Tema de casă **Sisteme în timp real**
Echipa: Pupăză Dănuţ-Cătălin Semestrul 2, 2021-2022
e-mail: danut.pupaza@stud.acs.upb.ro

SISTEM IN TIMP REAL PENTRU CONTROLAREA SEMAFOARELOR UNEI INTERSECŢII

1 Formularea problemei

Obiectivul proiectului este de a eficientiza traficul vehiculelor dintr-o intersectie a doua strazi precum cea din figura de mai jos. Diferenta fata de o intersectie conventionala si eficientizarea se vor realiza folosind senzori de distanta pentru detectarea pozitiei urmatoarului vehicul care se apropie de intersectie de pe fiecare strada si controlarea semafoarelor in timp real in functie de traficul de pe cele doua strazi.

Datele de intrare in sistem sunt colectate de la cei doi senzori ultrasonici folositi pentru determinarea distantelor la care se afla urmatoarele vehicule ce vor veni in intersectie de pe fiecare strada.

Datele de iesire sunt semnale digitale catre cele doua semafoare si afisarea in serial monitor a distantelor de fiecare data cand sunt determinate.

Pentru implementarea sistemului in timp real am definit 6 task-uri cu aceeasi prioritate, am sincronizat task-urile folosind 7 semafoare binare astfel incat codul nu se ruleaza inutil.

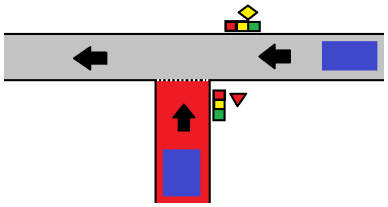

Figure 1: Intersectie abordata

2 Analiza problemei

- **Cerinte functionale ale sistemului:**
  - **Detectarea vehiculelor de pe cele doua strazi**
  - **Aprinderea/stingerea celor doua semafoare in functie de trafic**
  - **Afisarea valorilor distantei de la cei doi senzori in serial monitor atunci cand sunt determinate.**
- **Cerinte non-functionale ale sistemului:**
  - **Eroarea distantei determinate sa fie de maxim un centimetru**
  - **Daca nu este trafic, un vehicul sa astepte maxim doua secunde pana va avea verde.**
  - **Componente reutilizabile, nefiind deteriorate sau sudate/lipite**

  - **Sistem adaptabil in alte locatii prin schimbarea firelor la lungimea necesara**
  - **Cand distanta este determinata, sa se afiseze in serial monitor inaintea efectuarii oricarui alt task**

- **Modelul cazurilor de utilizare** - definire utilizatori:
  - **Operator Mentenanta**

- **Modelul cazurilor de utilizare** - definire cazuri de utilizare:

Caz utilizare:	Monitorizare sistem
Actori	Operator, Senzori
Descriere:	Colecteaza datele de la senzori pentru a putea fi vizualizate de operator
Tip:	Primar
Includes:	-
Extends:	Actionare semafoare

Table 1: Descrierea cazurilor de utilizare.

Caz utilizare:	Actionare semafoare
Actori	-
Descriere:	Actioneaza semafoarele
Tip:	Primar
Includes:	-
Extends:	-

Table 2: Descrierea cazurilor de utilizare.

Caz utilizare:	Intretinere sistem
Actori	Operator
Descriere:	Verificarea bunei conexiuni si functionari in vederea mentenantei
Tip:	Secundar
Includes:	-
Extends:	Update Software

Table 3: Descrierea cazurilor de utilizare.

Caz utilizare:	Update software
Actori	-
Descriere:	Modificarea/dezvoltarea codului unde se descopera aceasta nevoie
Tip:	Secundar
Includes:	-
Extends:	-

Table 4: Descrierea cazurilor de utilizare.

