Universitatea Tehnică “Gheorghe Asachi” din Iaşi

Facultatea de Automatică şi Calculatoare

Domeniul: Ingineria sistemelor

Specializarea: Automatică şi Informatică Aplicată

Anul universitar 2023-2024

Proiect la disciplina

Comunicaţii în sisteme de conducere

Tema 3

Student:	**Grupa:**	**Rol (C,T,R,U)**
Butacu Ionel-Cătălin	**1404A**	**C - MS**
Vacaru Alexandru-Ioan	**1404A**	**C - J**
Horobet Iulia Nicoleta	**1404A**	**T**
Secrieru Stefănel-Marian	**1404A**	**R**
Munteanu Alexandru	**1404A**	**U**

Cuprins:

1. Tema proiectului

1.1. Specificaţiile de proiectare

Folosind Microsistemul BIG8051 să se implementeze un protocol de comunicaţie serială pentru 5 noduri conectate în rețea, având următoarele caracteristici:

1	Port	UART1 (RS-485 - folosind un adaptor extern TTL – RS-485)
2	Parametri comunicatie	115200, 9, N, 1
3	Formatul mesajului	Adresă HW nod destinaţie
		Adresă HW nod sursă
		Tipul mesajului: 0 sau 1 Sursa mesajului (numai mesaje de tip 1)
		Destinația finală a mesajului (numai mesaje de tip 1)
		Lungime date (numai mesaje de tip 1)
		Date (numai mesaje de tip 1)
		Cod detectare erori: LRC (+)
4	Codificare mesaj	ASCII hex

1.2. Formatul mesajelor:

Binar:

Adresa hardware nod destinatie	Adresa hardware nod sursa	Tip mesaj	Adresa nod sursa mesaj	Adresa nod destinatie mesaj	Lungime	Date	Suma de control
1 octet	1 octet	1 octet	1 octet	1 octet	1 octet	Lungime	1 octet

ASCII hex:

Adresa hardware nod destinatie	Adresa hardware nod sursa	Tip mesaj	Adresa nod sursa mesaj	Adresa nod destinatie mesaj	Lungime	Date	Suma de control	Sfarsit mesaj
1 octet	2 octeti	2 octeti	2 octeti	2 octeti	2 octeti	2 X Lungime	2 octeti	0Dh,0Ah

1.3. Particularități de implementare

- Câmpul *adresă hardware nod destinaţie* va conţine adresa nodului care trebuie să recepţioneze mesajul.
- Câmpul *adresă hardware nod sursă* va conţine adresa nodului care efectiv transmite mesajul.
- Câmpul *tip mesaj* poate fi, după caz:
    - 0:
        - mesaj de interogare transmis de master către un nod slave sau de slave către master, în lipsa altui mesaj util;
        - mesaj de tip jeton, trimis de către un nod pentru transferul jetonului către un alt nod.
    - 1 – mesaj de date – mesaj care conţine un text ASCII introdus de utilizator de la interfața de intrare a unui nod.
- Câmpul *adresă nod sursă mesaj* va indica sursa mesajului;
- Câmpul *adresă nod destinaţie mesaj* va indica destinația finală a mesajului:
    - un mesaj de tip 1, transmis de un nod slave către un alt node slave, va fi retransmis de către nodul master către nodul slave destinație finală;
    - la protocolul bazat pe jeton poate lipsi (este identic cu adresa destinație hardware).
- Câmpul *lungime* reprezintă numărul de octeţi al câmpului de date;
- Octetul din câmpul *suma de control* va fi calculat ca suma modulo 2 a tuturor octeţilor din câmpurile anterioare.
- Mesajul va fi codificat în binar și transmis octet cu octet portului serial UART0, conectat la un adaptor extern TTL - RS-485.

2. Scurtă prezentare a resurselor HW şi SW utilizate

Microsistemul BIG8051 – MikroElektronika.

Microsistemul oferă o platformă flexibilă pentru programarea şi dezvoltarea de aplicaţii cu microcontrolere 8051. Numeroase module, precum butoane, LED-uri, afişaje LCD, un difuzor (buzzer) piezoelectric, interfeţe şi controlere de comunicaţii, convertoare analog / numerice şi numeric / analogice etc., permit utilizatorilor să testeze cu uşurinţă funcţionarea aplicaţiilor direct pe placa de dezvoltare.

O vedere de ansamblu a microsistemului este prezentată în *Fig.1.1,* iar o scurtă descriere a blocurilor funcţionale este dată în *Fig.1.2*

