Universitatea Tehnică „Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare

Sisteme de Control Încorporate

Proiectarea sistemelor încorporate

Proiect

Studenți:

Bodnariuc Florin

Slivinschi Ionela

Grupa 1A, anul II, master, SCI

Cuprins

[1. Introducere 3](#_Toc187921288)

1. [Scopul Proiectului 3](#_Toc187921289)
2. [Importanța Proiectului 3](#_Toc187921290)
3. [Domeniul de Aplicare 3](#_Toc187921291)

[2. Analiza cerințelor 5](#_Toc187921292)

[3. Proiectarea sistemului 6](#_Toc187921293)

1. [Prezentare plăcuță BIG8051 6](#_Toc187921294)
2. [Design software 8](#_Toc187921295)

[4. Implementare 10](#_Toc187921296)

1. [Implementarea codului pentru simularea receiver-ului 10](#_Toc187921297)
2. [Implementarea codului pentru simularea transmitter-ului 11](#_Toc187921298)

[5. Testare și validare 12](#_Toc187921299)

[5. Concluzie 15](#_Toc187921300)

# Introducere

## Scopul Proiectului

Scopul acestui proiect este implementarea unui sistem de comunicare între două plăcuțe de dezvoltare Big8051, utilizând o rețea CAN simulată pentru a implementa protocolul Unified Diagnostic Services (UDS). Acesta reprezintă un protocol standardizat utilizat în principal în domeniul automotive, pentru diagnosticarea și gestionarea componentelor electronice ale vehiculelor. Proiectul urmărește să creeze un cadru funcțional în care să se poată utiliza protocolul UDS pentru realizarea a cinci servicii generice, fiecare având rolul de a simula interacțiuni specifice unui sistem de diagnostic.

## Importanța Proiectului

Proiectul este deosebit de important în contextul dezvoltării tehnologiilor moderne din domeniul automotive și al sistemelor embedded. Implementarea protocolului UDS pe o rețea CAN simulează condițiile de funcționare ale unui vehicul real, permițând testarea și validarea funcționalității serviciilor de diagnostic în condiții controlate. De asemenea, folosirea plăcuțelor Big8051 în combinație cu o rețea CAN ajută la aprofundarea cunoștințelor legate de protocoale de comunicație, microcontrolere și arhitecturi de sisteme embedded, fiind astfel un instrument didactic valoros.

Prin dezvoltarea celor cinci servicii generice, proiectul contribuie la creșterea înțelegerii modului în care se pot implementa și gestiona procesele de diagnosticare și întreținere a vehiculelor, având un impact direct în domenii precum dezvoltarea software-ului pentru automobile, mentenanța acestora și integrarea tehnologiilor IoT în infrastructuri vehiculare.

## Domeniul de Aplicare

Domeniul de aplicare al acestui proiect este în principal cel al industriei automotive, unde protocoalele de diagnosticare standardizate, cum este UDS, sunt esențiale pentru gestionarea eficientă a sistemelor electronice ale vehiculului. Rețeaua CAN este un element fundamental în arhitectura vehiculară, iar implementarea acestui proiect poate fi extinsă și în alte domenii, cum ar fi automatizarea industrială, vehiculele autonome, sau aplicațiile care necesită o comunicare rapidă și fiabilă între modulele unui sistem complex.

# Analiza cerințelor

1. **Comunicarea prin protocolul CAN a celor două plăcuțe de dezvoltare**:  
Stabilirea unei rețele de comunicație CAN pentru schimbul de date între cele două plăcuțe Big8051.

2. **Simularea protocolului UDS prin implementarea a cinci servicii și requesturilor pentru acestea**:  
Implementarea protocolului UDS pentru a simula cereri de diagnosticare și comunicare între plăcuțe.

3. **Implementarea răspunsului pozitiv și negative, cu NRC specific**:

* **Răspuns pozitiv**: Atunci când requestul este procesat cu succes de ECU, se va transmite un răspuns pozitiv care confirmă succesul operațiunii. Raspunsul pozitiv este construit din ID-ul serviciului accesat plus 40 in baza hexa. Pentru serviciul 1016, raspunsul pozitiv va fi 5016.
* **Răspuns negativ**: În caz de eroare sau imposibilitatea procesării cererii (de exemplu, serviciul solicitat nu este disponibil sau cererea este incorectă), se va transmite un răspuns negativ, care va include un cod de eroare specific. Răspunsul negative este un mesaj pe 3 bytes: 7F SID NRC.