- **Modelul cazurilor de utilizare** - definire diagrama cazurilor de utilizare: Figura 2

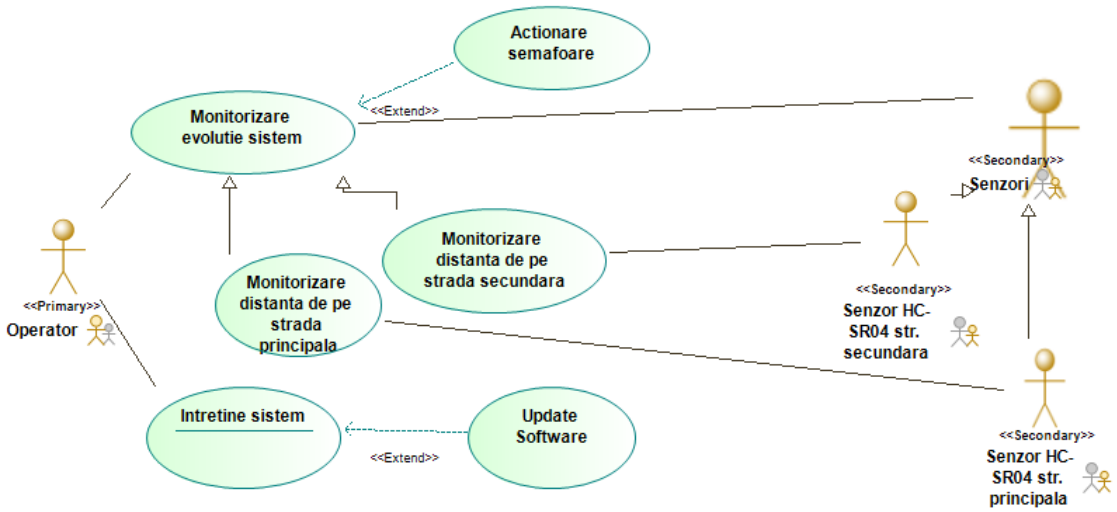

Figure 2: Diagrama cazurilor de utilizare

3 Definirea arhitecturii hardware - software

- **Prezentarea arhitecturii hardware** Arhitectura hardware este prezentata in figurile 3 si 4, in cadrul acesteia am folosit:
  - **Arduino Mega**
  - **2 senzori ultrasonici HC-SR04**
  - **6 rezistente de 100 ohmi**
  - **6 leduri**
  - **fire tata-tata**
  - **2 breadboard-uri**

- **Prezentarea modelului de programare pe arhitectura hardware aleasa**

  Modelul de programare este de programare cu procesare in timp real, avand codul structurat in taskuri, astfel fiind usor de urmarit/inteles, usor de modificat si se evita rularea inutila a codului prin sincronizare.

- **Prezentarea arhitecturii globale hardware - software**

  Pentru colectarea datelor de la cei doi senzori ultrasonici sunt definite doua task-uri care determina cele doua distante. Pentru afisarea in serial monitor a distantelor determinate sunt definite doua task-uri. Pentru controlul celor doua semafoare sunt definite doua task-uri. Toate taskurile sunt sincronizate prin intermediul a 7 semafoare care asigura buna functionare si evitarea rularii inutile a codului.

