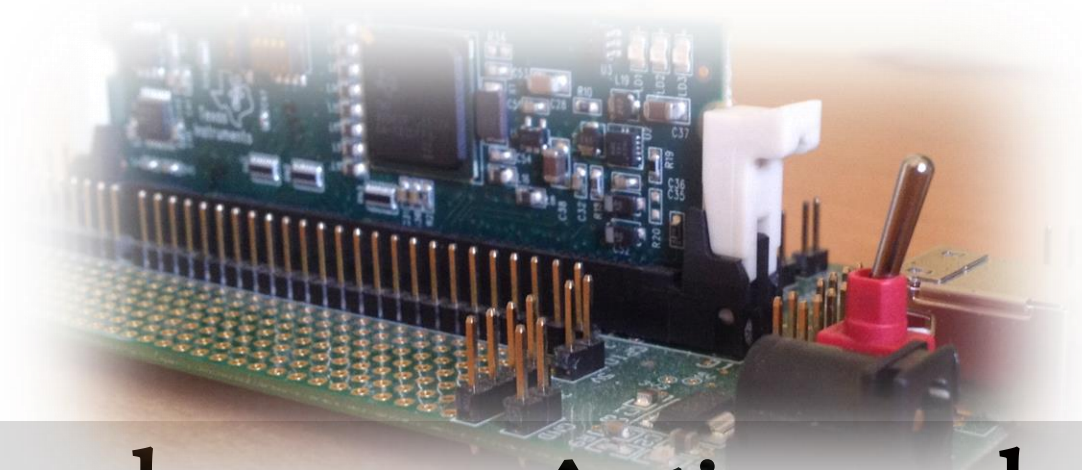
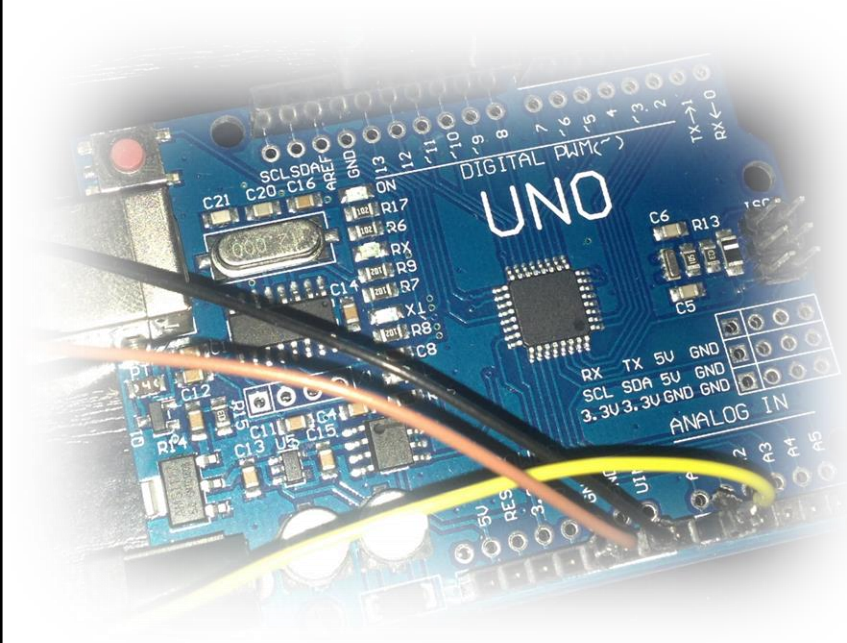
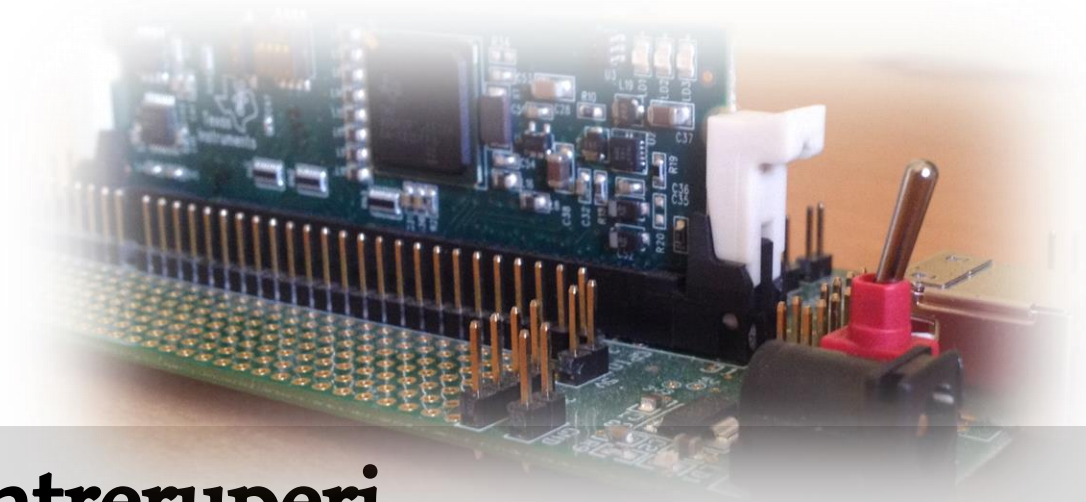
Sisteme de calcul în timp real