4. **Implementarea requesturilor cu feedback vizual și auditiv**:

* **Afisarea pe LCD**: Afișarea serviciului accesat.
* **Utilizarea ieșirilor LED**: Simularea implementării serviciului prin pornire/oprire LED.
* **Utilizarea buzzer-ului**: Semnalizarea evenimentelor prin buzzer.

# Proiectarea sistemului

## Prezentare plăcuță BIG8051

Plăcuța BIG8051 este o platformă de dezvoltare bazată pe microcontrolerul **8051**, care face parte din familia microcontrolerelor dezvoltate de Intel. Acest microcontroler este bine cunoscut pentru fiabilitatea, performanța și versatilitatea sa în implementarea de soluții embedded. Plăcuța Big8051 extinde aceste capabilități, oferind dezvoltatorilor o bază solidă pentru prototipurile și aplicațiile hardware.

**Caracteristici Cheie ale Plăcuței BIG8051**

* **Microcontroler 8051**:
  + Procesor pe 8 biți.
  + Frecvență de operare de până la 12 MHz (variază în funcție de implementare).
  + Arhitectură Harvard cu memorie separată pentru date și instrucțiuni.
* **Memorie**:
  + **RAM**: Memorie internă disponibilă pentru stocarea variabilelor și a datelor temporare.
  + **ROM/EPROM**: Memorie pentru stocarea programului (firmware) la pornirea sistemului.
* **I/O**:
  + Multiple porturi I/O pentru conectarea la diverse periferice.
  + Conectivitate pentru implementarea de interfețe externe (LED-uri, senzori, butoane, motoare etc.).
* **Comunicare**:
  + **CAN (Controller Area Network)**: Suport pentru protocolul CAN, utilizat în aplicațiile auto și industriale pentru comunicarea între unități de control.
  + **USART**: Interfață pentru comunicații seriale, utilizată pentru transmiterea și recepționarea datelor prin UART.
  + **I2C/SPI**: Suport pentru protocoale de comunicare serie pentru periferice externe.
* **Afisaj și control**:
  + Ecran LCD pentru vizualizarea datelor și statusului aplicației.
  + Butoane și tastaturi pentru input de date.
  + LED-uri și buzzer pentru feedback vizual și auditiv.

## Design software

A diagram of a computer program

Description automatically generated

Figura 1. Diagrama de flux pentru simulare Receiver

A black and white diagram

Description automatically generated

Figura 2. Diagrama de flux pentru simulare Transmitter

# Implementare

## Implementarea codului pentru simularea receiver-ului

Partea cheie este configurarea celor două obiecte CAN:

* Obiectul de transmisie (TX) cu ID-ul CAN 0.
* Obiectul de recepție (RX) cu ID-ul CAN 1.

CAN\_Init\_Object(1, CANID\_STD, 0x000, 1, MSGOBJ\_TX); // MasgObj = 1, STD, CANID = 0, DLC = 1, TX

CAN\_Init\_Object(2, CANID\_STD, 0x001, 1, MSGOBJ\_RX); // MasgObj = 2, STD, CANID = 1, DLC = 1, RX

Recepționarea și procesarea mesajelor CAN este următorul pas. Când sunt primite date noi pe CAN, acestea sunt verificate pentru a vedea dacă corespund unui serviciu definit într-o listă (ServicesList).

* Dacă mesajul primit corespunde unui serviciu valid, se efectuează o acțiune specifică și se trimite un raspuns pozitiv
* Dacă serviciul nu este implementat, se trimite un răspuns negativ.

CAN\_MsgTX.Date.Byte[0] = (CAN\_MsgRX.Date.Byte[0] + 0x40);

CAN\_Transmit(1,0,&CAN\_MsgTX);

if((unsigned int)CAN\_MsgRX.Date.Byte[0] == 0x10) {

// Acțiuni pentru serviciul 0x10

LCD\_PutStr(0, 0, "Request");

LCD\_PutStr(1, 0, "Service 10");

buzzer(2000); // Sunet buzzer

}

else if((unsigned int)CAN\_MsgRX.Date.Byte[0] == 0x2E) {

// Acțiuni pentru serviciul 0x2E

LCD\_PutStr(0, 0, "Request");