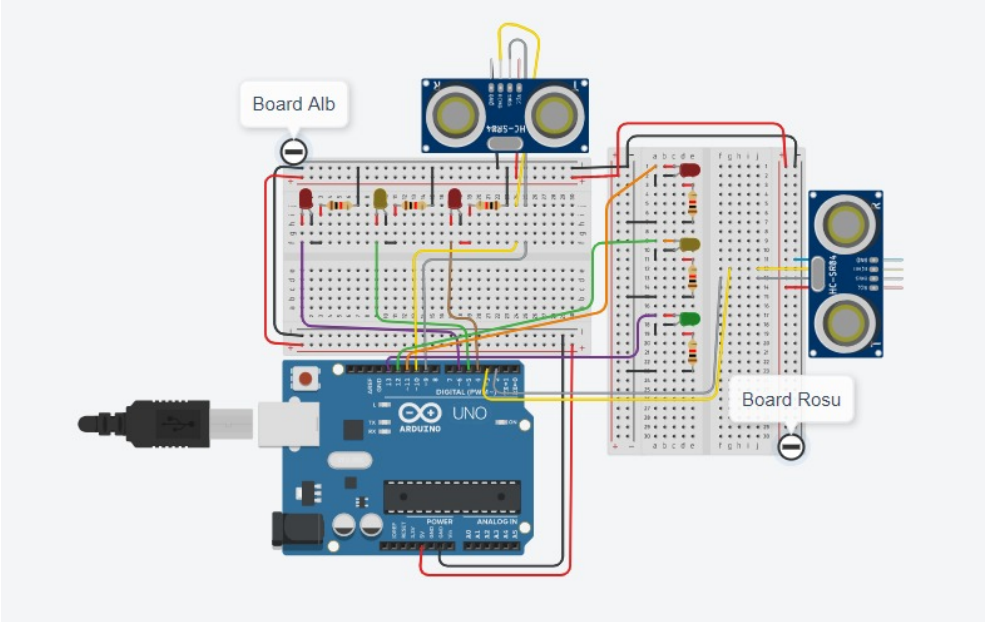

Figure 3: Imagine arhitectura hardware

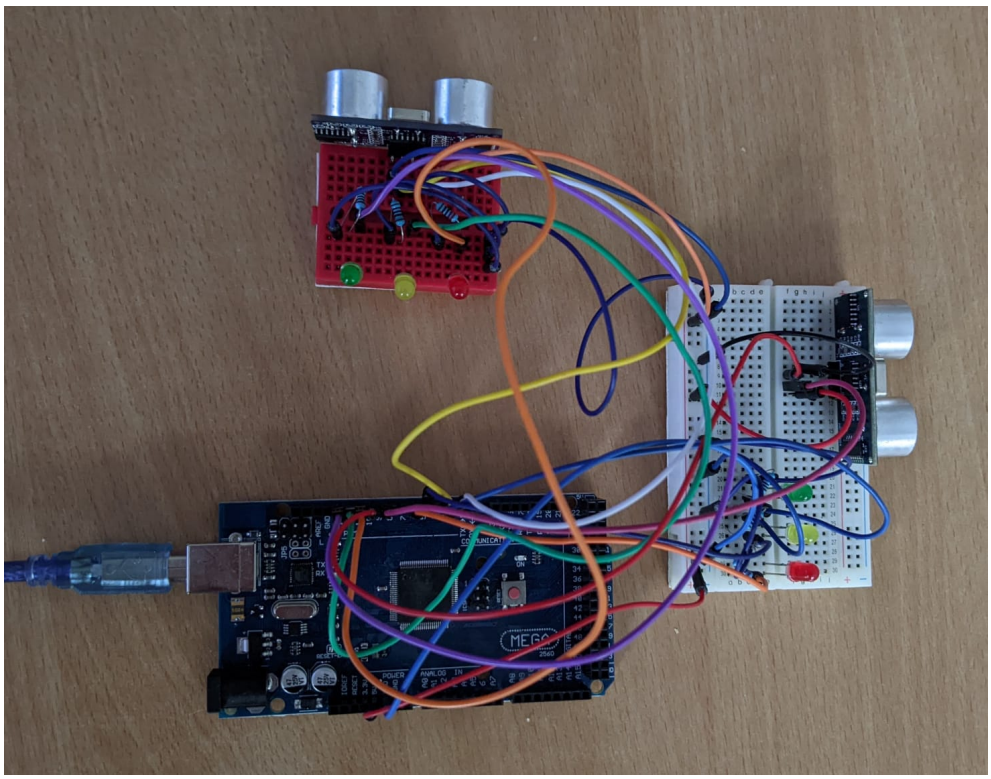

Figure 4: Imagine implementare hardware

- **Evidentiarea fluxului de date si informatie in aceasta arhitectura**

  Datele de intrare sunt reprezentate de datele analogice ale senzorilor ultrasonici ce urmeaza sa fie interpretate. Datele de iesire sunt semnale digitale catre cele 6 leduri ce formeaza cele doua semafoare.

- **Evidentierea limitelor de timp**

  Daca semafoarele sunt in varianta default, sistemul va reactiona la detectia unui vehicul la maxim 2.5 secunde, daca semafoarele sunt in cursul schimbarii prioritatii, sistemul va reactiona la maxim 10 secunde. Sistemul va reactiona la detectia unui vehicul la minim 0.1 secunde.

4 Definirea soluţiei de implementare software

- **Explicarea alegerii solutiei de implementare**

  In framework-ul Arduino, avem doar taskuri, nu avem procese sau fire de executie, astfel incat actiunile paralele, cu identitate functionala, vor rula in taskuri.

