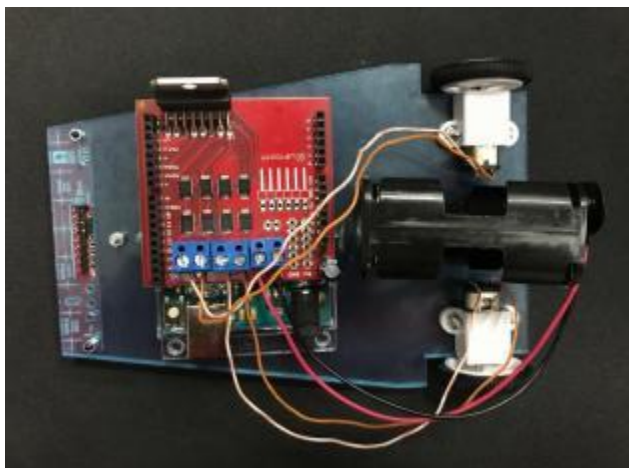


RoboSmart



**CONCURS DE
ROBOTICĂ
RoboSmart**

Ediția 1
2017-2018



DEPARTAMENTUL
ELECTRONICĂ, CALCULATOARE
ȘI INGINERIE ELECTRICĂ

**Universitatea din Pitești
Facultatea de Electronică,
Comunicații și Calculatoare**



INSPECTORATUL ȘCOLAR
JUDEȚEAN ARGEȘ

Sponsori



Arduino – Programarea robotului

- Structura unui program Arduino
- Controlul pinilor machetei
- Comanda motoarelor
- Noțiuni avansate
- Portul serial



Prezintă:



Florin-Marian Bîrleanu

<http://igotopia.ro>

Structura unui program Arduino

Sketch = Structure + (Values + Functions)

- **Program** = *Instrucțiune1 Instrucțiune2 ...*
- *Instrucțiune* = Definiție de **funcție**
sau
Definiție de **constantă** sau **variabilă** (globală)
- Obligatoriu: Definiția funcției **setup()**
Definiția funcției **loop()**

Structura unui program Arduino

- => Cel mai scurt program Arduino:

```
void setup() {  
}
```

```
void loop() {  
}
```

- Ce înseamnă **setup()** și **loop()**?

```
void main() {  
    setup();  
    while (1) {  
        loop();  
    }  
}
```

Structura unui program Arduino

- Funcția **setup()**:
 - E apelată **automat** la pornirea programului
 - (Adică atunci când se alimentează macheta sau când se apasă butonul de reset)
 - Trebuie puse acolo **inițializări**, lucruri care vor rula o singură dată

Structura unui program Arduino

- Funcția **loop()**:
 - E apelată într-o buclă infinită
 - Aici se fac citiri de intrări, calcule, scrieri de ieșiri
 - Dar când se termină programul? (Niciodată!)
 - La “stingerea” plăcuței.

Structura unui program Arduino

- Constante:

```
#define NUME valoare  
const tip nume = valoare;
```

← !...

← 😊

- Variabile: ← globale

```
tip nume;
```

sau:

```
tip nume = valoare;
```

129, 0127,
0x12F,
B01110101
(0b01110101)

Tipuri (noi): boolean, byte, word, String

Variabilele într-un program Arduino

- Globale ← vizibile peste tot
- Locale ← vizibile în blocul lor
- Statice ← ca globale, pentru o funcție
- Volatile ← cu acces concurrent