LCD\_PutStr(1, 0, "Service 2E");

led0 = 1; // Aprinde LED-ul

}

// (Alte condiții pentru alte servicii)

## Implementarea codului pentru simularea transmitter-ului

* **Inițializare (BIG8051\_Init)**:
* Dezactivează Timerul Watchdog (WDT)
* Inițializează ceasul sistemului
* Inițializează registrele SFR (Special Function Registers)
* Inițializează LCD-ul (Display cu Cristale Lichide)
* Inițializează tastatura
* Inițializează Timerul 0 pentru a genera întreruperi la fiecare 1 milisecundă
* Inițializează UART0 (Universal Asynchronous Receiver/Transmitter) pentru comunicația serială la o viteză de 9600 baud
* Inițializează controlerul CAN pentru operațiuni la 1 Mbit/s
* Activează întreruperile globale

void BIG8051\_Init(void)

{

    WDT\_Disable();               // dezactiveaza WDT

    SYSCLK\_Init();               // initializeaza si selecteaza oscilatorul ales in osc.h

    PORT\_Init();                 // conecteaza perifericele la pini si stabileste tipul pinilor

    LCD\_Init();                  // 2 linii, display ON, cursor OFF, pozitia initiala (0,0)

    KEYB\_Init();                 // initializeaza driver tastatura matriciala

    Timer0\_Init();               // initializeaza Timer 0 - intreruperi la 1 ms

    UART0\_Init(EIGHT\_BIT, 9600); // initializeaza UART0 - conectare terminal la UART-USB

    CAN\_Init();                  // initializeaza controlerul CAN, 1 Mbit/sec

    EA = 1;                      // validare globala intreruperi

}

* **Funcții:**
* wait(t, in\_us): Această funcție introduce o întârziere pentru o durată specificată (t) în milisecunde (dacă in\_us este fals) sau microsecunde (dacă in\_us este adevărat).

/\*\*

\* WAIT

 \* @param t: timp de asteptare (intarziere)

 \* @param in\_us: timpul in microsecunde daca true, in milisecunde daca false

\*/

void wait(int t, bool in\_us)

{

    (in\_us == 0) ? Delay\_us(t) : Delay(t);  // intarziere in milisecunde sau microsecunde

}

* print\_menu(): Această funcție afișează un meniu de opțiuni pentru ca utilizatorul să le selecteze, inclusiv Controlul Sesiunii de Diagnosticare, Scriere Date prin Identificator, Citire Date prin Identificator, Ștergere Informații de Diagnosticare și Control I/O prin Identificator.

/\*\*

\* PRINT MENU

 \* printeaza meniul principal

\*/

void print\_menu()

{

    UART0\_Putstr("\r\n");

    UART0\_Putstr("Please select option from menu:           \r\n");

    UART0\_Putstr("A. Diagnostic Session Control:\t\t0x10.   \r\n");

    UART0\_Putstr("B. Write Data by Identifier:      0x2E.   \r\n");

    UART0\_Putstr("C. Read Data by Identifier:       0x22.   \r\n");

    UART0\_Putstr("D. Clear Diagnostic Information:  0x14.   \r\n");

    UART0\_Putstr("#. I/O Control by Identifier:     0x2F.   \r\n");

    UART0\_Putstr("\r\n                                      \r\n");

}

* option(opt): Această funcție ia o opțiune selectată de utilizator (opt) și transmite mesajul CAN corespunzător pe baza opțiunii selectate. De asemenea, afișează opțiunea selectată pe LCD. Opțiunile valide sunt:
  + 'A' (Controlul Sesiunii de Diagnosticare - 0x10),
  + 'B' (Scriere Date prin Identificator - 0x2E),
  + 'C' (Citire Date prin Identificator - 0x22),
  + 'D' (Ștergere Informații de Diagnosticare - 0x14) ,
  + '#' (Control I/O prin Identificator - 0x2F).
  + Orice altă opțiune este considerată invalidă.