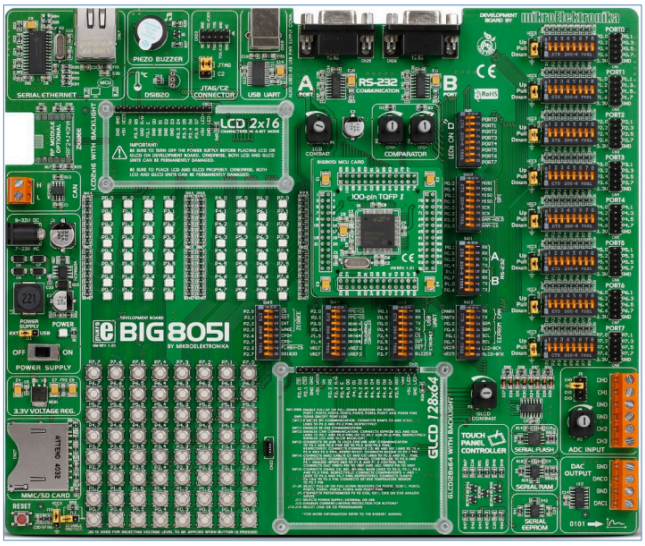


Fig 1.1

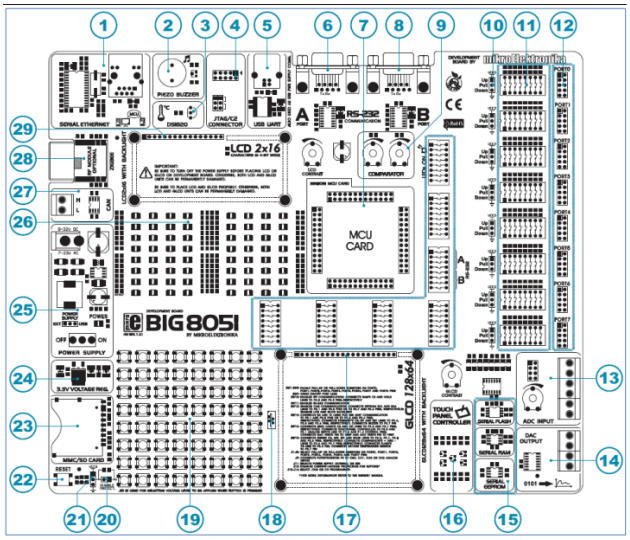


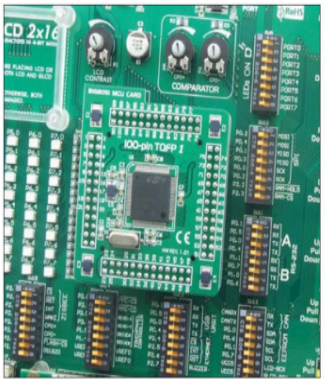
Fig 1.2

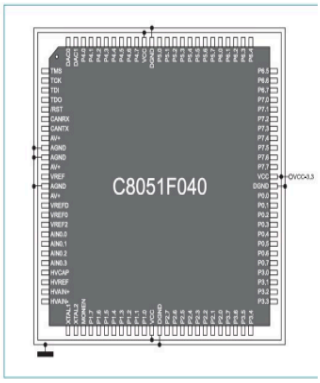
1. Modul interfaţă Ethernet
2. Buzzer piezoelectric
3. Conector pentru senzorul de temperatură
4. Conector pentru programator/debugger
5. Modul USB-UART şi alimentare USB
6. Modul interfaţă RS-232 - A
7. Conector card MCU
8. Modul interfaţă RS-232 - B
9. Comparator
10. Jumperi selectare nivel de fixare VCC | GND
11. DIP switch conectare pini la rezistoare de fixare
12. Conectori porturi I/E şi alimentare
13. Intrări convertor A/D
14. Ieşiri convertor D/A
15. Modul memorii seriale externe
16. Controler touch panel
17. Conector pentru afişaj grafic (GLDC)
18. Conector pentru touch panel
19. Butoane cu revenire conectate la pinii porturilor
20. Jumper scurtcircuitare rezistor de protecţie
21. Jumper selectare stare logică buton apăsat
22. Buton RESET
23. Conector card de memorie MMC/SD
24. Regulator de tensiune 3,3V c.c.
25. Conector şi selecţie sursă de alimentare
26. LED-uri pentru afişarea stării porturilor
27. Modul interfaţă CAN
28. Conector interfaţă ZigBee
29. Conector afişaj LCD alfanumeric

Sistemul de dezvoltare BIG8051 este echipat cu un *Microcontroler C8051F040* de la Silicon Laboratories, într-o capsulă TQFP cu 100 de pini, lipită pe un card MCU montat pe placă (Fig.1.3).

Caracteristici tehnice ale microcontrolerului C8051F040:

- microcontroler MSP (Mixed-Signal Processor) – procesare semnale digitale și analogice;
- compatibil software cu 8051 - același set de instrucțiuni;
- arhitectură pipeline - execută 70% din setul de instrucțiuni în 1 sau 2 perioade de tact;
- oscilator programabil intern, calibrat, 3÷24,5MHz
- execută până la 25 MIPS cu oscilator de 2 MHz;
- RAM intern de date de 4352 octeți (4KB 256);
- 64 KB memorie Flash, programabilă în sistem (ISP) în sectoare de 512 octeți;

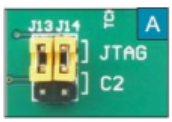




- 8 porturi de I/E configurabile;
- set extins de numărătoare – temporizatoare (Timers & PCA – Programmable Counter Array)
- convertoare A/D și D/A
- set extins de periferice de comunicații:
- 2 porturi seriale UART și un port SPI;
- Bosch Controller Area Network (CAN 2.0B);

Programarea microcontrolerului și depanarea programului:

Microcontrolerul C8051F040 este amplasat pe cardul MCU, care este montat pe sistemul de dezvoltare și poate fi programat cu un adaptor USB DEBUG de la Silicon Laboratories. Acesta se conectează ca în Fig.1.4 și funcționează atât ca debugger JTAG, cât și ca debugger C2. Pentru microcontrolerul C8051F040 se utilizează varianta JTAG, cu ajutorul J13 și J14.



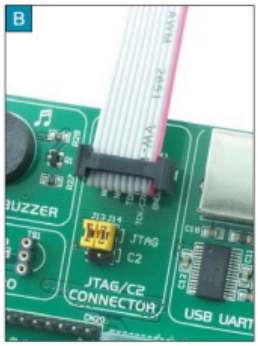


Fig.1.4 Conectarea adaptorului USB DEBUG

Organizarea memoriei microcontrolerului C8051F040:

Aceasta este similară cu cea de la 8051 standard (arhitectura Harvard modificată). Există două spații de memorie separate: de program și de date(Fig.1.5). Ele partajează același spațiu de adrese, dar sunt accesate cu tipuri diferite de instrucțiuni. CIP-51 are implementat un spațiu de 256 de adrese de memorie RAM internă și de 64KB de memorie internă program de tip Flash.

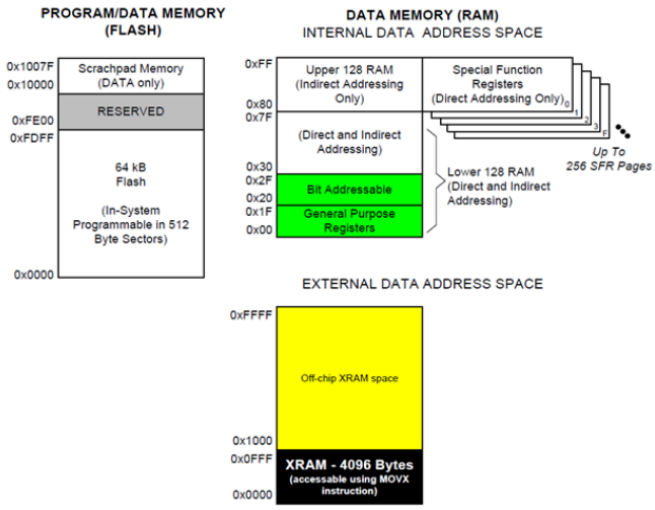


Fig.1.5 Organizarea memoriei

Memoria program:

Microcontrolerul C8051F040 are implementat un spațiu de 64Kb de memorie program, sub forma unui bloc continuu 0000h-FFFFh. Ultimii 512 octeți (FE00h-FFFFh) sunt rezervați din fabrică și nu trebuie folosiți.