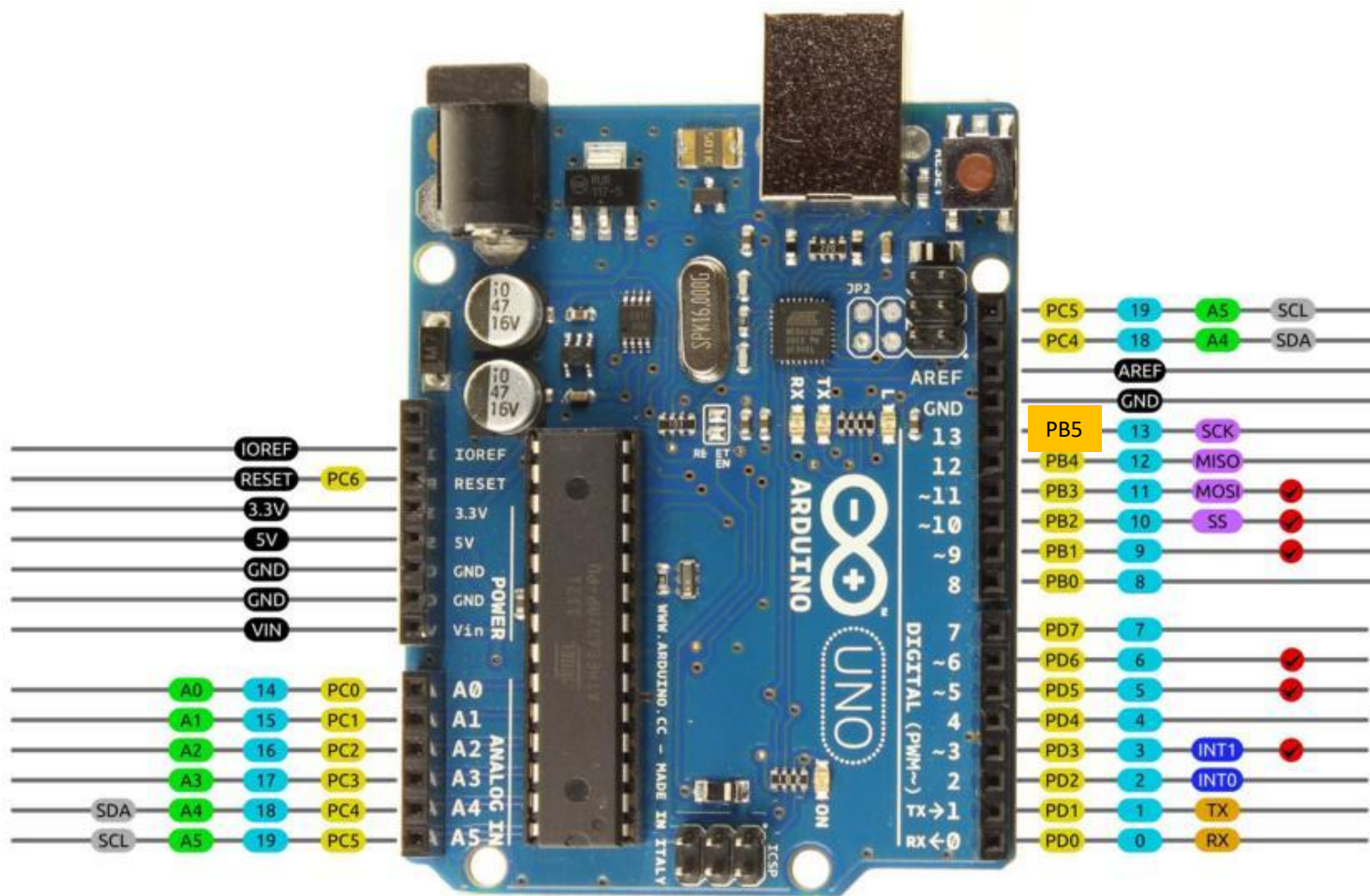
volatile împiedică eventuale optimizări pe care le-ar putea face compilatorul; obligă recitirea valorii din memorie la fiecare accesare a variabilei