- **Definirea taskurilor de timp-real si non-real time care compun aplicatia**

  In aplicatie se regasesc 6 taskuri de timp real:

  - **TaskAnalogReadRosu** - Citeste datele de la senzorul de pe strada secundara(board-ul rosu)
  - **TaskAnalogReadAlb** - Citeste datele de la senzorul de pe strada principala(board-ul alb)
  - **TaskAfisareRosu** - Afiseaza datele de la senzorul de pe strada secundara(board-ul rosu)
  - **TaskAfisareAlb** - Afiseaza datele de la senzorul de pe strada principala(board-ul alb)
  - **TaskSemaphoreRosu** - Controleaza semaforul de pe strada secundara(board-ul rosu)
  - **TaskSemaphoreAlb** - Controleaza semaforul de pe strada principala(board-ul alb)

- **Sincronizarea si comunicarea taskurilor**

  Pentru sincronizarea celor 6 taskuri am folosit 7 semafoare binare. Semafoarele binare sunt ceea ce am avut nevoie deoarece ele sunt apelate in taskuri diferite, dar nu este vorba de o aplicatie tip "producator-consumator".

  Pentru intarzierile necesare in manipularea semafoarelor am folosit functia vTaskDelay pentru a nu bloca procesorul, alte taskuri putand fi executate.

- **Modul de lucru al sistemului**

  - **daca ambele strazi sunt libere sau ocupate atunci semaforul va fi verde pe strada principala deoarece ea are prioritate.**
  - **daca strada principala este libera si pe cea secundara se apropie un vehicul atunci semaforul de la strada principala se va face rosu si cel de la strada secundara se va face verde pentru un anumit interval de timp.**
  - **daca pe strada principala se apropie un vehicul el va avea prioritate de trecere deindata ce va fi posibil, indiferent de traficul de pe strada secundara.**

5 Implementarea software a soluţiei propuse

- **Prezentarea solutiei software, listing programe**

Listing 1: Sketch intersectie timp-real

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h> // Este inclusa biblioteca semhr.h pentru utilizarea semafoarelor

// Definire task-uri ----------------------------------------

void TaskAnalogReadAlb( void *pvParameters );
void TaskAnalogReadRosu( void *pvParameters );

void TaskSemaphoreAlb( void *pvParameters );
void TaskSemaphoreRosu( void *pvParameters );

void TaskAfisareAlb( void *pvParameters );
void TaskAfisareRosu( void *pvParameters );

//-------------------------------------------------------------
// Definim handle-uri pentru semafoare, necesare identificarii ulterioare
SemaphoreHandle_t sem1;
SemaphoreHandle_t sem2;
SemaphoreHandle_t sem3;
SemaphoreHandle_t sem4;
SemaphoreHandle_t sem5;
SemaphoreHandle_t sem6;
SemaphoreHandle_t sem7;
//-------------------------------------------------------------
const int trigPinAlb = 9;
const int echoPinAlb = 10;

const int trigPinRosu = 2;
const int echoPinRosu = 3;

const int boardRosuRosu = 11;
const int boardRosuGalben = 12;
const int boardRosuVerde = 13;

const int boardAlbRosu = 6;
const int boardAlbGalben = 5;
const int boardAlbVerde = 4;

float  distanceAlb, distanceRosu;

//-------------------------------------------------------------
// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  distanceAlb= 30;
  distanceRosu= 30;
```

```
  // Cream  semafoare binare
  sem1 = xSemaphoreCreateBinary();
  sem2 = xSemaphoreCreateBinary();
  sem3 = xSemaphoreCreateBinary();
  sem4 = xSemaphoreCreateBinary();
  sem5 = xSemaphoreCreateBinary();
  sem6 = xSemaphoreCreateBinary();
  sem7 = xSemaphoreCreateBinary();
// Semafoarele sunt default 0
xSemaphoreGive(sem1);



  // Sunt create taskuri, fiecare cu 128 de cuvinte de memorie rezervate si prioritate egala cu

  xTaskCreate(
    TaskAnalogReadAlb
    ,  "task2"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

   xTaskCreate(
    TaskAnalogReadRosu
    ,  "task3"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

   xTaskCreate(
    TaskSemaphoreAlb
    ,  "task4"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

    xTaskCreate(
    TaskSemaphoreRosu
    ,  "task5"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

   xTaskCreate(
    TaskAfisareAlb
    ,  "task6"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );
```