/\*\*

\* OPTION

 \* @param opt: in functie de opt se va efectua setul corespunzator de instructiuni

 \*  + daca optiunea este valida:

 \*      - se va scrie in consola optiunea selectata, octetul ce va fi trimis pe CAN

 \*      - se va scrie pe LCD octetul ce va fi trimis pe CAN

 \*      - se va trimite pe CAN octetul corespunzator optiunii selectate

 \*  + daca optiunea este invalida:

 \*      - se va avertiza utilizatorul ca a selectat o optiune invalida

 \*      - se va scrie pe LCD octetul ce va fi trimis pe CAN ce reprtezinta optiune invalida (0xFF)

 \*      - se va trimite pe CAN octetul 0xFF

\*/

void option(unsigned char opt)

{

    if (opt == DSC)

    {

        UART0\_Putstr("\t- Diagnostic Session Control:   0x10. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0x10"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0x10;    // Pune in buffer octetul de transmis

    }

    else if (opt == WDI)

    {

        UART0\_Putstr("\t- Write Data by Identifier:     0x2E. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0x2E"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0x2E;    // Pune in buffer octetul de transmis

    }

    else if (opt == RDI)

    {

        UART0\_Putstr("\t- Read Data by Identifier:      0x22. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0x22"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0x22;    // Pune in buffer octetul de transmis

    }

    else if (opt == CDI)

    {

        UART0\_Putstr("\t- Clear Diagnostic Information: 0x14. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0x14"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0x14;    // Pune in buffer octetul de transmis

    }

    else if (opt == IOCI)

    {

        UART0\_Putstr("\t- I/O Control by Identifier:        0x2F. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0x2F"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0x2F;    // Pune in buffer octetul de transmis

    }

    else

    {

        UART0\_Putstr("\t- Invalid option. \r\n");

        LCD\_PutStr(0, 0, "Trimis: 0xFF"); // afiseaza pe LCD octetul trimis

        CAN\_MsgTX.Date.Byte[0] = 0xFF;    // Pune in buffer octetul de transmis

    }

}

* buzzer(t, f, in\_us): Această funcție generează un semnal pătrat pe pinul buzzerului la o frecvență specificată (f) pentru o durată specificată (t) în milisecunde (dacă in\_us este fals) sau microsecunde (dacă in\_us este adevărat).

/\*\*

\* BUZZER

 \* @param t: timp de redare a sunetului prin buzzer

 \* @param f: frecventa sunetului

 \* @param in\_us: timpul oferit este in microsecunde?

 \* Reda prin buzzer un sunet de frecventa 'f' pentru perioada 't' de timp

\*/

void buzzer(int t, double f, bool in\_us)

{

    double period = 0;      // perioada de redare a sunetului: period = 1/f. [s]

    double cycles = 0;      // numar de cicli (perioade): cycles = t \* f.

    unsigned int iter = 0;  // iterator

    double s = 10 ^ 3;      // baza de timp SI: din ms in s.

    if (f == 0.0)

    {

        wait(t, in\_us);     // liniste (pauza)

    }

    else

    {

        if (in\_us)          // timp in microsecunde

            s \*= s;         // 1s = 10^3ms = 10^6us (10^3 \* 10^3)

        period = s / f;

        cycles = (t \* f) / s;

        for (iter = 0; iter < cycles; iter++)

        {

            // jumi-juma buzzer on-off (PWM 50%)

            //  \_\_    \_\_    \_\_    \_\_

            //    |\_\_|  |\_\_|  |\_\_|  |\_\_|

            buz = 1;

            wait((int)(period / 2), in\_us); // jumatate de perioada on

            buz = 0;

            wait((int)(period / 2), in\_us); // jumatate de perioada off

        }

    }

}

* sad\_song(): Această funcție redă un cântec trist simplu pe buzzer folosind frecvențe și durate predefinite.

/\*\*

\* SAD SONG

 \* avertizare ca s-a ales un scenariu negativ

\*/

void sad\_song()

{

    int bpms = 1600; // bit per ms

    buzzer(bpms / 2, G, USE\_US);

    buzzer(bpms / 2 ^ 3, 0.0, USE\_US);

    buzzer(bpms / 2, Gb, USE\_US);

    buzzer(bpms / 2 ^ 3, 0.0, USE\_US);

    buzzer(bpms / 2, F, USE\_US);

    buzzer(bpms / 2 ^ 3, 0.0, USE\_US);

    buzzer(bpms / 2 ^ 3, E, USE\_US);

    buzzer(bpms / 2 ^ 5, 0.0, USE\_US);

    buzzer(bpms / 2 ^ 3, E, USE\_US);

    buzzer(bpms / 2 ^ 5, 0.0, USE\_US);

    buzzer(bpms / 2 ^ 3, E, USE\_US);

    buzzer(bpms / 2 ^ 5, 0.0, USE\_US);

    buzzer(bpms / 2 ^ 3, E, USE\_US);

    buzzer(bpms / 2 ^ 5, 0.0, USE\_US);

}

* received(): Această funcție verifică dacă au fost primite mesaje CAN și le afișează pe terminalul UART și pe LCD. De asemenea, redă cântecul trist dacă mesajul primit nu se potrivește cu răspunsul așteptat.

