Ionel Daniel MORARIU

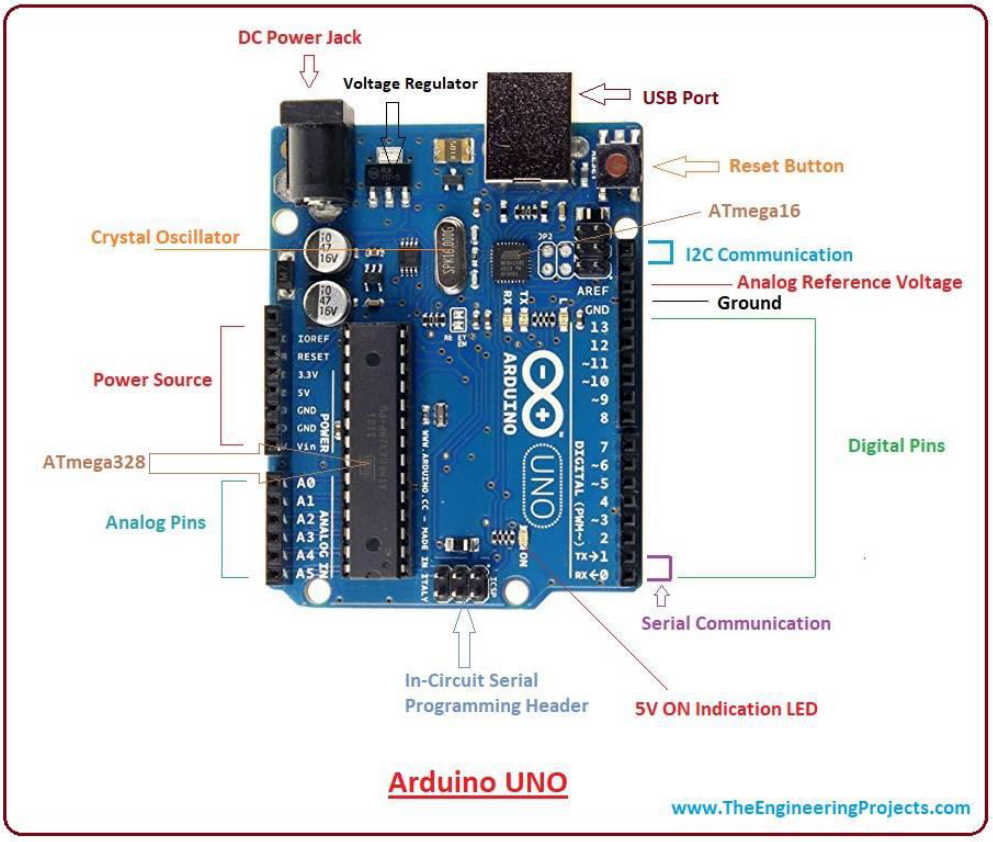
Sisteme de Intrare Ieșire și Echipamente periferice

Îndrumar de laborator

Cuprins

Arduino. Portul Paralel

1.1 Caracteristici generale



Arduino UNO este o placă de dezvoltare bazată pe un microcontroller ATmega 328P pe 8 biți la 16MHz cu 32kB memorie flash (EEPROM) și 2Ko memorie SRAM.

Placa de dezvoltare conține:

- 14 pini digitali de intrare/ieșire D0...D13 (6 ieșiri pot fi programate cu PWM - Pulse With Modulation, pinii 3,5,6,9,10,11)
- 6 pini analogici cu o rezoluție de 10 biți (A0 ... A5), măsoară tensiuni între 0 – 5V în funcție de pinul AREF
- Rx/TX - o linie seriala cu nivele logice TTL
- 2 pini pentru magistrala I^2C
- 2 timere pe 8 biți
- 1 timer pe 16 biți
- Un conector USB pentru conectarea la calculator pentru transferul codului în controler utilizând o aplicație specializată

1.2 Regiștri Arduino

Microcontrolerul ATmega pune la dispoziția programatorului 3 porturi paralele care permit manipularea rapidă și low-level a pinilor de I/O ai microcontrolerului. Porturile paralele pot fi configurate software pentru a fi de intrare sau de ieșire (ele din start sunt configurate ca fiind de intrare). Microcontrolerul ATmega 328P pune la dispoziție 3 astfel de porturi numite:

- Port B (pinii digitali ai plăcii de dezvoltare de la 8 la 13)
- Port C (pinii analogici ai plăcii de dezvoltare de la A0 la A5)
- Port D (pinii digitali ai plăcii de dezvoltare de la 0 la 7)

Fiecare port paralel este controlat din microcontroler prin intermediul a 3 registre de configurare / date:

- Registrul **DDRx** (Data Direction Register) – este registrul de configurare și determină dacă pinul corespunzător din portul paralel este de INPUT (primește date) sau OUTPUT (transmite date). Fiecare bit din acest registru controlează un pin din portul respectiv. ”x„ semnifică numele portului iar pentru a configura pinul pentru a fi de intrare se pune valoare 0 iar pentru ieșire se pune valoarea 1;
- **PORTx** (Data Register) – registru de date care permite citire/ scriere informațiilor pe portul respectiv;
- **PINx** (Read Register) - registru de date folosit pentru a citi un pin configurat de intrare..