Memoria program este înscrisă cu ajutorul programatorului JTAG și poate fi de regulă doar citită din programul utilizatorului:

- când sunt extrase instrucțiunile pentru a fi executate;
- cu ajutorul unor instrucțiuni MOVC, pentru acces la date constante, amplasate în memoria program.

Memoria de date:

Spațiul de memorie internă de date de 256 de octeți RAM este organizat la fel ca la 8052. Zona inferioară de 128 octeți este adresabilă atât direct, cât și indirect și aici regăsim cele 4 bancuri de registre de uz general (00h-1Fh), în care sunt mapate registrele R0-R7, precum și un spațiu de 16 locații adresabile atât pe octet, cât și pe bit (adrese de octet 20h-3Fh și adrese de bit 00h-7Fh).

Zona superioară de memorie, adresabilă numai indirect și utilizată de obicei pentru stivă, ocupă același spațiu de adrese cu zona registrelor cu funcții speciale (SFR), care este adresată direct. Diferitele spații de memorie ale microcontrolerelor din familia 8051 pot fi folosite pentru amplasarea codului și datelor folosind opțiuni ale mediului integrat de dezvoltare Keil µVision (pentru a stabili reguli implicite) sau extensii ale limbajului C pentru specificarea explicită a spațiului de memorie în care va fi amplasat codul și variabilele programului.

Astfel, codul aplicației este amplasat întotdeauna, implicit în memoria program, acest lucru nu mai trebuie specificat explicit.

Dezvoltarea si integrarea software a proiectului

Keil PK51 este o platformă integrată (IDE – Integrated Development Environment) pentru dezvoltarea aplicaţiilor pentru microcontrolere 8051 şi compatibile. Aceasta include un manager de proiecte (Keil µVision3), un editor de text sursă, compilatorul C51, asamblorul A51, editorul de legături L51, gestionarul de biblioteci LIB51, un simulator/depanator, precum şi alte programe utilitare necesare pentru generarea codului executabil.

PK51 dispune de o interfaţă grafică Windows, iar managerul de proiecte (Keil µVision3) ţine evidenţa tuturor fişierelor unei aplicaţii şi lansează automat în execuţie toate programele utilitare necesare pentru realizarea operaţiilor solicitate de utilizator.

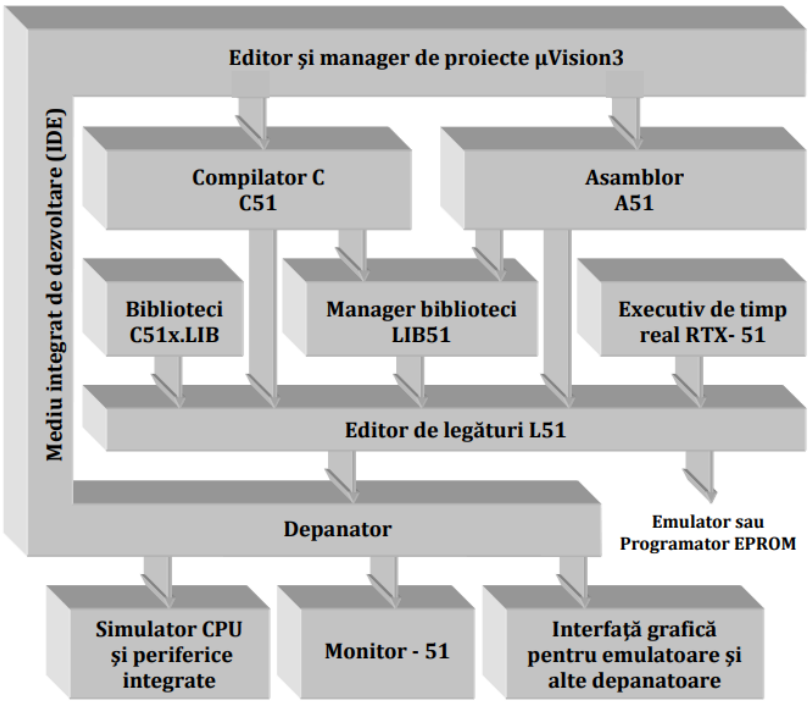


Fig.1.6 Structura PK51 - ciclul de dezvoltare al unei aplicaţii

Prelucrarea fişierelor sursă: compilarea, asamblarea şi editarea legăturilor – operaţii realizate sub controlul managerului de proiecte Keil µVision3, pune în evidenţă erorile de sintaxă introduse la scrierea programului, care pot fi corectate off-line.

În schimb, verificarea comportării on-line a aplicaţiei şi eliminarea erorilor logice, de concepţie, se face la momentul execuţiei, prin depanare fie:

- folosind *simulatorul integrat în mediul de dezvoltare;*
- *direct pe sistemul hardware* pentru care a fost proiectată aplicaţia, folosind monitorul de comunicaţie Monitor-51, un emulator (ICE-In-Circuit Emulator) sau un debugger JTAG;
- folosind alte programe de depanare simbolică la nivel de cod sursă.

Simulatorul integrat permite testarea modulelor software şi a perifericelor interne ale microcontrolerelor a căror funcţionare este reprodusă prin simulare. De asemenea, se poate simula apariţia unor stimuli externi, absolut necesari pentru a pune în evidenţă cât mai multe din situaţiile care pot să apară la o execuţie reală pe sistemul hardware.

Pentru microsistemul BIG8051 utilizat în proiect s-a folosit un dispozitiv *Silicon Labs USB Debug Adapter,* conectat la portul JTAG al microcontrolerului C8051F040 și la un port USB al calculatorului gazdă pe care se execută Keil µVision3.

