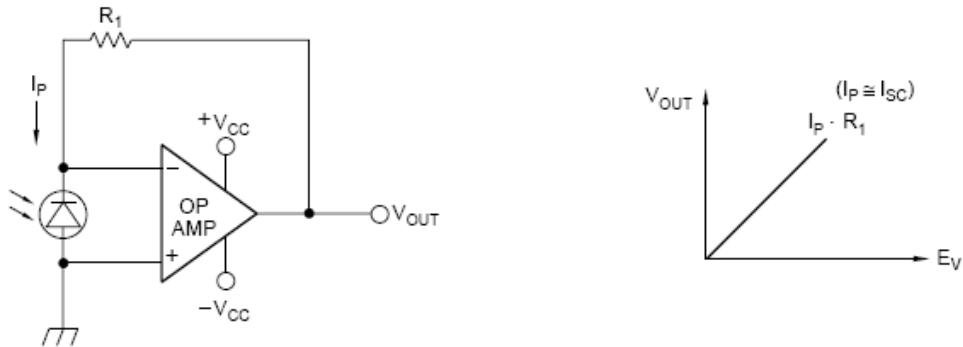


Fig. 9 Folosirea unei fotodiode ca senzor de lumină

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 obișnuită, 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 măsura intervalului 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 biți sau 16 biți. 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, decât 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  $f_{\text{CLOCK}}$  (frecvențe 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 și funcția `delay_ms(valoare)` dar ține procesorul blocat (nu lucrează pe întreruperi).

### Aplicație 1: folosirea Timer1 pentru generarea unui eveniment periodic

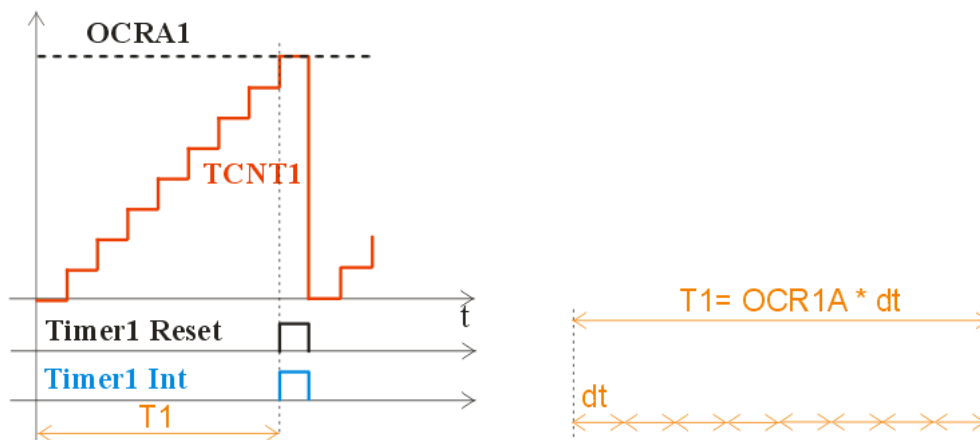


Fig. 10 Timerul 1 în modul CTC

Vom folosi modul CTC (Clear Timer on Compare Match) care înseamnă că valoarea din registrul de numărate TCNT1 crește pînă 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.
- cînd  $TCNT1 = OCRA1$  (Output Compare Register), se generează o întrerupere și se resetează timerul, după care incrementarea continuă de la 0
- alegînd OCRA1 și frecvența 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 și  $dt$  mare



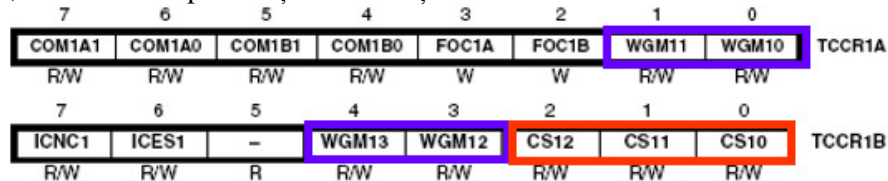
- N mare: registru de 16b, nu 8b; verificați în datasheet care timer e pe 16b !
- dt mare: prescaler cât mai mare deci frecvență cât mai mică.