Fiecare bit din acești registri corespunde unui singur pin. În figurile următoare este prezentat cum sunt așezații biții pentru fiecare pin corespunzător. Anumiți pini din acești regiștii pot îndeplinii și alte funcții. În moment în care se activează cealaltă funcționalitate a pinului respectiva, acesta nu mai poate fi utilizat ca și pin de comunicație paralelă.

În continuarea specificăm funcțiile pentru porturile B, C și D pentru placa de dezvoltare folosită.

1.2.1 PORTB

Este un port de comunicație paralelă care permite scrierea/ citirea informaților pe 8 biți. Pentru a specifica individual direcția fiecărui biți (de intrare/ieșire) trebuie scris registrul DDRB. Pentru placa de dezvoltare folosită din PORTB se utilizează doar 6 cei mai puțin semnificativ biți și sunt legați la pinii plăcii astfel:

-	-	Pin 13	Pin 12	Pin 11	Pin 10	Pin 9	Pin 8

1.2.2 PORTD

Este un port de comunicație paralelă care permite scrierea/ citirea informaților pe 8 biți. Pentru a specifica direcția fiecărui pin trebuie scris registrul DDRD. Pinii specificați în portul D sunt corespunzători pentru placa de dezvoltare folosită, altfel PORTD este un registru pe 8 biți de intrare/ieșire.

Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0

1.2.3 PORTC

Port de comunicație paralelă pe 8 biți. în care se găsesc pinii analogici de intrare. Acești pini sunt doar de intrare

-	-	A5	A4	A3	A2	A1	A0

Informații suplimentare se găsesc în documentația ATmega secțiunea 18, pagina 122-130.

1.3 Interfața Arduino IDE(Integrated Development Enviroment)

Permite dezvoltarea de aplicații pentru arduino, compilarea acestora și încărcarea în memoria Flash a controlerului. Memoria Flash se scrie în condiții speciale (comutată în starea de scriere) și datele sunt păstrarea în memorie și atunci când se întrerupe alimentarea modulului. Aplicația se va scrie și se va compila pe calculator (pe o platforma), iar rularea se va face de pe controler (pe altă platformă) de aceea după fiecare compilare noua aplicație trebuie înscrisă în controler pentru a rula.

În controler la un moment dat rulează doar o aplicație care trebuie să conțină o buclă care să nu se termine niciodată deoarece o dată terminată aplicația, controlerul nu mai are ce executa (nu exista sistem de operare). Revenirea din această stare se poate face doar după operația de reset. De aceea orice aplicație pe controler trebuie să conțină o funcție de inițializare care se execută o singură dată la pornire și o funcție care se va executa la infinit. Funcția main a unei aplicații pe controler arată astfel:

```
void setup()
{
      //configurare regiștrii controler, se executa o singură dată
```

```
}
int main()
{
	setup();
	while (1)
	{
		//codul care se dorește executat
		// sau apelul funcției loop ()
	}
}
```

În cadrul interfeței Arduino IDE este folosit limbajul C pentru scrierea codului aplicației, iar acesta se poate scrie codul în două moduri:

- scriind complet codul cum este cel prezentat mai sus inclusiv funcția main()
- folosind funcțiile create automat de interfață și anume: funcția de configurare numită setup() și funcția care se execută în buclă numită loop(). În acest caz utilizatorul nu mai vede funcția main iar codul din cadrul funcției loop() este apelat in cadrul buclei la infinit astfel că aceasta nu trebuie scrisă și în interiorul acesteia.

Pentru compilarea unei aplicații se poate accesa din meniu intrarea Sketch-> Verify/Compile. Dacă compilarea a fost cu succes se va crea automat un fișier cod pentru arduino care poate fi încărcat în controler folosind comanda Sketch->Upload.

1.4 Probleme propuse

Problema 1. Pe placa de extensie de la arduino există 4 LED-uri conectate pe pinii 10, 11, 12, 13. Să se scrie o aplicație care permite aprinderea acestor 4 LED-uri. Se va analiza schema electrică pentru a vedea modul de conectare al acestor pini.

Codul corespunzător pentru această problemă ar putea fi:

```
void setup()
{
	DDRB = 0x3C;	//configurare direcție liniilor din registrul B ca
			//și ieșire
			//unde sunt legate cele 4 LED-uri
}
int main()
{
	setup();
	PORTB = 0x00;	//aprinderea celor 4 LED-uri
	while (1)
	{
	}
}
```

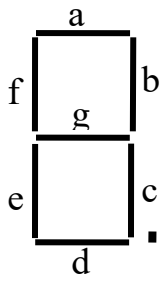
Problema 2. Să se modifice aplicația anterioară astfel încât la apăsarea butonului S1 (conectat pe pinul A1 din portul C) cele 4 LED-uri să se aprindă iar la apăsarea butonului S2 (conectat pe pinul A2 din portul C) cele 4 LED-uri să se stingă.