/\*\*

\* RECEIVED

 \* se apeleaza cand s-a primit un mesaj pe CAN

 \* - Daca mesajul primit este cu 0x40 mai mare decat cel trimis: s-a receptionat un raspuns pozitiv;

\*/

void received(){

    UART0\_Putstr("\r\nRaspuns primit: ");       //  Afiseaza in consola ca s-a primit un mesaj nou

    if ((CAN\_MsgTX.Date.Byte[0] + 0x40) == CAN\_MsgRX.Date.Byte[0])

    {

        UART0\_PutByteHex(CAN\_MsgRX.Date.Byte[0]);   // afiseaza pe terminal octetul primit

        UART0\_Putstr("\t- OK");                     // OK

    }

    else

    {                                               //  0x7F 0xZZ 0x11 - NOK

        UART0\_PutByteHex(0x7F);                     //

        UART0\_PutByteHex(CAN\_MsgRX.Date.Byte[0]);   //

        UART0\_PutByteHex(0x11);                     //

        UART0\_Putstr("\t - NOK!");                  //

        // play something sad :)

        sad\_song();

    }

    UART0\_Putstr("\r\n\r\n ");

    LCD\_PutStr(1, 0, "Primit: ");

    LCD\_PutByte(1, LCD\_col, CAN\_MsgRX.Date.Byte[0]); // afiseaza pe LCD octetul primit

}

* **Programul principal:**
* Inițializează variabile pentru stocarea intrării utilizatorului, contoare de buclă și obiecte de mesaje CAN.
* Apeleză BIG8051\_Init pentru a inițializa resursele hardware și software.
* Configurează două obiecte de mesaje CAN: unul pentru transmisie (Tx) și unul pentru recepție (Rx).
* Afișează meniul pe terminalul UART.

    unsigned char tasta, counter = 0, iter = 0, buf[16], here = 0;

    unsigned int tx = 0;

    BIG8051\_Init(); // Initializare resurse HW & SW BIG8051

    CAN\_Init\_Object(1, CANID\_STD, 0x001, 1, MSGOBJ\_TX); // MasgObj = 1, STD, CANID = 0, DLC = 1, TX

    CAN\_Init\_Object(2, CANID\_STD, 0x000, 1, MSGOBJ\_RX); // MasgObj = 2, STD, CANID = 1, DLC = 1, RX

    print\_menu();

* Intră într-o buclă infinită:
* Verifică dacă există o intrare de la tastatură.
* Dacă se apasă o tastă, apelează funcția option pentru a gestiona selecția și a transmite mesajul CAN corespunzător.

        if (tasta = KEYB\_Input())

        {

            option(tasta);

            CAN\_Transmit(1, 0, &CAN\_MsgTX);

        }

* Verifică dacă a fost primit un nou mesaj CAN.
* Dacă se primește un nou mesaj, apelează funcția received pentru a procesa și afișa mesajul.
* Afișează din nou meniul după o întârziere.

        if (CAN\_MsgRX.NewDat)

        {                         // S-au primit date noi?  - da :)

            CAN\_MsgRX.NewDat = 0; // daca da, sterge indicatorul date noi

            received();

            wait(10^3, false);

            print\_menu();

        }

În general, acest cod este un program simplu pentru un microcontroler BIG8051 care permite unui utilizator să trimită și să primească mesaje CAN în scopuri de diagnosticare. Oferă o interfață bazată pe meniu pentru selectarea diferitelor tipuri de mesaje CAN și afișează mesajele primite pe terminalul UART și pe LCD. Funcția sad\_song adaugă o notă de umor în caz de comportament neașteptat.

# 5. Testare și validare

**1. Comunicarea prin protocolul CAN a celor două plăcuțe de dezvoltare**

**Test:**  
Verificarea stabilității comunicării între cele două plăcuțe prin CAN, transmiterea și recepționarea datelor.