3. Descrierea protocoalelor utilizate

3.1 Protocolul Master-Slave

Protocolul se bazeaza pe tehnica de control al accesului la mediu (master & slave). Această tehnică prevede existența unui nod numit master (arbitru central) care să controleze momentele în care celelalte noduri din rețea pot transmite sau recepționa date. Dreptul de acces este acordat de către master , pe rând, în mod ciclic, fiecărui nod slave, printr-un mesaj adresat acestuia. Fiecare mesaj pe care un nod slave îl va transmite va trece mai întai prin nodul master, apoi acesta va decide destinația finală, în funcție de adresa indicată în mesaj .

Există 2 tipuri de mesaje :

- un mesaj de interogare, care va fi transmis de master, pe rând, fiecărui slave pentru a determina dacă acesta are un mesaj în așteptare. Acesta nu va conține câmpurile de lungime și de date. În cazul în care nu are mesaj, slave-ul va transmite la rândul său masterului un mesaj de interogare.

- al doilea tip de mesaj este cel care conține date de transmis. El poate apărea ca răspuns al slave-ului la o interogare din partea masterului și trimis mai departe slave-ului destinație sau direct de la master, dacă acesta a primit o comandă de transmisie din partea utilizatorului (de la tastatura proprie).

Descrierea funcționării programului pentru nodul master:

Secvenţa de iniţializare

- afişaj LCD;
- coprocesor tastatură;
- port serial;
- variabilele programului (de exemplu: adresa HW a nodului).

Afişare parametri program:

Linia 1: Master/Slave, Adresa, COM0/COM2, ASCII/Binar.
Linia 2: RxM – nod - ultimul mesaj de date recepţionat cu succes şi care nod l-a transmis.

Afişare meniu comenzi:

Linia 3: 1 – TxM 2 – Stare.
Linia 4: utilizată de comenzile 1 şi 2.

Nodul Master este cel care coordonează transmisia mesajelor în rețea. Alege un slave inițial (slave 1), apoi va intra în buclă și va urma setul următor de instrucțiuni.

Verifică dacă are un mesaj de transmis nodului respectiv (nod[slave].full==0 sau1). În cazul în care este un mesaj de trimis, masterul îl va trimite la adresa indicată prin funcția TxMesaj(nod destinație) ; în caz contrar trimite un mesaj de interogare (cod funcție 0).

Apelează funcția RxMesaj(slave) cu adresa nodului de la care se așteaptă un răspuns trimisă ca parametru (așteptarea se va face cu timeout o secundă , moment în care se va trece la următorul slave). În cazul în care mesajul receptat este de tip 0, se va trece la următorul nod slave. Pentru un mesaj de tip 1 adresat nodului master, acesta îl va afișa pe ecran (linia a doua a LCD-ului). Dacă vine un mesaj de tip 1 pentru destinat altui slave, îl va memora pentru o transmitere ulterioară.

După terminarea funcției RxMesaj se verifică dacă s-a apăsat o tastă. Pentru valoarea 1 (TxM) va solicita adresa și datele care urmează a fi trimise slave-ului , iar pentru valoarea 2 se va verifica și afișa starea bufferului nodului slave indicat.

Descrierea funcționării programului pentru nodul slave:

Nodul Slave aşteaptă în buclă infinită un mesaj de la master sau o comandă de la tastatură. De la master există posibilitatea de a primii un mesaj de tip 0 sau de tip 1. În cazul primirii unui mesaj de tip 1 atunci acesta este afişat pe linia 2, iar în cazul mesajului de tip 1 este transmis către master. Dacă nodul slave nu are un mesaj de tip 1 de transmis atunci transmite către master un mesaj de tip 0. Dacă apare o comandă de la tastatură, tratează comanda. Pentru comanda TxM: solicit adresa destinaţie, preia adresa destinaţie, solicit mesajul, preia şi memorează mesajul. Pentru comanda stare mesaje: solicită adresa destinaţie, preia adresa destinaţie, afişează dacă există sau nu un mesaj care aşteaptă să fie transmis.

3.2 Protocolul bazat pe Jeton

Protocolul de tip jeton este o metodă utilizată în sistemele de comunicații pentru a gestiona accesul la mediul de transmisie partajat, prevenind coliziunile de date. Protocolul funcționează prin circularea unui jeton unic între nodurile rețelei, astfel numai nodul care deține jetonul are permisiunea de a transmite date la un moment specific. După ce transmisia este completă, jetonul este trecut la următorul nod.

Acest proces continuă într-o manieră ciclică, asigurând că toate nodurile au șanse egale de a comunica, eliminând posibilele conflicte de acces simultan la canalul de comunicare. Protocolul utilizează un proces riguros, de generare, regenerare și detectie pierdere a jetonului, asigurând funcționarea corectă a rețelei și evitand blocarea rețelei.

La inițializarea rețelei, primul nod intrat/adaugat in rețea, *genereaza jetonul* inițial și îl introduce în rețea. *Regenerarea jetonului* are loc în cazul în care jetonul original este detectat de către un nod ca fiind pierdut (nodul respectiv ce deținea jetonul s-a blocat/este picat). Astfel, se inițiază procesul de regenerare, regenerand un nou jeton și îl transmite mai departe următorului nod în rețea. Acest lucru previne blocajele în rețea și asigură continuitatea comunicării.

Prin aceste mecanisme de detecție și regenerare a jetonului și a datelor, protocolul asigură integritatea și fiabilitatea comunicării în rețelele de tip jeton.

4. Interfața I/E și conectarea la mediul de comunicație (nivelul fizic)

Microcontrolerul C8051F040 dispune de 8 porturi de I/E de 8 biți (P0-P7). Toate porturile fiind adresabile și pe bit, iar pinii porturilor suportă și nivele TTL (5V)
Toți pinii porturilor pot fi configurați în **3 moduri de funcționare:**

- operare normală (push-pull);
- drenă în gol (ȘI cablat);
- cu fixare slabă a nivelului la VDD (weak pullup)

Operare normală (push – pull) – comanda pentru pin (0 sau 1) vine pe linia PORT-OUTPUT și, dacă driverul de ieșire este validat (/PORT-OUTENABLE = 0), comandă cele două tranzistoare în opoziție, iar pe pin este forțat nivelul dorit: 0 sau 1.

- ✔ În acest mod, pinul este configurat ca ieșire și nu poate fi conectat la un alt pin de ieșire, ci doar la pini de intrare. Starea pinului de ieșire poate fi citită prin program, dacă pinul nu este selectat ca intrare analogică.