Problema 3. Să se modifice aplicația 2 astfel încât ledurile să comute între următoarele 2 stări (la apăsarea celor 2 butoane):

D4	D3	D2	D1
X	-	-	X
-	X	X	X

Problema 4. Să se scrie un program care afișează pe cei 4 digiți cu 7 segmente de pe shield-ul de învățare numărul semi-grupei.

Cele 7 segmente de la un digit sunt numerotate de la a la g și sunt așezate astfel:



Obs:

- Pe fiecare digit exista și segmentul 8 care este punctul;
- Pentru afișarea cifrelor în hexazecimal (de la 0 la F) valorile pentru „B” și „D” se vor afișa pe display cu litere mici pentru a nu se confunda cu cifrele „8” și „0”.

Pentru comanda celor 4 digiți avem la dispoziție 2 circuite de memorare (latch-uri 74HC595) legate în serie și 3 linii de comandă:

- Linia LCHCLK activă pe LOW permite selectarea celor 2 latch-uri (Port D bit 4)
- Linia SFTCLK – este linia CLOCK pentru programarea celor 2 latch-uri (Port D bit 7)
- Linia SDI – este linia de DATE (Port B bit 0)

După selectarea circuitului se transmite pe rând cei 2 bytes care sunt reprezentați astfel:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	punct	g	f	e	d	c	b	a
Byte 2	-	-	-	-	Digit 4	Digit 3	Digit 2	Digit 1

Primul byte conține valoarea care se va afișa pe led-uri (sunt codificate cele 7 segmente ale digitului) și al doilea byte specifică în partea mai puțin semnificativă digitul pe care se va afișa valoarea dorită. Cei doi byte se vor transmite bit cu bit începând cu cel mai semnificativ bit din primul byte. Primul byte care conține valoarea dorita este pe logica negativă (se pune valoare ”0” logic pe LED-ul care se dorește aprins) iar al doilea byte este pe logica normală (se pune valoarea ”1” pe digitul care se dorește folosit).

Pașii necesari pentru scrierea pe digiți sunt:

1. Se selectează cele două latch-uri (Linie LCHCLK pe ”0”) și se pune linia de date SDI pe ”0”;
2. Se pune linia SFTCLK pe ”0”;
3. Se pune bitul dorit pe linia SDI;
4. Se pune linia SFTCLK pe ”1”;
5. Se repetă pașii 2-4 pentru toți biții din cei doi byte (chiar si pentru biții 4-7 din Byte 2);
6. Se pune linia SDI pe ”0”;
7. Se de-selectează circuitele punând linia LCHCLK pe ”1”.

În bucla principală, la viteză maximă, se vor reafișa valorile de pe cei 4 digiți.

Problema 5. Să se scrie o aplicație care afișează pe digitul 1 descris în problema anterioara valoarea reprezentată de tasta care este apăsată, folosind o tastatură matriceală 4x4 conectată la dispozitiv.

Pentru controlul celor 16 taste, tastatura matriceală pune la dispoziție 8 pini dintre care 4 sunt pentru controlul celor liniilor și 4 sunt pentru controlul coloanelor. La intersecția unei coloane cu o linie se găsește o tasta. Tastatura matriceală este conectată la dispozitiv astfel (s-a ales această conectare pentru compatibilitatea cu learning bord-ul folosit până acuma) și se recomandă configurarea liniilor ca și pini de intrare și a coloanelor ca și pini de ieșire:

Pini tastatură matriceală	Pin controler Arduino
Linie 1	PIN 2 (PORTD bit 2)
Linie 2	PIN 3 (PORTD bit 3)
Linie 3	PIN 5 (PORT D bit 5)
Linie 4	PIN 6 (PORTD bit 6)
Col 1	PIN 10 (PORTB bit 2)
Col 2	PIN 11 (PORTB bit 3)
Col 3	PIN 12 (PORTB bit 4)
Col 4	PIN 13 (PORTB bit 5)

Pentru a putea folosi pe liniile de intrare rezistența de PULL-UP existentă pe Arduino (rezistentă care ține linia de intrare la 5V în mod natural iar din exterior

noi putem pune doar valoarea de 0V) se recomandă realizarea următoarelor configurări:

1. Bitul 4 (PUD – PULL-UP Disable) din registrul MCUCR configurat pe ”0”, implicit acest bit este configurat pe ”0” deci PULL-UP este activ. Mai multe detalii în specificații Atmega pag. 120 și pagina 102.
2. Bitul corespunzător din registrul DDRx configurat pentru intrare (deci valoarea ”0”)
3. Bitul corespunzător din registrul PORTx setat pe ”1” înainte de a se realiza operația de citire de pe portul respectiv, iar pentru citirea registrului trebuie folosit PINx.