### 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:



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 (frecvență mică)

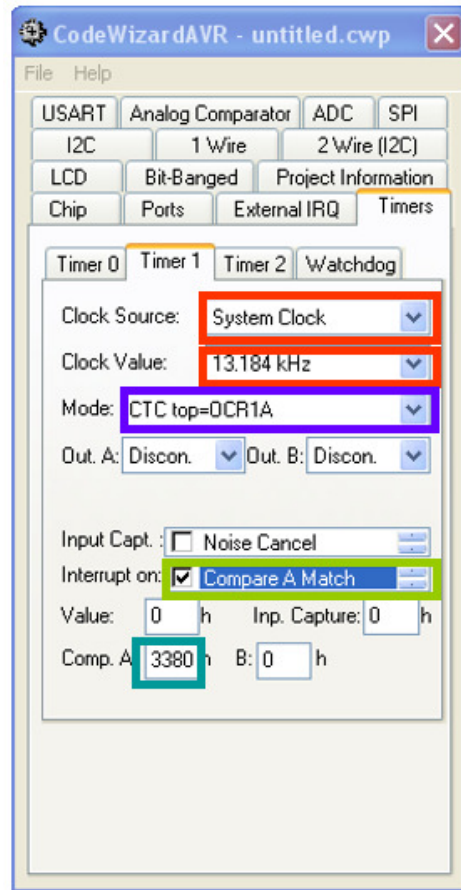
- presupunem că dispunem de un cuarț de 13.5MHz

- $f_{\text{cuart}} = 13.5\text{MHz}$  → trebuie divizare cu  $13500000 > 65536$  (16 biți) → nu se poate direct  
→ trebuie prescaler (în general, cu cât ceasul e mai rapid și evenimentul dorit e mai lent, cu atât 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**);  $13.5\text{MHz} / 1024 = 13184\text{Hz}$
- am ajuns la 13184Hz, dorim 1Hz: mai divizăm cu  $13184 = 3380h$   
→ OCR1AH = 33h, OCR1AL = 80h

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

- TCCR1A = 00000000 și
- 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 cât 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

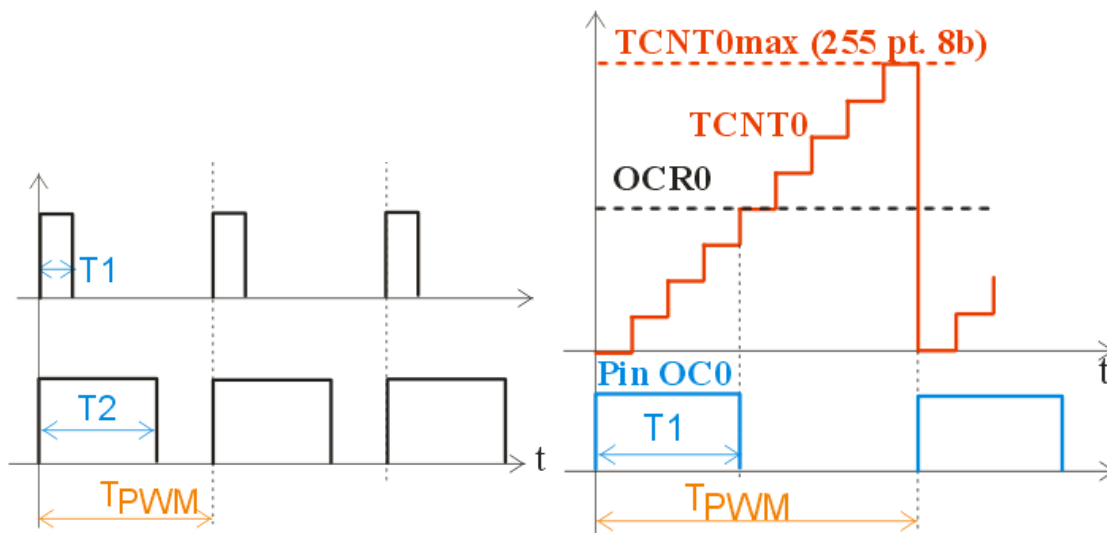


Fig. 12 Modul PWM al timerelor

După cum se știe, PWM (*Pulse Width Modulation*) înseamnă un semnal cu perioada constantă ( $T_{PWM}$  pe figura 11, 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 cât factorul de umplere este mai mare, cu atât *media* semnalului de la ieșire este mai mare, deci, de exemplu LED-ul luminează mai tare.