**Pași de testare:**

* Se apasă o tastă de la tastatura (KEYB\_Input()).
* Datele sunt transmise de la o plăcuță la alta prin CAN.
* Mesajele trimise sunt afișate pe LCD și pe UART.

**Rezultate așteptate:**

* Mesajul transmis este corect afisat pe LCD și UART.
* Datele sunt corect transmise prin CAN, iar a doua plăcuță le primește corect.

**Rezultate obținute:**

* Testul a fost finalizat cu succes, mesajele au fost transmise corect între plăcuțe, iar LCD și UART au afișat mesajele corespunzătoare.

**2. Simularea protocolului UDS prin implementarea a cinci servicii și requesturilor pentru acestea**

**Test:**  
Verificarea implementării serviciilor UDS și gestionarea corectă a cererilor de diagnosticare.

**Pași de testare:**

* Se trimite un request de diagnosticare pentru fiecare din cele cinci servicii implementate (0x10, 0x2E, 0x22, 0x2F, 0x14).
* Fiecare serviciu va răspunde cu un mesaj adecvat (pozitiv sau negativ).

**Rezultate așteptate:**

* Fiecare serviciu trimite un răspuns corespunzător.
* Serviciile 0x10, 0x2E, 0x22, 0x2F, 0x14 sunt implementate corect, iar feedback-ul este dat conform cerințelor.

**Rezultate obținute:**

* Serviciile au fost implementate corect, iar răspunsurile au fost corecte. LCD-ul și UART au afisat corect statusul pentru fiecare serviciu, iar LED-urile și buzzer-ul au semnalizat feedback-ul corect.

**3. Implementarea răspunsului pozitiv și negativ, cu NRC specific**

**Test:**  
Testarea răspunsurilor pozitive și negative pentru request-uri, inclusiv gestionarea erorilor (NRC).

**Pași de testare:**

* Se trimite un request valid pentru serviciul 0x10, 0x2E, 0x22, 0x2F, 0x14, iar răspunsul pozitiv trebuie să fie generat cu succes (ID-ul serviciului + 0x40).
* Se trimite un request invalid (serviciu neimplementat) și se așteaptă un răspuns negativ cu un cod de eroare specific.

**Rezultate așteptate:**

* Răspunsurile pozitive sunt corect formate (de exemplu, 0x50 pentru serviciul 0x10).
* Răspunsurile negative includ codul de eroare corect (de exemplu, 0x7F 0x12 0x11 pentru serviciu neimplementat).

**Rezultate obținute:**

* Răspunsurile pozitive și negative au fost corect implementate. LCD și UART au afisat corect răspunsurile primite, iar mesajele de eroare au fost procesate și trimise corespunzător.

**4. Implementarea request-urilor cu feedback vizual și auditiv**

**Test:**  
Verificarea feedback-ului vizual și auditiv pentru răspunsurile la request-uri.

**Pași de testare:**

* Se trimite un request valid și se verifică:
  + **LCD**: Afișarea serviciului accesat.
  + **LED**: Activarea LED-ului P2.0 la accesarea serviciului 0x2E.
  + **Buzzer**: Activare buzzer pentru accesarea serviciului 0x10
* Se trimite un request invalid și se verifică:
  + **UART**: Afișarea raspunsului negativ
  + **Buzzer**: Activarea buzzer-ului pentru a semnaliza eroarea.

**Rezultate așteptate:**

* Feedback vizual pe LCD și LED corespunzător serviciului solicitat
* Buzzer-ul va semnaliza erorile sau evenimentele critice corespunzător.

**Rezultate obținute:**

* LCD a afisat corect serviciul accesat sau mesajul de eroare.
* LED-ul P2.0 a fost aprins
* Buzzer-ul a semnalizat corect erorile cu un semnal sonor scurt, confirmând funcționarea completă a feedback-ului auditiv.

# Concluzie

Toate funcționalitățile proiectului au fost testate cu succes. Comunicarea CAN între plăcuțe este stabilă, serviciile UDS sunt implementate corect, iar răspunsurile pozitive și negative sunt procesate și afișate corect. Feedback-ul vizual și auditiv a fost gestionat corespunzător, oferind utilizatorului o interfață clară pentru interpretarea stării sistemului.