Dacă se activează rezistența de PULL-UP toate liniile de intrare de la tastatură, dacă nici o tasta nu este apăsată, nu sunt în aer ci sunt la 5V datorită rezistenței. În acest caz toate coloanele de la tastatură, care sunt de ieșire, trebuie să le ținem în 5V în mod normal. Pentru a verifica dacă o anumită tastă de pe o anumită coloană este apăsată se parcurg următorii pași:

1. Pe bitul corespunzător coloanei pe care vrem să o testăm se pune valoare ”0”.
2. Pe portul în care se găsesc conectate cele 4 linii de la tastatură se scrie valoarea ”1” pe biții respectivi (pentru a activa rezistența de PULL-UP)
3. Se citește valoarea din portul în care se găsesc cele 4 lini folosind instrucțiunea PINx.
4. Pe bitul corespunzător coloanei selectate la pasul 1 se pune valoarea ”1”.
5. Se verifică pe rând cei 4 biți din registrul citit la pasul 3 iar dacă bitul corespunzător unei linii este pe ”0” înseamnă că butonul de la intersecția coloanei selectate la pasul 1 și linia curentă a fost apăsat.

2 Arduino. Timer

2.1 Caracteristici generale

Controlerul ATmega are disponibile pentru programator 5 timere. Un timer sau numărător este un circuit integrat care permite măsurarea unei perioade programate de timp (numărarea până la o anumită valoare cu o anumită viteză fixă) și anunțarea programatorului când trece perioada specificată de timp.

Timer 0 - este un timer pe 8 biți sincron și este folosit pentru realizarea de funcții de întârziere de genul delay()

Timer 1,3,4 – este un timer pe 16 biți sincron cu PWM

Timer 2 – este un timer pe 8 biți asincron

Pentru fiecare timer poate fi selectată o sursă separată de semnală care să genereze frecvența de numărare. De obicei se folosește frecvența CPU care este 16MHz pentru Arduino. Contorul maxim (valoarea maximă de numărare) pentru un timer este de 256 pentru timerele pe 8 biți și 65536 pentru cele pe 16 biți. În funcție de necesitate se poate alege un time pe 8 sau pe 16 biți.

Timer-ul poate fi configurat în mai multe moduri:

- Normal Mode – numărare în sus (incrementare) nu este resetat automat
- CTC – Clear Timer on Compare Match. Atunci când contorul timer-ului atinge valoare din registrul de numărare timerul va fi resetat. Registru OCRx este utilizat pentru a manipula rezoluția contorului. Contorul este reseta la 0 când valoarea acestuia atinge valoarea de numărare.
- Fast PWM – Pulth Width Modulation - ieșirea din registrul OCRx este utilizată pentru a genera semnal PWM

2.2 Regiștrii Arduino pentru Timer 1

Timer 1 este un timer / numărător pe 16 biți care are 2 unități de ieșire de comparare independente și 2 regiștrii de comparare.

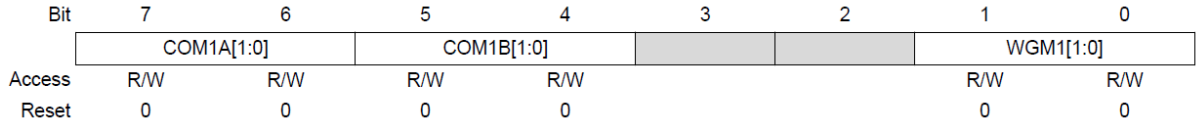
În continuare prezentăm configurarea pentru Timer 1 (TC1) care este un Timer/Countre pe 16 biți cu PWM. Pentru celelalte timere se poate folosi documentația completă de la ATmega.

Timer 1 are următoarele caracteristici:

- Permite configurare de contoare pe 16 biți, deci o rezoluție mult mai bună
- 2 unități de ieșire de comparare notate cu A și B, astfel poate fi programat cu 2 perioade diferite de timp.
- O unitate de intrare de comparare
- Permite programarea cu încărcare automată după numărare

Se poate schimba comportamentul unui timer prin intermediul regiștrilor dedicați pentru configurare. În continuare prezentăm regiștrii timer-ului 1:

2.2.1 TCCRxA – Timer/Counter Registrul de Control A



Biții 6:7, 4:5 – COM1A/B – Comparare mod de ieșire pentru canal

Controlează comportamentul pinilor de comparare de ieșire. Prezentăm configurarea pinilor pentru modul de lucrul al time-ului fără PWM. Pentru acest laborator recomand configurarea ca ”Normal port operation” și modul de operare Normal. Detalii despre acest registru în documentația ATmega pagina 179.

Biții 7:6 Biții 5:4	Descriere
00	Funcționare în modul normal.
01	Comutare între OC1A/OC1B pentru comparare
10	Ștergere OC1A/OC1B la potrivire comparare. Setează ieșirea la Low.
11	Setare OC1A/OC1B la potrivire comparare. Setează ieșirea la High.

Biții 1:0 – WGM – Descrierea modului de generare a formei de undă. Acești biți sunt considerați împreună cu biții 3 și 4 din registrul TCCRxB

Biții WGMx[3:0]	Descriere
0000	Normal (counter)
0001, 0010,0011	PWM, Fază corectă pe 8,9,10 biți
0100	CTC (Clear Timer on Compare Match), pag.169
0101, 0110, 0111	Fast PWM, 8,9,10 biți
1000, 1001, 1010, 1011	PWM, Fază și Frecventă corectă
1100	CTC
1100	Rezervat
1110	Fast PWM
1111	Fast PWM

2.2.2 TCCRxB – Timer/Counter Registrul de Control B

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM1[3]	WGM1[2]	CS1[2:0]		
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Bit 7 – ICBC1 – Filtrare pin de Input Capture - Activat pe 1 va filtra intrarea de pe pinul ICP1. Funcția de filtrare necesită 4 valori succesive ale eșantioanelor de pe pinul ICP1pentru modificarea ieșirii. Se folosește pentru a evita citirea și interpretarea fluctuațiilor pe pinul de intrare până se stabilizează linia la trecerea dintr-o stare în alta.