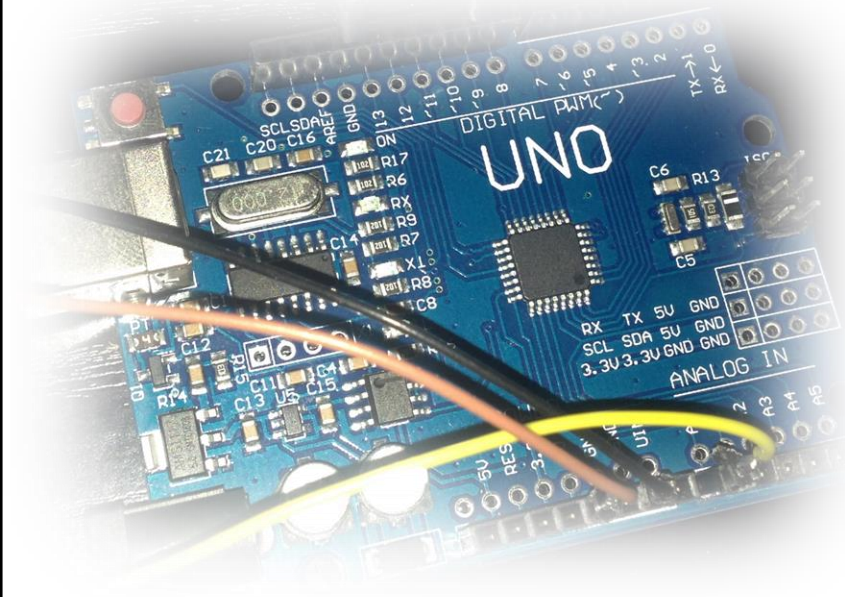
Întreruperi, sisteme de operare în timp real



Sisteme de calcul în timp real



Întreruperi



Întreruperi

- Ce este o întrerupere?
  - O „abatere” de la funcționarea normală, prestabilită a procesului;
  - Un fenomen, (de obicei), declanșat în mod non-determinist;
  - Are drept consecință suspendarea execuției programului;

- Care este rolul întreruperilor într-un proces?
  - ✓ De a permite execuția prioritară a unui proces critic (ex. sesizarea unui defect, și prevenția acestuia – decuplare la temperatură);
  - ✓ De a „concentra” puterea de calcul a sistemului dat, asupra unui proces critic sau cel puțin dependent de timp;
  - ✓ De a rezolva problemele de execuție în timp (ex. calculul vitezei în funcție de numărul de impulsuri furnizat de encoder, numărarea impulsurilor, calculul frecvenței unui semnal etc...);

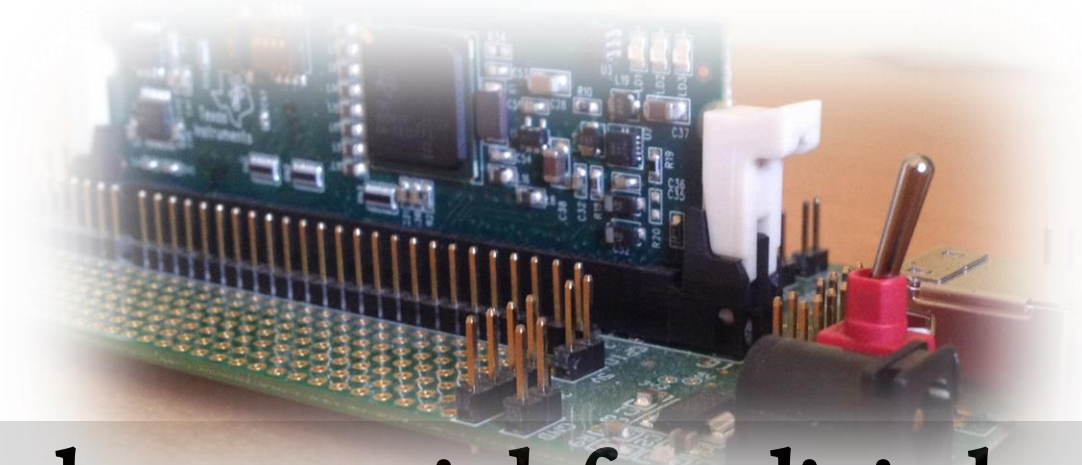
Întreruperi

- Tipuri de întreruperi des utilizate:
  - ✓Întreruperi declanșate pe cale fizică (eng. hard);
  - ✓Întreruperi declanșate pe cale logică (eng. soft);