Controlul pinilor în Arduino

- Pini digitali
- Pini analogici
- Pini PWM



Controlul pinilor în Arduino



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT

Pinii digitali

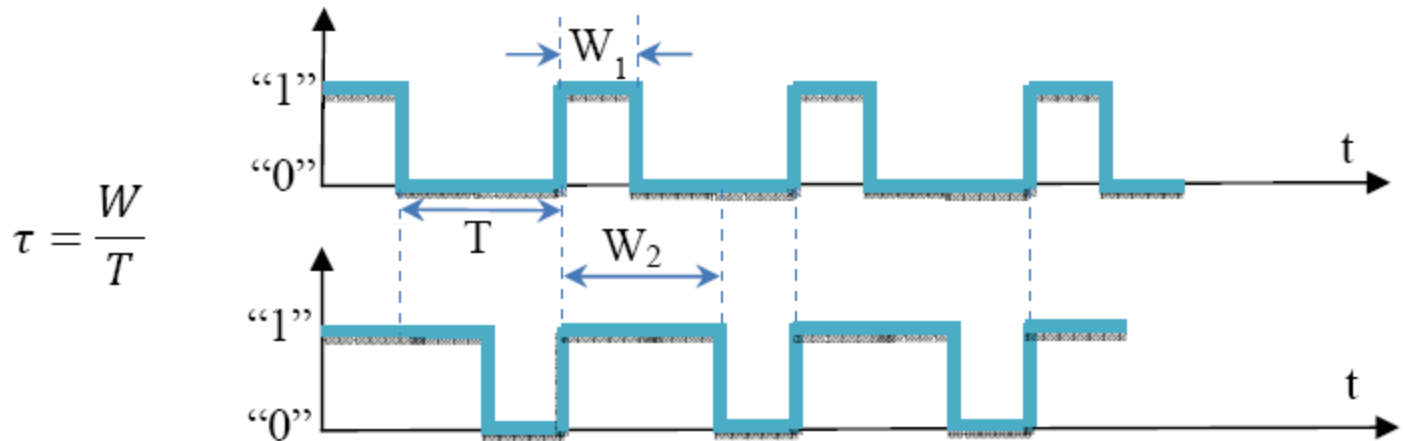
- Intrări sau ieșiri digitale ← (0,1, 2, ..., 13, A0, A1, ..., A5)
- Valori posibile:
0 (LOW), 1 (HIGH)
- Configurarea:
`pinMode(nr_pin, {INPUT, OUTPUT, INPUT_PULLUP});`
- Scrierea: ← Aprinderea unui LED
`pinMode(nr_pin, OUTPUT);`
`digitalWrite(nr_pin, HIGH);`
- Citirea: ← Citirea unui buton
`pinMode(nr_pin, INPUT_PULLUP);`
`{HIGH, LOW} = digitalRead(nr_pin);`

Pinii analogici

- Intrări analogice ← (A0, A1, ..., A5)
- Valori posibile:
 - 0 (0V), 1023 (5V)
 - ADC/10b => rezoluția = 5V/1024
 - O citire durează 100 μs
 - **analogReference**({DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL});
- Citirea: ← Citirea unei tensiuni
int valoare = analogRead(nr_pin);

Pinii PWM

- Ieşiri “analogice” ← (3, 5, 6, 9, 10, 11)
- “Valori” posibile:
0 (oprit, $\tau=0$), 255 (pornit la maxim, $\tau=1$)



- Scrierea:
`analogWrite(nr_pin, {0, 1, 2, 3, ..., 255});`

Comanda motoarelor

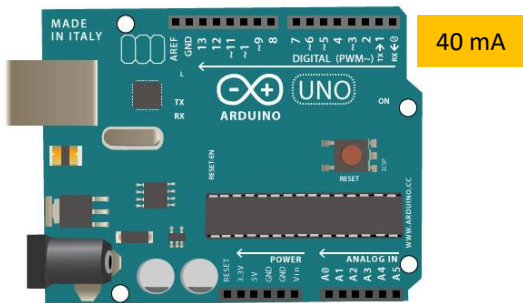
- Motor =
 - masină electrică ce convertește energia electrică în energie mecanică
 - rotor + stator
- Motor de c.c.