Bit 6 – ICES1 – Selecție front pentru pinul de Input Capture – Specifică care front este folosit pentru a declanșa evenimentul la comutare. Valoarea de 0 înseamnă folosirea frontului descrescător (trecerea din 1 în 0) iar valoarea de 1 înseamnă folosirea frontului crescător.

Biții 4:3 – descriși la registrul TCCRxA

Biții 2:0 - CS1 – selecție semnal de ceas. Permite specificarea valorii de prescalare pentru semnalul de ceas (schema de la pagina 135 si detalii la 215).

Biții CS1[2:0]	Descriere
000	Fără semnal de ceas, Timer oprit
001	$clk_{I/O}$ / 1 – fără prescalare
010	$clk_{I/O}$ / 8 – cu prescalare
011	$clk_{I/O}$ / 64 – cu prescalare
100	$clk_{I/O}$ / 256 – cu prescalare
101	$clk_{I/O}$ / 1024 – cu prescalare
110	Sursă de ceas externă. Activare pe front descrescător
111	Sursă de ceas externă. Activare pe front crescător.

Și acest registru recomand pentru acest laborator configurare fără Input Capture, modul de operare Normal și selectarea pentru valoarea de prescalare dorită (1, 8, 64, 256 sau 1024). Detalii despre acest registru în documentația ATmega pagina 182.

2.2.3 TCNT1 (TCNT1L și TCNL1H) – Valoarea contorului pe 16 biți (Low și High)

Bit	15	14	13	12	11	10	9	8
	TCNT1[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	TCNT1[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Păstrează valoarea de numărare, pe 16 biți, de la care timerul începe și crește/scade valoarea până la resetarea acestui registru sau până valoarea din acest registru este egală cu valoarea din registrul OCR1A sau OCR1B(depinde de configurare). Detalii despre acest registru în documentație ATmega pagina 185.

Pentru a calcula perioada unui timer in modul CTC se folosește formula:

$$T_{Timer} = T_{CLK} * N * OCR1n \quad astfel \quad OCR1n = \frac{T_{Timer}}{T_{CLK} * N}$$

Unde N reprezintă factorul de prescalare (care poate fi 1, 8, 64, 256 sau 1024)

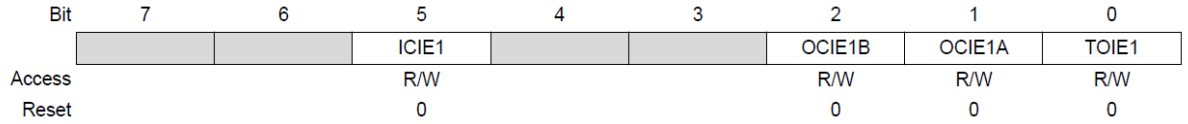
2.2.4 OCR1A (OCR1AL și OCR1AH), OCR1B (OCR1BL și OCR1BH)

Bit	15	14	13	12	11	10	9	8
	OCR1A[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	OCR1A[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Registru comparare de ieșire pentru canalul A sau canalul B pe 16 biți. Detalii despre acest registru la pagina 187.

2.2.5 TIMSK1 - registrul mascare întrerupere timer 1



Bit	7	6	5	4	3	2	1	0
			ICIE1			OCIE1B	OCIE1A	TOIE1
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Permite validarea / invalidarea generării de întrerupere pentru timer 1. Valoarea 1 pe un anumit bit va duce la generarea întreruperii corespunzătoare (atunci când este cazul).

Bit 5 – ICIE1 - Generare întrerupere pe Input Capture - 1 pe acest bit va valida generarea unei întreruperi atunci când se schimba semnalul, conform configurării, pe Input Capture

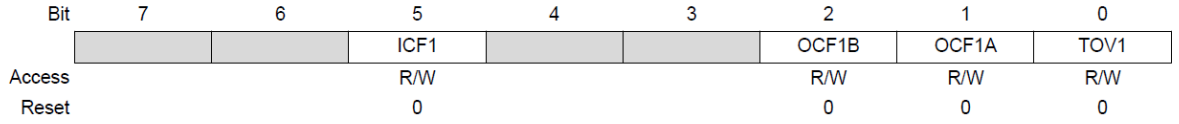
Bit 2 – OCIE1B – Activare generarea unei întreruperi la potrivirea ieșirii pe comparatorul B

Bit 1 – OCIE1A – Activare generarea unei întreruperi la potrivirea ieșirii pe comparatorul A

Bit 0 – TOIE1 – Activare generarea unei întreruperi atunci când apare depășire pe contor.

Detalii în documentația ATmega la pagina 209.