- Elementele constitutive ale unei întreruperi:
  - ✓Rutina de prioritizare / tratare a întreruperilor;
  - ✓Vectorul surselor de întrerupere;

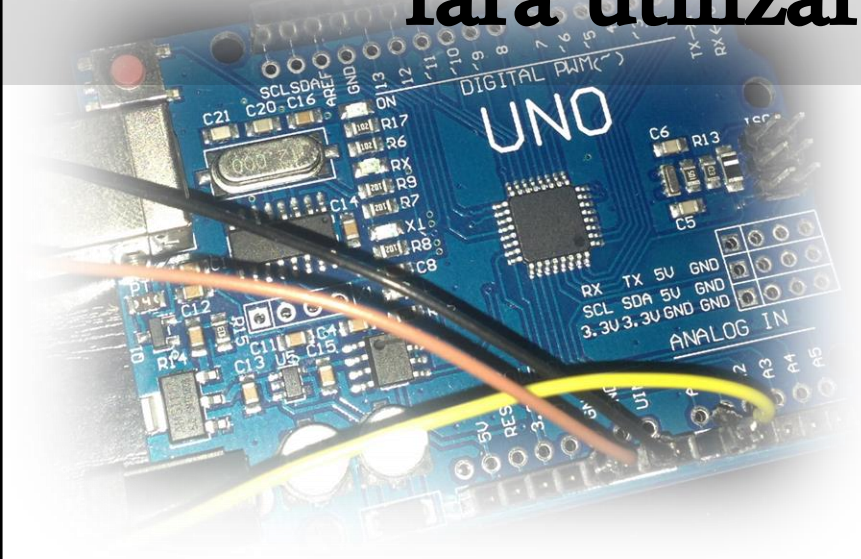
- Pentru procesoarele de tip AVR (ex. Arduino) există:
  - ✓Doi pini digitali care permit apelarea întreruperii pe cale fizică;
  - ✓În limbajul Wiring funcția „attachInterrupt()”;

Sisteme de calcul în timp real



Sesizarea şi semnalarea unui defect digital

fără utilizarea întreruperilor



Întreruperi

- Structura programului scris în limbaj Wiring (ArduinoIDE):

```
int senzor_detectie_defect = 2;
int led_verde = 8;
int led_rosu = 9;
```

Întreruperi

- Structura unui program scris în limbaj Wiring (ArduinoIDE):

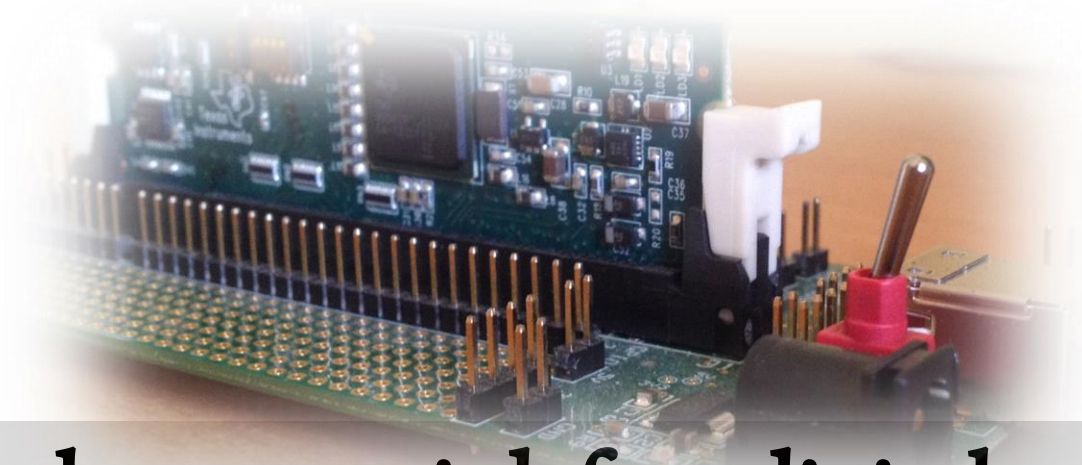
```
void setup() {
pinMode (led_verde, OUTPUT);
pinMode (led_rosu, OUTPUT);
pinMode (senzor_detectie_defect, INPUT);
}
```

Întreruperi

- Structura unui program scris în limbaj Wiring (ArduinoIDE):

```
void loop() {
int stare_defect = digitalRead(senzor_detectie_defect);
if (stare_defect == 0) {
digitalWrite(led_verde, HIGH);
digitalWrite(led_rosu, LOW); }
else {
digitalWrite(led_verde, LOW);
while(1) {
digitalWrite(led_rosu, HIGH);
delay(100);
digitalWrite(led_rosu, LOW);
delay(100);
 }
}
```