```
   xTaskCreate(
    TaskAfisareRosu
    ,  "task7"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

   vTaskStartScheduler(); // Este pornit algoritmul de planificare
}
//-----------------------------------------------------------------------
//--------------loop---------------------------------------------------
void loop()
{
  // Empty. Things are done in Tasks.
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/
//------------------------------------------------------------------------
void TaskAnalogReadRosu(void *pvParameters)  // This is a task.
{
  pinMode(trigPinRosu, OUTPUT);
  pinMode(echoPinRosu, INPUT);
  float durationRosu;
  while(1)
  {

    if (xSemaphoreTake(sem1, portMAX_DELAY) == pdTRUE) {

      Serial.println(" ");
      Serial.println("TaskAnalogReadRosu");
      digitalWrite(trigPinRosu, LOW);
      vTaskDelay(2);
      digitalWrite(trigPinRosu, HIGH);
      vTaskDelay(10);
      digitalWrite(trigPinRosu, LOW);

      durationRosu = pulseIn(echoPinRosu, HIGH);
      distanceRosu = (durationRosu*.0343)/2;
      xSemaphoreGive(sem2);
      vTaskDelay( 50 );

      if(distanceRosu<10){
      xSemaphoreGive(sem3);
      vTaskDelay( 50 );
      }
      else{
        xSemaphoreGive(sem1);
        vTaskDelay( 50 );
      }
    }
    vTaskDelay( 200 );
  }
```

```
}
//--------------------------------------------------------------------
void TaskAnalogReadAlb(void *pvParameters)  // This is a task.
{
  pinMode(trigPinAlb, OUTPUT);
  pinMode(echoPinAlb, INPUT);
  while(1)
  {
    float durationAlb;
    if (xSemaphoreTake(sem3, portMAX_DELAY) == pdTRUE) {

      Serial.println("TaskAnalogReadAlb");
      digitalWrite(trigPinAlb, LOW);
      vTaskDelay(2);
      digitalWrite(trigPinAlb, HIGH);
      vTaskDelay(10);
      digitalWrite(trigPinAlb, LOW);

      durationAlb = pulseIn(echoPinAlb, HIGH);
      distanceAlb = (durationAlb*.0343)/2;
      xSemaphoreGive(sem4);
      vTaskDelay( 50 );

      if(distanceAlb>10){
      xSemaphoreGive(sem5);
      vTaskDelay( 50 );
      }
      else{
        xSemaphoreGive(sem1);
        vTaskDelay( 50 );
      }

    }
    vTaskDelay( 100 );
  }
}


//--------------------------------------------------------------------
void TaskAfisareRosu(void *pvParameters)  // This is a task.
{

   while(1)
  {

    if (xSemaphoreTake(sem2, portMAX_DELAY) == pdTRUE) {
      Serial.println("TaskAfisareRosu");
      Serial.print("DistanceRosu: ");
      Serial.println(distanceRosu);
    }
    vTaskDelay( 100 );
  }
}

//---------------------------------------------------
```

```
void TaskAfisareAlb(void *pvParameters)  // This is a task.
{

   while(1)
  {

    if (xSemaphoreTake(sem4, portMAX_DELAY) == pdTRUE) {
      Serial.println("TaskAfisareAlb");
      Serial.print("DistanceAlb: ");
      Serial.println(distanceAlb);
    }
    vTaskDelay( 100 );
  }
}

//-------------------------------------------------------------
void TaskSemaphoreAlb(void *pvParameters){
  pinMode(boardAlbRosu, OUTPUT);
  pinMode(boardAlbGalben, OUTPUT);
  pinMode(boardAlbVerde, OUTPUT);

  digitalWrite(boardAlbRosu, LOW);
  digitalWrite(boardAlbGalben, LOW);
  digitalWrite(boardAlbVerde, HIGH);

  while (1)
  {

    if (xSemaphoreTake(sem5, portMAX_DELAY) == pdTRUE) {

      Serial.println("TaskSemaphoreAlbPartea1");
      digitalWrite(boardAlbRosu, LOW);
      digitalWrite(boardAlbGalben, HIGH);
      digitalWrite(boardAlbVerde, HIGH);
      vTaskDelay(100);
      digitalWrite(boardAlbRosu, HIGH);
      digitalWrite(boardAlbGalben, LOW);
      digitalWrite(boardAlbVerde, LOW);
      xSemaphoreGive(sem6);
      vTaskDelay( 50 );
    }

    if (xSemaphoreTake(sem7, portMAX_DELAY) == pdTRUE) {
      Serial.println("TaskSemaphoreAlbPartea2");
      digitalWrite(boardAlbRosu, LOW);
      digitalWrite(boardAlbGalben, LOW);
      digitalWrite(boardAlbVerde, HIGH);

      xSemaphoreGive(sem1);
      vTaskDelay( 50 );
    }
  vTaskDelay( 100 );
  }
}
```