2.2.6 TIFR1 – registru flag de întrerupere



Bit	7	6	5	4	3	2	1	0
			ICF1			OCF1B	OCF1A	TOV1
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Indică o întrerupere, dacă aceasta este validă. Bitul corespunzător poate fi interogat pentru a vedea dacă a expirat perioada de numărare.

Bit 5 – ICF1 – bit setat atunci când apare Input Capture (se generează și o întrerupere dacă este activă).

Bit 2 – OCF1B – bit setat atunci când apare potrivire pe comparatorul B (se generează și o întrerupere dacă este activă).

Bit 1 – OCF1A – bit setat atunci când apare potrivire pe comparatorul A (se generează și o întrerupere dacă este activă).

Bit 0 – TOV1 -bit setat atunci când apare depășire.

Detalii în documentația ATmega la pagina 212.

2.3 Probleme propuse

Problema 1. Să se scrie o aplicație care schimă periodic la o secundă valorile afișate pe leduri folosind un timer prin interogare. Cele două valori afișate pe leduri sunt:

D4	D3	D2	D1
X	-	-	X
-	X	X	X

Pași necesari pentru configurare timer 1:

1. Se configurează timerul conform valorilor dorite in registrul TCCR1A și TCCR1B (valoarea de prescale).
2. Se încarcă registrii OCR1A(Low si High) cu valoare de numărare dorită. Valoarea de numărare se calculează în funcție de perioada la care se dorește apariția timer-ului conform formulei de mai sus. Se resetează registrul TCNT1 (Timerul 1 în loc să utilizeze flag-ul de overflow, utilizează registrul OCR de comparare și flagul corespunzător OCF).
3. În bucla programului principal se monitorizează flagul OCF corespunzător prin interogară și se așteaptă setarea acestuia.
4. Se resetează registrul TCNT1 și se setează flagul OCF și se repetă pașii 3 și 4.

De exemplu pentru a configura Timerul 1 la valoarea de 100ms (0.1s) și cu prescalare de 1024, valoarea care trebuie înscrisă în registrul OCR1 este:

$$OCR1R = \frac{0.1s}{\frac{1}{16.000.000} * 1024} = 1562$$

Problema 2. Să se scrie o aplicație care va afișa pe cele 4 leduri D1,D2,D3,D4 următoarea secvență.

D4	D3	D2	D1
-	-	-	-
-	-	-	X
-	-	X	X
-	X	X	X
X	X	X	X
-	X	X	X

-	-	X	X
-	-	-	X

Problema 3. Să se modifice problema 1 astfel încât să se aprindă pe rând câte un LED din cele 4 dacă butonul S1 nu este apăsat și să fie stins doar un LED din cele 4 dacă butonul S1 este apăsat.

Stare buton S1	D4	D3	D2	D1
neapăsat	-	-	-	X
	-	-	X	-
	-	X	-	-
	X	-	-	-
apăsat	X	X	X	-
	X	X	-	X
	X	-	X	X
	-	X	X	X

Problema 4. Să se modifice problema anterioară astfel încât să se utilizeze timerul 1 prin întrerupere.

Pentru lucrul prin întreruperi în plus față de configurarea standard trebuie activată și generare unei întreruperi atunci când comparatorul între TCNT1 și registru OCR1A sau OCR1B întoarce egalitate. Pentru a scrie o rutină de tratare a întreruperii se scrie o funcție cu numele ISR iar în paranteză se scrie vectorul de întrerupere pentru care se dorește apelul acelei rutine de întrerupere. Tabela cu vectorii de întrerupere pentru controlerul ATmega338 se găsește în documentație la pagina 85. În rutina de tratare a întreruperii trebuie resetat (sau rescris) registrul TCNT1.

Exemplu pentru rutina de tratare a întreruperii configurată pentru generare întrerupere pentru compararea cu registrul OCR1A

```
ISR(TIMER1_COMPA_vect)

{

      TCNT1 = 0;

      //your program here

}
```

Exemplu pentru rutina de tratare a întreruperii configurată pentru generare întrerupere pentru overflow pe registrul TCNT1

```
ISR(TIMER1_OVF_vect)

{

      TCNT1 = valoare_de_numărare;

      //your program here

}
```

Înainte de configurarea timerului prin întrerupere trebuie dezactivate toate întreruperile folosind funcția `noInterrupts();` iar la finalul configurării trebuie activate toate întreruperile folosind funcția `interrupts();`

Problema 5. Să se scrie un program de comandă a unui motor pas cu pas pe portul paralel. Se recomandă realizarea următoarele legături între motor și portul paralel:

- comandă înfășurarea 1 a motorului - la pinul 13 al portului paralel;
- comandă înfășurarea 2 a motorului - la pinul 12 al portului paralel;
- comandă înfășurarea 3 a motorului - la pinul 11 al portului paralel;
- comandă înfășurarea 4 a motorului - la pinul 10 al portului paralel;

De la comanda unei înfășurări la comanda unei înfășurări vecine motorul se va deplasa și va face un pas. În funcție de sensul de rotație dorit se poate comanda înfășurările în diferite direcții. Pentru un cuplu al motorului mai bun se poate comanda la un moment dat câte 2 înfășurări simultan. Se poate face ca motorul să execute la un moment dat doar o jumătate de pas (numit comandă cu micro-pășire) atunci când alternativ se va alimenta prima data de exemplu înfășurarea 1 apoi simultan înfășurarea 1 și 2, apoi doar înfășurarea 2 și tot așa. După trimiterea unei comenzi fiecare motor, în funcție de performanțele acestuia, are nevoie de o perioada scurtă de timp (de câteva ms) pentru a și executa comanda. În funcție de perioada dintre 2 comenzi se poate regla viteza cu care se rotește un motor.