Observație: un factor de umplere de  $\frac{1}{2}$  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  $\frac{1}{2}$  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  $\eta_2 = T_2/T_{PWM}$  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 (OCO pentru Timer0, etc) să urmărească valoarea  $T_1$  ca pe figura 12:

- cât timp  $t < T_1$ , OC0=1 deci LED aprins
- când  $T_1 < t < T_{PWM}$ , OC0=0 deci LED stins
- apoi, când se atinge sfârșitul 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 ATmega16, OC0 (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).

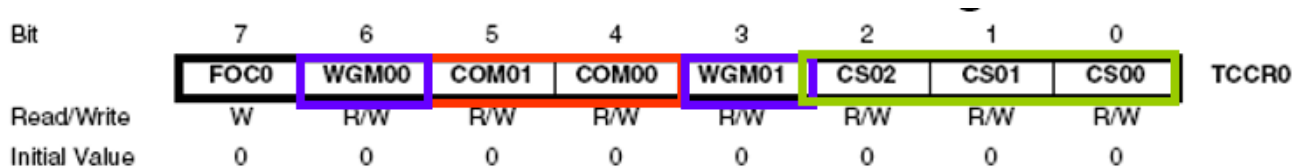
Exemplu de calcul PWM – Timer 0 – pinul OC0:

La PWM → **frecvența constantă, factorul de umplere variabil**; frecvența/perioada PWM o vom seta la început și va rămîne constantă, variînd 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 frecvența pentru fcuart = 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 biți: 256
- obținem  $f_{PWM} = 13500000/1024/256 = 51 \text{ Hz}$
- OBS: 51Hz poate fi acceptabil la becuri/motoare, datorită inerției, dar în cazul LED-urilor această frecvență este vizibilă ca un ușor “flicker”; prin urmare valoarea nu ne convine.
- alegem așadar un prescaler mai mic: 256, implicit  $f_{PWM}$  va fi mai mare
- $f_{PWM} = 13500000/256/256 = 205 \text{ Hz}$
- deci folosim prescaler de 256 → CS02:00 = 100 cf. tabelului din datasheet

Valorile corespunzătoare ale registrelor vor fi ca în figura 13:



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)

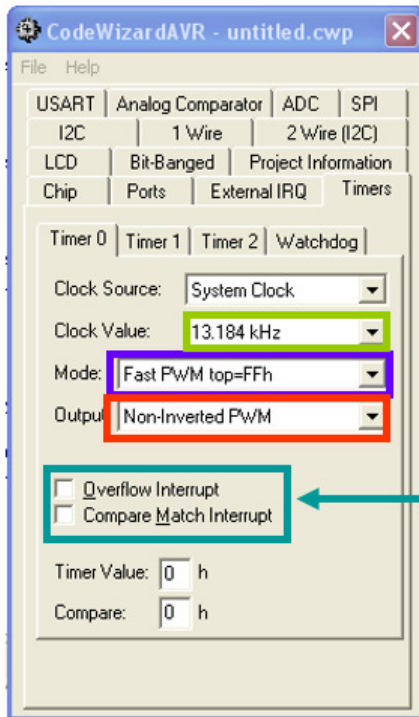
CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk <sub>IO</sub> /(No prescaling)
0	1	0	clk <sub>IO</sub> /8 (From prescaler)
0	1	1	clk <sub>IO</sub> /64 (From prescaler)
1	0	0	clk <sub>IO</sub> /256 (From prescaler)
1	0	1	clk <sub>IO</sub> /1024 (From prescaler)
1	1	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 = 01101100 = 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.



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

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:

```
// initializam 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 variaza intensitatea luminoasa a unui LED conectat la pinul
// OC0 in 4 pasi
// intensitatea se schimba la cite o secunda, schimbind 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
    }
}
```