```
//-------------------------------------------------------------
void TaskSemaphoreRosu(void *pvParameters){
  pinMode(boardRosuRosu, OUTPUT);
  pinMode(boardRosuGalben, OUTPUT);
  pinMode(boardRosuVerde, OUTPUT);

  digitalWrite(boardRosuRosu, HIGH);
  digitalWrite(boardRosuGalben, LOW);
  digitalWrite(boardRosuVerde, LOW);

  while (1)
  {

    if (xSemaphoreTake(sem6, portMAX_DELAY) == pdTRUE) {
      Serial.println("TaskSemaphoreRosu");
      digitalWrite(boardRosuRosu, LOW);
      digitalWrite(boardRosuGalben, LOW);
      digitalWrite(boardRosuVerde, HIGH);
      vTaskDelay(500);
      digitalWrite(boardRosuGalben, HIGH);
      vTaskDelay(100);
      digitalWrite(boardRosuRosu, HIGH);
      digitalWrite(boardRosuGalben, LOW);
      digitalWrite(boardRosuVerde, LOW);
      xSemaphoreGive(sem7);
      vTaskDelay( 50 );
    }
    vTaskDelay( 100 );
  }
}
//-----------------------------------
```

- **Prezentatea rezultatelor executiei aplicatiei software** In figura de mai jos(Figura 5) se poate vedea ce se afiseaza in serial monitor in decursul rularii programului. Am incarcat un videoclip pe Youtube unde arat cum functioneaza sistemul(https://youtu.be/tDlY5khqfcs).

```
TaskAnalogReadRosu
TaskAfisareRosu
DistanceRosu: 25.55

TaskAnalogReadRosu
TaskAfisareRosu
DistanceRosu: 4.99
TaskAnalogReadAlb
TaskAfisareAlb
DistanceAlb: 6.74

TaskAnalogReadRosu
TaskAfisareRosu
DistanceRosu: 7.01
TaskAnalogReadAlb
TaskAfisareAlb
DistanceAlb: 28.85
TaskSemaphoreAlbPartea1
TaskSemaphoreRosu
TaskSemaphoreAlbPartea2
```

Figure 5: Testare

6 Testarea aplicaţiei şi validarea soluţiei propuse

Dupa testarea aplicatiei, s-a verificat faptul ca executia codului este corecta, predictibila, determinista. Pentru buna intelegere a rularii codului am adaugat in fiecare task afisarea numelui lui. Am incarcat un videoclip pe Youtube unde arat cum functioneaza sistemul(https://youtu.be/tDlY5khqfcs).

- **Ce anume poate influenta functionarea sistemului in timp real pentru care ati scris aplicatia?**
  - **slabirea conexiunilor firelor**
  - **uzarea senzorilor**
  - **conditiile de mediu precum umiditatea**
- **Testarea si validarea aplicatiei**

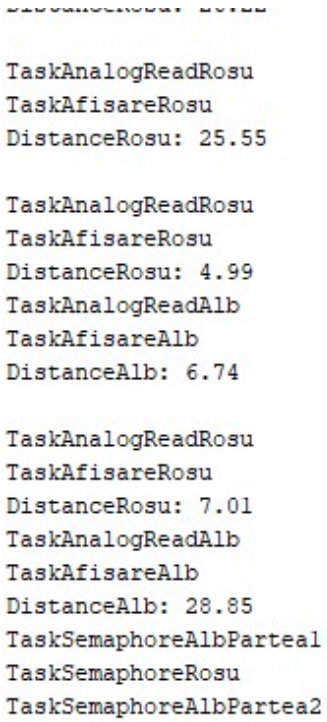

Figure 6: Testare

References

[1] Richard Barry *Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide*. Real Time Engineers Ltd. 2016.

[2] Amazon Web Services *FreeRTOS V10.0.0 Reference Manual*. 2017 Amazon.com