3 Arduino. Portul Serial

3.1 Caracteristici generale

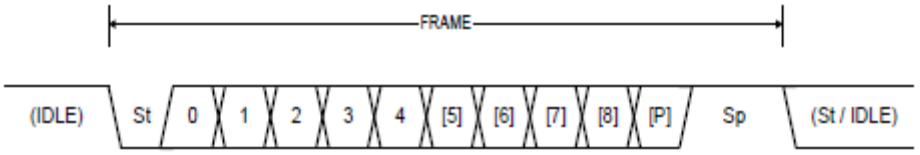
Controlerul ATmega are disponibile pentru programator de două porturi de comunicație serială notate USART0 și USART1 (Universal Synchronous Asynchronous Receiver Transceiver). Ambele porturi de comunicație serială pot funcționa full duplex, sincron sau asincron la viteze mari de comunicație. Nivelele de tensiune pe porturile seriale de la controlerul ATmega respectă standardul TTL și nu pot fi conectate direct pe portul serial de la un PC, care respecta standardul RS232. Pentru a putea fi conectate trebuie folosit un convertor de la TTL la RS232.

Portul de comunicație serială poate fi configurat în mai multe moduri:

- Asynchronous Normal Mode – mod normal de comunicație asincron (bit UMSEL = 0 și U2Xn = 0)
- Asynchronous Double Speed mode – viteza dublă de comunicație mod asincron (bit UMSEL = 0 și U2Xn=1)
- Synchronous Master mode – mod normal comunicație sincronă (bit UMSEL = 1)

Formatul unui frame de comunicație serială conține următoarele informații:

- 1 bit de start
- 5-9 biți de date
- fără paritate sau 1 bit de paritate pară / impară
- 1 sau 2 biți de stop



3.2 Regiştrii ATmega pentru USART0

Pentru configurarea comunicației pe portul serial controlerul ATmega pune la dispoziție 5 regiștrii de configurare / comunicație. Un registru numit UDR0 pentru transmisie / recepție date, un registru pe 16 biți pentru configurare viteză de comunicație numit UBRR0 (format din UBRR0L și UBRR0H) și 3 regiștrii de

control și stare notați UCSR0A, UCSR0B, UCSR0C. Detalii complete despre funcționarea acestor regiștrii se găsesc în documentația ATmega paginile 284-291.

3.2.1 UDR0 – USART0 registrul de date de intrare/ieșire

Este registru de date pentru comunicația serială care permite transmiterea / recepționarea caracterelor pe / de pe portul serial. Scrierea în acest registru va declanșa transmiterea biților corespunzători pe portul serial. Citirea acestui registru va returna ultimul caracter recepționat pe portul serial și care se găsește în bufferul de recepție de date.

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – TXB / RXB[7:0] USART Transmit / Receive Data Buffer

3.2.2 UCSR0A – USART0 registrul de control și stare A

Acest registru permite verificarea stării portului de comunicație serială pentru a vedea dacă se poate transmite următorul caracter sau s-a recepționat un caracter sau a apărut o eroare de comunicație.

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

Bit 7 - RXCn – Recepție caracter – acest bit este pe 1 atunci când sunt date recepționate corect în bufferul de recepție care nu au fost citite încă. Se resetează automat după citirea registrului de date. Poate fi utilizat pentru generarea unei întreruperi.

Bit 6 - TXCn – Transmisie caracter – acest bit este pe 1 atunci când s-a transmis ultimul bit din caracterul curent și când nu mai sunt date curente prezente în bufferul de transmisie. Acest bit poate fi șters prin scrierea valorii 1 pe poziția corespunzătoare. Acest bit este monitorizat atunci când vrem să știm dacă s-a terminat transmisia. Poate fi utilizat pentru generarea unei întreruperi.

Bit 5 - UDREn – Registrul de date gol – acest bit este pe 1 atunci când bufferul de transmisie date este gol și poate fi transmis următorul caracter. Acest bit se verifică înainte să începem transmisia unui caracter. Poate fi utilizat pentru a genera o întrerupere.

Ceilalți biți sunt biți de eroare și trebuie testați atunci când dorim să vedem dacă a apărut o eroare (documentație ATmega pag. 284)

3.2.3 UCSR0B – USART0 registrul de control și stare B

Acest registru permite configurarea modului de funcționare a portului de comunicație serială.

Bit	7	6	5	4	3	2	1	0
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 - RXCIEn – acest bit îl punem pe 1 atunci când dorim să activăm generarea unei întreruperi la recepție de date.

Bit 6 - TXCIEn – acest bit îl punem pe 1 atunci când dorim să activăm generarea unei întreruperi atunci când s-a terminat de transmis datele din bufferul de transmisie date.