Parametri: 6 V, 120 mA (max: 1.6 A)

Comanda motoarelor

- Ce e driver-ul de motor?



40 mA



2000 mA



Comanda motoarelor

- Sens rotație?

- Înainte: $M_{xA} \leftarrow \text{HIGH}; M_{xB} \leftarrow \text{LOW}$

- Înapoi: $M_{xA} \leftarrow \text{LOW}; M_{xB} \leftarrow \text{HIGH}$

- La stânga: $M_d - \text{înainte}; M_s - \text{oprit}$

- Sau: $M_d - \text{înainte}; M_s - \text{înapoi}$

- La dreapta: $M_d - \text{oprit}; M_s - \text{înainte}$

- Sau: $M_d - \text{înapoi}; M_s - \text{înainte}$

Comanda motoarelor

- Viteză rotație?

- Oprit:

$M_{xA} \leftarrow \text{LOW}; M_{xB} \leftarrow \text{LOW}$

- Pornit (la maxim):

$M_{xA} \leftarrow \text{HIGH}; M_{xB} \leftarrow \text{LOW}$

- La $v\%$ din maxim:

$M_{xA} \leftarrow \text{PWM}(v); M_{xB} \leftarrow \text{LOW}$

Notiuni avansate

- Controlul direct al pinilor

Operații pe biți

- Lucrul cu întreruperi

Temporizări

- C pentru Arduino

Utilizarea memoriei de program pentru date



Controlul direct al pinilor

- Pro:
 - Mai rapid
 - Se pot controla simultan mai mulți pini
 - Programul rezultat ocupă mai puțin
- Con:
 - Mai dificil de înțeles
 - Mai deschis către erori
 - Mai puțin portabil

Controlul direct al pinilor

- **3 porturi:**
 - **B** (pinii 8-13)
 - **C** (pinii ADC)
 - **D** (pinii 0-7)
- Fiecare e controlat de către 3 **registri:**
 - **DDRx** – Port x **D**ata **D**irection **R**egister (R-W)
 - **PORTx** – **P**ort x data register (R-W)
 - **PINx** – **P**ort x **I**nput pins register (R)

Controlul direct al pinilor

- Exemplu:

DDRD = B11111110;

ooooooooi

← o – output; i – input

PORTD = B10101001;

HLHLHLL?

← H – high (1); L – low (0)

byte D = PIND;

76543210

← numerotarea biților

Operații la nivel de bit

- Extragere biți
- Setare biți
- Resetare biți
- Inversare biți



Operații la nivel de bit

Extragere biți

(← bitRead)

0 1 1 1 1 0 1 0 &
0 0 0 0 1 0 0 0
=> 0 0 0 0 1 0 0 0

← masca

← rezultatul

0 1 1 1 1 0 1 0 &
0 0 0 0 0 1 0 0
=> 0 0 0 0 0 0 0 0

== (1 << 2)

Operații la nivel de bit

Setare biți

(← bitSet)

0 1 1 1 1 0 1 0 |

0 0 0 0 1 0 0 0

← masca

=> 0 1 1 1 1 0 1 0

← rezultatul

0 1 1 1 1 0 1 0 |

0 0 0 0 0 1 0 0

== (1 << 2)

=> 0 1 1 1 1 1 1 0

Operații la nivel de bit

Resetare biți

(← bitClear)

0 1 1 1 1 0 1 0 &
1 1 1 1 0 1 1 1 ← masca
=> 0 1 1 1 0 0 1 0 ← rezultatul

0 1 1 1 1 0 1 0 &
1 1 1 1 1 0 1 1 == ~000000100 == ~(1<<2)
=> 0 1 1 1 1 0 1 0

Operații la nivel de bit

Inversare biți

	0	1	1	1	1	0	1	0	^
	0	0	0	0	1	0	0	0	
=>	0	1	1	1	0	0	1	0	

← masca
← rezultatul

	0	1	1	1	1	0	1	0	^
	0	0	0	0	0	1	0	0	
=>	0	1	1	1	1	1	1	0	

Lucrul cu întreruperi

```
while (true)
{
    if (! INTRERUPERE)
        ExecutaUrmatoareaInstructiune();
    else
        ExecutaISR();
}
```