Sisteme de calcul în timp real



Sesizarea şi semnalarea unui defect digital utilizând întreruperile

Întreruperi

- Structura unui program scris în limbaj Wiring (ArduinoIDE):

```
const byte led_rosu = 9;

const byte senzor_detectie_defect = 2;

volatile byte stare_defect = LOW;
```

Întreruperi

- Structura unui program scris în limbaj Wiring (ArduinoIDE):

```
void setup() {
pinMode(led_rosu, OUTPUT);
pinMode(senzor_detectie_defect, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(senzor_detectie_defect)
, blink, CHANGE);
}
```

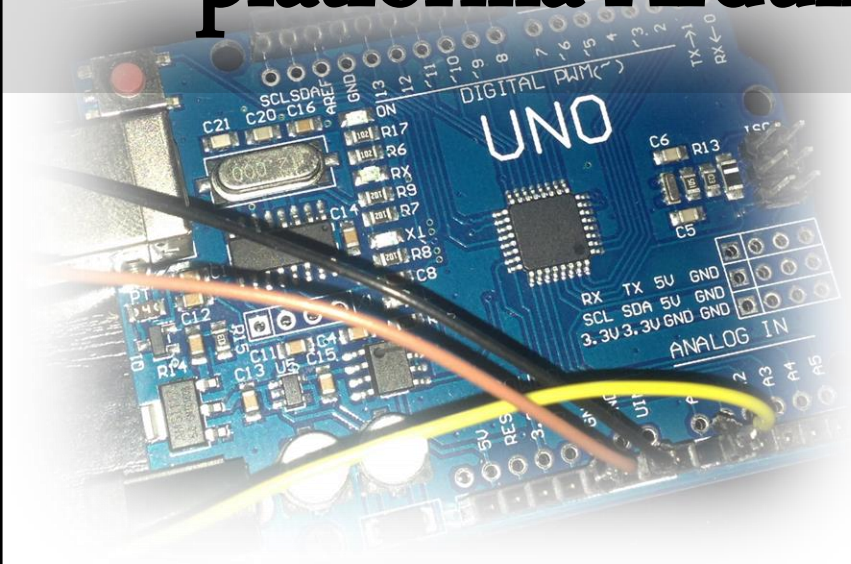
Întreruperi

- Structura unui program scris în limbaj Wiring (ArduinoIDE):

```
void loop() {
digitalWrite(led_rosu, stare_defect);
}
void blink()
{
stare_defect = !stare_defect;
}
```

Sisteme de calcul în timp real

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)



Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

- Scopurile utilizării sistemelor de operare în timp real:
  - ✓Realizarea mai multor operații cu execuție în mod paralel;
  - ✓Prioritizarea operațiilor;
  - ✓Optimizarea procesului de execuție al programului;
  - ✓Efectuarea operațiilor dependente de timpul de excecuție, la momentul potrivit;

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

- Structura unui program în timp real:

```
#include <Arduino_FreeRTOS.h>
Int led_rosu = 9;

// definirea sarcinilor de lucru:
void TaskSemnalizare( void *pvParameters );
void TaskCitireSenzor( void *pvParameters );
```

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

```
void setup() {
  Serial.begin(9600);
   while (!Serial) {
}
  // Crearea sarcinilor de lucru
  xTaskCreate(
 TaskSemnalizare
    ,  (const portCHAR *) "Semnalizare"
    ,  128  // Dimensiunea stivei
    ,  NULL
    ,  2  // Prioritate;
    ,  NULL );
  xTaskCreate(
    TaskCitireSenzor
    ,  (const portCHAR *) "CitireSenzor"
    ,  128  // Dimensiunea stivei
    ,  NULL
    ,  1  // Prioritate
    ,  NULL );
  // Rutina de prioritizare a sarcinilor de lucru va intra în funcțiune
}
```

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

```
void loop()
{

}
```

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

```
void TaskSemnalizare(void *pvParameters)
{
  (void) pvParameters;
  pinMode(led_rosu, OUTPUT);
  for (;;) //sarcina de lucru nu se întrerupe;
  {
    digitalWrite(led_rosu, HIGH);
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
    digitalWrite(led_rosu, LOW);
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
  }
}
```

Sistem de operare în timp real pentru platforma Arduino (ex. Free R.T.O.S.)

```
void TaskCitireSenzor(void *pvParameters)
{
  (void) pvParameters;
  for (;;)
  {
    int Senzor = analogRead(A0);
    Serial.println(Senzor);
    vTaskDelay(1);
  }
}
```