Bit 5 - UDRIEn – acest bit îl punem pe 1 atunci când dorim generarea unei întreruperi atunci când bufferul de transmisie date este gol. Această întrerupere apare chiar dacă înainte nu am transmis nimic pe portul serial comparativ cu întreruperea generată la activarea bitului TXCIEn care apare doar după ce am transmis un caracter.

Bit 4 - RXENn – activare recepție. Punerea acestui bit pe 1 va permite funcționarea portului serial și în modul de recepție date. Atâta timp cât acest bit este pe 0 orice caracter recepționat este ignorat.

Bit 3 - TXENn – activare transmisie. Punerea acestui bit pe 1 va permite funcționarea portului serial și în modul de transmisie date. Atâta timp cât acest bit este pe 0 orice caracter scris în registrul de date de la portul serial va fi ignorat.

Bit 2 - UCSZn2 – dimensiune caracter – împreună cu biții 0 și 1 din registrul UCSR0C permite specificarea numărului de biți de date care vor fi transmiși /recepționați pe portul serial.

Bit 1 – RXB8n – bitul de date 8 – este al nouălea bit de date recepționat pe portul serial dacă sa configurat frame-ul de comunicație serială pe 9 biți.

Bit 0 – TXB8n – bitul de date 8 – este al nouălea bit de date din caracterul care trebuie transmis dacă s-a configurat frame-ul de comunicație pe 9 biți. Se va scrie acest bit înainte să se scrie restul de 8 biți în registrul de date.

3.2.4 UCSR0C – USART registrul de control și stare C

Acest registru permite configurarea modului în care se comunică pe portul serial.

Bit	7	6	5	4	3	2	1	0
	UMSELn[1:0]		UPMn[1:0]		USBSn	UCSZn1 / UDORDn	UCSZn0 / UCPHAn	UCPOLn
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Biții 7:6 – UMSELn – permit selectarea modului de funcționare a portului de comunicație serială

Biții 7:6	Descriere
00	Comunicație asincrona
01	Comunicație sincrona
10	Rezervat
11	SPI master (sincon)

Biții 5:4 – UPMn – permite specificarea modului de calcul a parității

Biții 5:4	Descriere
00	Fără paritate
01	Rezervat
10	Paritate pară
11	Paritate impară

Bit 3 – USBSn – numărul de biți de STOP. Scrierea valori 0 pe acest bit înseamnă 1 bit de STOP iar scrierea valorii de 1 pe acest bit înseamnă 2 biți de STOP

Bit 2:1 – UCSZn1, UCSZn0 – împreună cu bitul UCSZn2 din registrul de control B permite specificarea numărului de biți de date.

UCSZn[2:1]	Descriere
000	5 biți de date

001	6 biți de date
010	7 biți de date
011	8 biți de date
100, 101, 110	rezervați
111	9 biți de date

Bit 0 – UCPOLn – polaritate semnal ceas pentru modul sincron. Pentru modul asincron trebuie inițializat cu 0.

3.2.5 UBRR0H(L) – USART registrul de Baud Rate partea High si Low

Acest registru permite scrierea valorii de divizate pe 12 biți care va duce la obținerea valorii de baud rate (număr de biți/s) dorite. Registrul UBRRnH conține cei mai semnificativ 4 biți din valoarea de divizare iar registrul UBRRnL conține cei mai puțin semnificativ 8 biți din valoarea de divizare.

Bit	15	14	13	12	11	10	9	8
					UBRRn[11:8]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bit	7	6	5	4	3	2	1	0
	UBRRn[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Pentru calcularea valorii de divizare în modul de comunicație asincron se poate folosi următoarea formulă:

$$BAUD = \frac{fosc}{16\,(UBRRn+1)} \qquad UBRRn = \frac{fosc}{16*BAUD} - 1$$

Unde f_{OSC} este frecvența oscilatorului folosit pentru comunicația serială (de obicei se folosește frecvența CPU care în cazul nostru este 16MHz). De obicei obținem 2 valori apropiate pentru același baud rate caz în care se va alege valoare care produce eroarea cea mai mică.

3.3 Probleme propuse

Problema 1. Să se scrie o aplicație care configurează portul serial pentru a funcționa la anumiți parametrii de comunicație (alegeți viteza de comunicație, nr.

biți date, nr. biți STOP, paritate) și care să transmită pe linia serială, în buclă, toate caracterele din alfabet (litere mari de la ’A’ la ’Z’).

Problema 2. Să se realizeze o aplicație care transmite pe portul serial caracterul recepționat (așa cum a venit) dacă SW1 nu este apăsat și îl convertește din literă mare în literă mică (dacă este cazul) în cazul în care SW1 este apăsat.

Problema 3. Să se scrie o aplicație care răspunde la următoarele comenzi recepționate pe portul serial:

- ’0’ – ’3’ – aprinde corespunzător unul din cele 4 leduri
- ’8’ – aprinde toate cele 4 leduri
- ’9’ – stinge toate cele 4 leduri
- În rest trimite codul înapoi la calculator

Problema 4. Să se modifice problema anterioară astfel încât recepția de caractere să se realizeze prin întrerupere.