Întrerupere la apăsarea unui buton

```
volatile byte v = 0;
void inverseaza() { v = 1-v; }
void setup() {
    pinMode(13, OUTPUT);
    pinMode(2, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(2),
                   inverseaza, FALLING);
}
void loop() {
    digitalWrite(13, v);
}
```

Notă: **delay()** și **millis()** nu merg aici.



Întrerupere la 1 Hz (temporizare)

- Se programează direct **timerele**
- => Nu or să mai meargă: `millis()`, `delay()`, `analogWrite()`
- Se lucrează direct cu **regiștrii**: `TCCRxA`, `TCCRxB`, `TCNTx`, `OCRxA`, `TIMSKx`



(Detalii în ATMEL ATmega328P **datasheet**...)

Înterupere la 1 Hz (temporizare)

```
void setup() {  
    pinMode(13, OUTPUT);  
    cli(); // noInterrupts();  
    // set timer 0 interrupt @ 1Hz:  
    TCCR0A = 0;  
    TCCR0B = 0;  
    TCNT0 = 0; // init counter to 0.  
    OCR0A = 15624; //16MHz/1024-1  
    TCCR0A |= (1<<WGM01);  
    TCCR0B |= (1<<CS12) | (1<<CS10);  
    TIMSK0 |= (1<<OCIE1A);  
    sei(); // interrupts();  
}  
  
void loop() {  
    ... // whatever  
}
```

```
boolean toggle0 = 0;
```

```
ISR (TIMER0_COMPA_vect) {  
    toggle0 = 1 - toggle0;  
    digitalWrite(13, toggle0);  
}
```

TCCR1B |= (1<<WGM12); // timer 1
TCCR2A |= (1<<WGM21); // timer 2

Detalii despre acești
registri: în
datasheet-ul
microcontrolerului

Temporizări (foarte) scurte

- `delay(...);`
- `millis()`
- `delayMicroseconds(...);` → 2 μs (2000 ns)
- `__asm("nop\n");` → 62.5 ns



C pentru Arduino

- Tipuri de date: `byte`, `boolean`, `word`, `String`, `int`, `char`, `long`, `float`
- Constante numerice:
 - predefinite: `false`, `true`, `LOW`, `HIGH`, `INPUT`, `OUTPUT`, `INPUT_PULLUP`, `LED_BUILTIN`
 - `123`, `B01110101`, `013`, `0xAB` (eventual cu `u`, `l` sau `ul` la final)
- `sizeof`
(Ex.: `char str[] = "Arduino"; // sizeof(str) → 8`)
- `goto` etichetă

Salvarea constantelor în memoria de program

- `#include <avr/pgmspace.h>`
- `const byte data[] PROGMEM = {1, 2, 3, 4, 5};`
- `pgm_read_byte_near(data+0);`
- `Serial.print(F("Lots of text ..."));`
(F = macro ce face ca textul acesta să fie salvat în memoria de program (FLASH))

Portul serial

– Inițializare:

- `Serial.begin({300, 600, 1200, 2400, 4800, 9600, ...});`
- `while (! Serial) ; // așteaptă...`

– Scriere:

- `Serial.print(49); // '49'`
- `Serial.write(49); // '1'`

– Citire:

- `if (Serial.available()) { intVar = Serial.read(); }`

Despre ce am vorbit:

- Care sunt **elementele unui program** Arduino
(*constante, variabile, funcții*)
- Cum se utilizează **pinii**: *digitali, analogici, PWM*
(prin funcții vs control direct (prin regiștri))
- Cum se comandă **motoarele**
(sens, viteză, direcție)
- Cum se fac **temporizări**
- și comunicații **seriale**