Operare cu drenă în gol – se obține dacă /PORT-OUTENABLE = 1: tranzistorul de sus este blocat continuu și cel de jos este deschis numai când comanda pe linia PORT-OUTPUT este 0. Astfel, microcontrolerul poate forța pinul doar pe nivel 0; când se comandă nivel 1 pinul flotează (HZ), dacă /WEAK-PULLUP = 1.

- ✔ În acest mod, pinul poate funcționa ca ieșire cu drena în gol și poate fi conectat la o altă ieșire cu drena în gol, precum și la pini de intrare. Starea pinului poate fi citită prin program dacă

pinul nu este selectat ca intrare analogică. Acest mod este folosit pentru conectarea mai multor ieșiri împreună, la aceeași linie fizică.

- ✔ În acest mod pinul poate funcționa și ca intrare, cu condiția ca PORT-OUTPUT să fie fixat la 1.

Operarea cu fixare slabă la VDD – când /PORT-OUTENABLE = 1, ambele tranzistoare sunt blocate și nivelul pe pin poate fi fixat slab la VDD prin /WEAK-PULLUP = 0 (se conectează printr-o rezistență internă de cca. 100KΩ la Vcc), dacă pinul nu este folosit ca intrare analogică.

- ✔ În acest mod starea pinului, care poate fi citită prin program, este 1 dacă pinul nu este comandat pe 0, în ultimul caz rezistorul de tip pull-up este automat dezactivat.
- ✔ Rezistorii de tip pull-up pot fi toți dezactivați, indiferent de nivelul comandat pe linia PORT-OUTPUT, dacă XBR2.7 = 1 (bitul Weak Pull-up Disable).

Descrierea funcțiilor interfeței cu utilizatorul

Funcțiile ce gestionează interfața cu utilizatorul pentru un sistem MS/JT cu noduri ce comunică prin mesaje sunt definite prin funcțiile: UserIO, Afisare_meniu, Afisare_mesaj, Error, TERM_Input. Ele oferă mecanisme pentru transmiterea și recepția mesajelor, afișarea stării sistemului și tratarea erorilor

- ➢ **TERM_Input(void)**

Această funcție citește o tasta de la terminal (UART0). Ca si funcționare:

- Salvează pagina curentă a registrului SFR.
- Setează SFRPAGE la LEGACY_PAGE.
- Verifică dacă a fost primită o tasta (RI0) și o returnează.
- Restaurează SFRPAGE.

- ➢ ***UserIO(void)***

Aceasta este funcția principală de interfață cu utilizatorul, ce gestionează input-ul și output-ul programului printr-un comutator switch pentru a schimba între diferite stări ale sistemului (STARE_IO).

Variabile Locale:

- tasta: tasta apăsată de utilizator.
- cmd: comanda selectată de utilizator.
- dest: adresa nodului destinatar.
- lng: lungimea mesajului.

Funcționare:

- Comutator pe Stări (STARE_IO):
    - Stare 0:
        - 1: Transmiterea unui mesaj (cmd = 1).
        - 2: Afișarea stării unui nod (cmd = 2).
    - Stare 1:
        - Dacă cmd == 1 și tasta este între '0' și '4' (exclusiv adresa proprie):
        - Setează dest și verifică dacă bufferul nodului este plin.
        - Configurează adresa hardware și adresele sursă/destinație.
        - Afișează mesajul cerut de la utilizator și trece la STARE_IO = 2.
        - Dacă cmd == 2 și dest este între '0' și '4':
        - Afișează starea bufferului (plin sau gol).
        - Revine la STARE_IO = 0 și afișează meniul.
    - Stare 2:
        - Dacă tasta este validă și lng este mai mic decât NR_CHAR_MAX, adaugă tasta în buffer.
        - Dacă nu, finalizează mesajul, marchează bufferul ca plin și revine la STARE_IO = 0.

➢ **Afisare_meniu(void)**

Această funcție afișează meniul inițial pe LCD și trimite mesaje la UART.

Funcționare:

- Setează AFISARE = 1.
- Afișează adresa nodului și tipul nodului (Master/Slave, Jeton/NoJet).
- Afișează parametrii specifici temei (ASC/BIN).
- Afișează meniul de comenzi: "1-TxM 2-Stare :>".

➢ **Afisare_mesaj(void)**

Această funcție afișează mesajul din bufferul de recepție al nodului curent.

Funcționare:

- Verifică dacă există un mesaj în buffer (retea[ADR_NOD].full).
- Afișează adresa nodului sursă și mesajul caracter cu caracter.
- Marchează bufferul ca gol după afișarea mesajului.

➢ **Error(char *ptr)**

Afișează un mesaj de eroare pe LCD și UART dacă AFISARE este activ.

Funcționare:

- Trimite mesajul de eroare la UART și LCD.

5. Programul principal

Inițierea resurselor programului

WDT_Disable();

- ➢ oprește timerul watchdog pentru a preveni resetările neintenționate ale microcontrolerului.

SYSCLK_Init();

- ➢ inițializează și selectează oscilatorul ales in osc.h

UART1_Init(NINE_BIT, BAUDRATE_COM);

- ➢ inițializare UART1 - port comunicatie (TxD la P1.0 si RxD la P1.1)

UART1_TxRxEN (1,1);

- ➢ configurarea UART1 pentru transmisie și recepție cu setările specificate.

PORT_Init ();

- ➢ configurează porturile pentru a conecta perifericele (UART0,UART1) și stabilește tipul pinilor.

LCD_Init();

- ➢ Configurează LCD-ul pentru a avea 2 linii,display ON, cursor OFF și poziția inițială (0,0).

KEYB_Init();

- ➢ Configurează driverul pentru tastatura locală.

UART0_Init(EIGHT_BIT, BAUDRATE_IO);

- ➢ Configurează UART0 pentru transmisie și recepție de date prin USB-UART (P0.0 si P0.1)

Timer0_Init();

- ➢ inițializare Timer 0

EA = 1;

- ➢ permite/validează întreruperile în program

SFRPAGE = LEGACY_PAGE;

- ➢ selecteaza pagina 0 SFR

```
for(i = 0; i < NR_NODURI; i++){    // initializare structuri de date
        retea[i].full = 0;              // inițializează buffer gol pentru toate nodurile
        retea[i].bufasc[0] = ':';     // pune primul caracter in buffer-ele ASCII ":"
}
Afisare_meniu();        // Afișează meniul de comenzi
```

Bucla principala

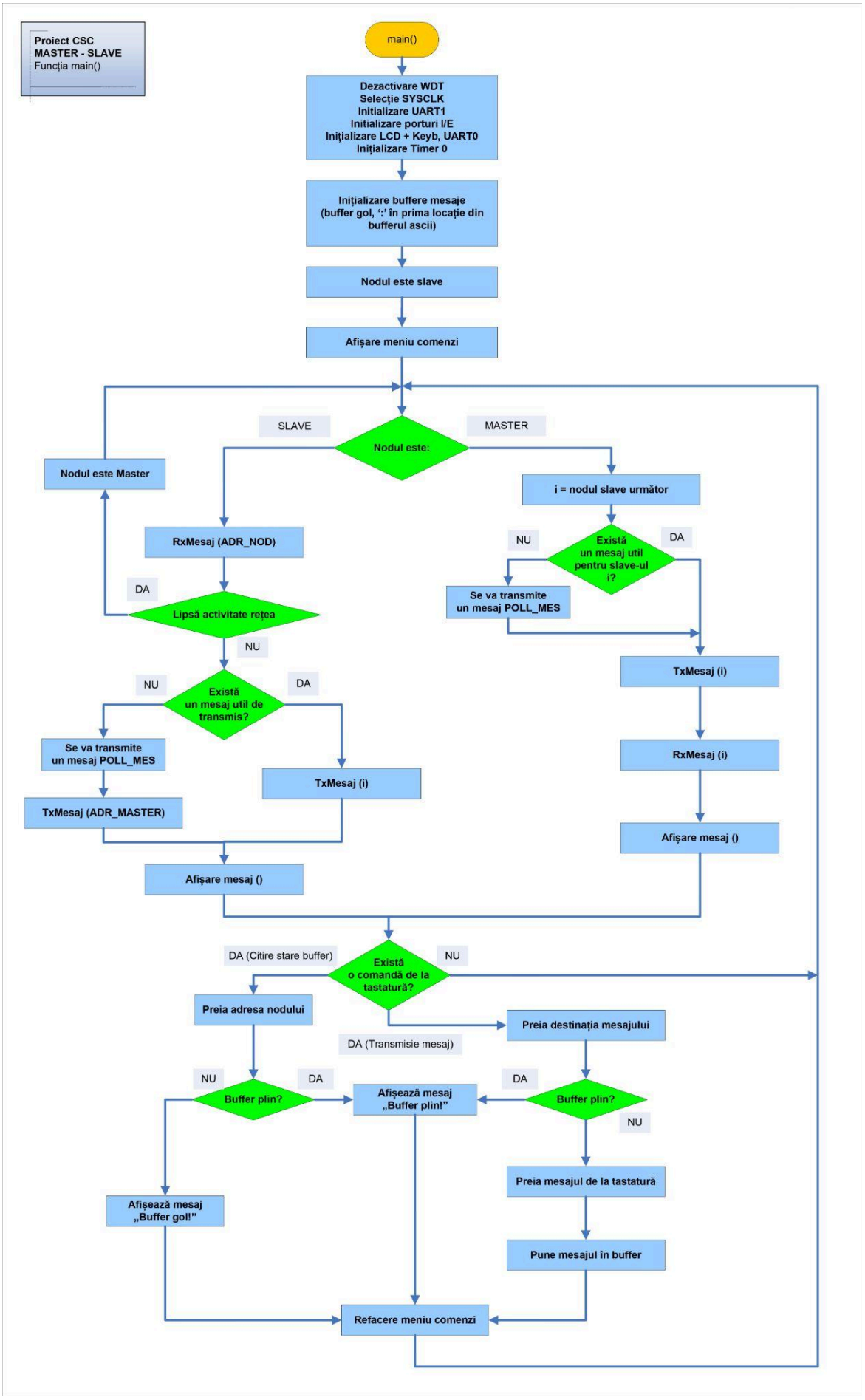
Bucla principală rulează indefinit și gestionează stările finite ale automatului de comunicație (FSA) și interfața cu operatorul, bucla ocupându-se de primirea și transmiterea mesajelor, precum și de interacțiunea cu operatorul prin intermediul funcțiilor eferente.

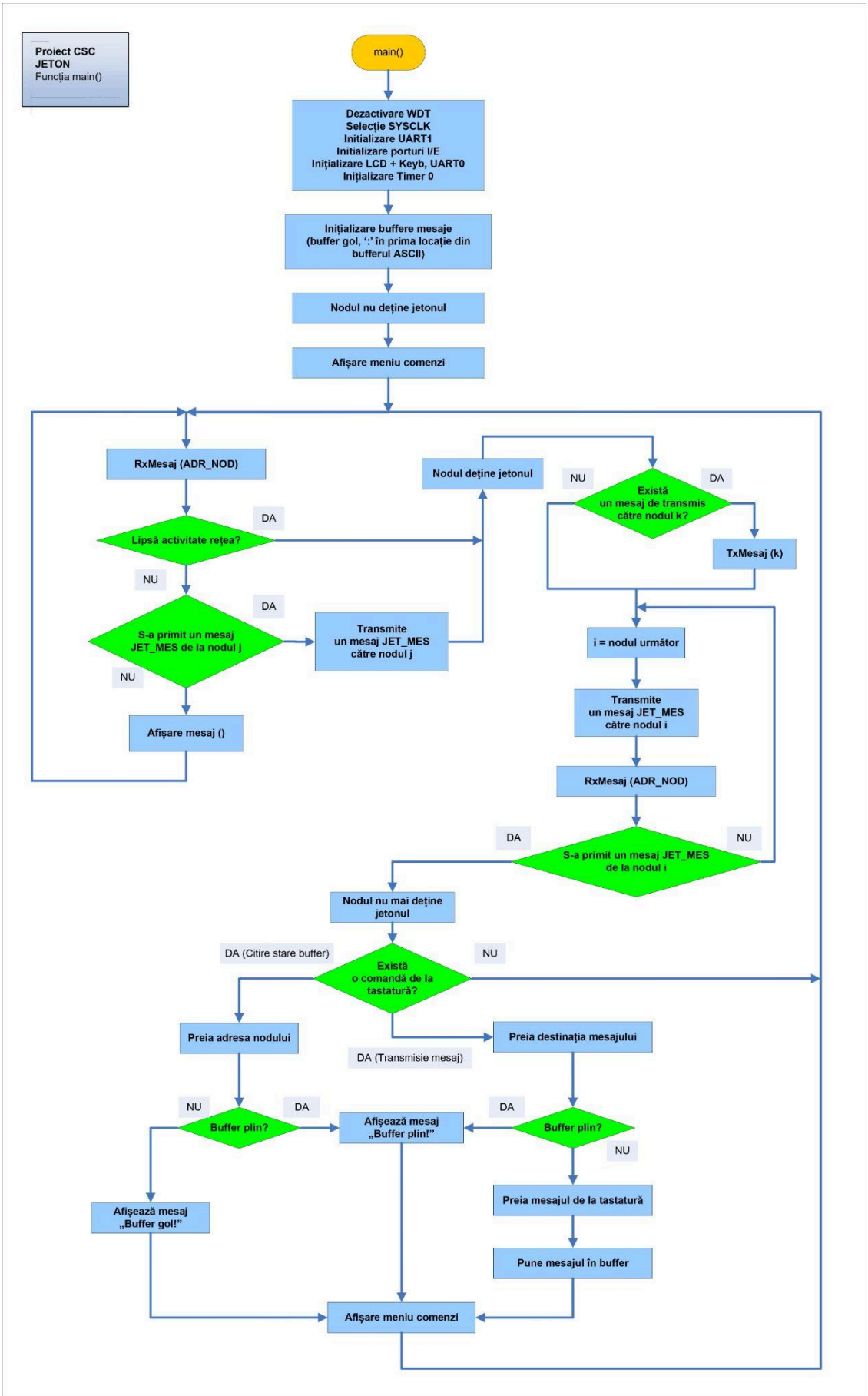
Stările finite ale automatului de comunicație (FSA) sunt:

- ➔ Starea 0: Nodul așteaptă să primească un mesaj sau un jeton.
  - ◆ Protocol JT: Nodul așteaptă un mesaj util sau jetonul. Dacă primește jetonul, trece în starea 1.
  - ◆ Protocol MS: Nodul slave așteaptă un mesaj de la master. Dacă apare un timeout, nodul devine master și trece în starea 2.
- ➔ Starea 1: Nodul are jetonul (JT) sau trebuie să trimită un mesaj (MS).
  - ◆ Protocol JT: Nodul caută un mesaj de trimis. Dacă găsește, trimite mesajul și trece în starea 2.
  - ◆ Protocol MS: Nodul trimite un mesaj către master sau construiește un mesaj de interogare dacă nu există alt mesaj de trimis. Revine în starea 0.
- ➔ Starea 2: Nodul trimite jetonul (JT) sau un mesaj de interogare (MS).
  - ◆ Protocol JT: Nodul trimite jetonul următorului nod și așteaptă confirmarea, trecând în starea 3.
  - ◆ Protocol MS: Nodul trimite un mesaj către următorul slave și așteaptă răspunsul, trecând în starea 3.
- ➔ Starea 3: Nodul așteaptă răspunsul la mesajul trimis.
  - ◆ Protocol JT: Nodul așteaptă confirmarea transferului jetonului. Dacă primește confirmarea, revine în starea 0.
  - ◆ Protocol MS: Nodul așteaptă răspunsul slave-ului. După răspuns, revine în starea 2.

Interfața cu operatorul/clientul/utilizatorul este asigurata de funcția UserIO(), ce este apelată la fiecare iterație a buclei principale pentru a gestiona interacțiunile cu utilizatorul, cum ar fi citirea tastaturii sau afișarea pe LCD.

Schemele logice ale programului principal


Proiect CSC
MASTER - SLAVE
Funcția main()
main()
Dezactivare WDT
Selecție SYSCLK
Initializare UART1
Initializare porturi I/E
Inițializare LCD + Keyb, UART0
Inițializare Timer 0
Inițializare buffere mesaje
(buffer gol, ':' în prima locație din bufferul ascii)
Nodul este slave
Afișare meniu comenzi
Nodul este:
SLAVE
MASTER
Nodul este Master
RxMesaj (ADR_NOD)
DA
Lipsă activitate rețea
NU
NU
Există
un mesaj util de
transmis?
DA
Se va transmite
un mesaj POLL_MES
TxMesaj (ADR_MASTER)
TxMesaj (i)
Afișare mesaj ()
i = nodul slave următor
NU
Există
un mesaj util
pentru slave-ul
i?
DA
Se va transmite
un mesaj POLL_MES
TxMesaj (i)
RxMesaj (i)
Afișare mesaj ()
DA (Citire stare buffer)
Există
o comandă de la
tastatură?
NU
Preia adresa nodului
Preia destinația mesajului
DA (Transmisie mesaj)
NU
Buffer plin?
DA
Afișează mesaj
„Buffer plin!"
DA
Buffer plin?
NU
Afișează mesaj
„Buffer gol!"
Preia mesajul de la tastatură
Pune mesajul în buffer
Refacere meniu comenzi


Proiect CSC
JETON
Funcția main()
main()
Dezactivare WDT
Selecție SYSCLK
Initializare UART1
Initializare porturi I/E
Inițializare LCD + Keyb, UART0
Inițializare Timer 0
Inițializare buffere mesaje
(buffer gol, ':' în prima locație din
bufferul ASCII)
Nodul nu deține jetonul
Afișare meniu comenzi
RxMesaj (ADR_NOD)
Lipsă activitate rețea?
DA
NU
S-a primit un mesaj
JET_MES de la nodul j
DA
NU
Transmite
un mesaj JET_MES
către nodul j
Afișare mesaj ()
Nodul deține jetonul
Există
un mesaj de transmis
către nodul k?
NU
DA
TxMesaj (k)
i = nodul următor
Transmite
un mesaj JET_MES
către nodul i
RxMesaj (ADR_NOD)
S-a primit un mesaj JET_MES
de la nodul i
DA
NU
Nodul nu mai deține
jetonul
Există
o comandă de la
tastatură?
DA (Citire stare buffer)
NU
Preia adresa nodului
Preia destinația mesajului
DA (Transmisie mesaj)
Buffer plin?
NU
DA
Afișează mesaj
„Buffer plin!"
Buffer plin?
DA
NU
Afișează mesaj
„Buffer gol!"
Preia mesajul de la tastatură
Pune mesajul în buffer
Afișare meniu comenzi

6. Pregătirea mesajelor pentru transmisie și servicii de transmisie

Serviciile de comunicație pentru pregătirea și transmisia mesajului se realizează prin intermediul funcțiilor ***TxMesaj*** și ***bin2ascii***. Funcția TxMesaj este responsabilă pentru pregătirea și transmiterea unui mesaj către un nod specificat în rețea, folosind un buffer pentru stocarea mesajului în format ASCII.

Descrierea serviciilor de comunicație

Funcția **TxMesaj** oferă următoarele servicii de comunicație:

- *Pregătirea mesajului*: funcția calculează suma de control (SC) a mesajului și o adaugă la mesaj.
- *Conversia mesajului* în format ASCII HEX: funcția bin2ascii este utilizată pentru a converti octeții mesajului în caractere ASCII HEX.
- *Transmiterea mesajului*: funcția transmite mesajul către nodul destinatie utilizând protocolul UART.

Descrierea codului implementat

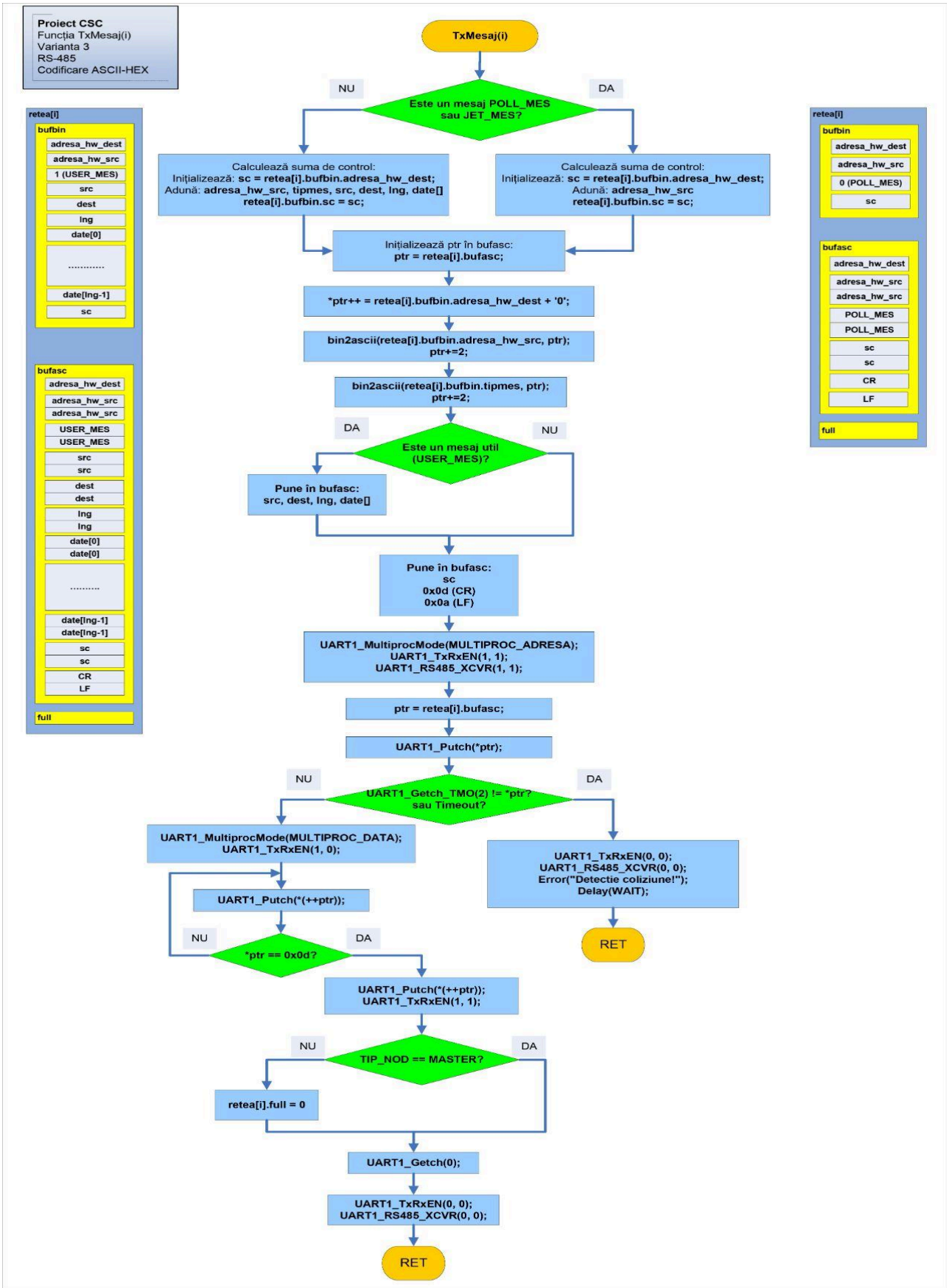
Codul implementat poate fi împărțit în următoarele secțiuni:

- Calcularea și stocarea sumei de control (SC):
    - Pentru mesajele de tip POLL_MES sau JET_MES, se calculează SC ca suma adresei hardware a destinatarului și a sursei.
    - Pentru alte tipuri de mesaje, SC se calculează ca suma adresei hardware a destinatarului, a sursei, tipului mesajului, adresei sursei mesajului, adresei destinatarului mesajului, lungimii datelor și a fiecărui octet de date din mesaj.
- Conversia și pregătirea mesajului:
    - Funcția bin2ascii este utilizată pentru a converti octeții din format binar în format ASCII HEX.
    - Mesajul este construit în buffer-ul ASCII (bufasc), incluzând adresele hardware, tipul mesajului, adresele sursei și destinatarului, lungimea datelor, datele efective și SC.
    - Se adaugă caracterele de terminare CR (carriage return) și LF (line feed) la sfârșitul mesajului.

- Transmisia mesajului:
    - Se activează transmisia și recepția pe UART1 și se configurează modul multiprocesor pentru a transmite adresa.
    - Dacă caracterul primit este diferit de cel transmis sau dacă se detectează un timeout, transmisia este anulată, se afișează un mesaj de eroare și se așteaptă o perioadă de timp.
    - Dacă nu există erori, se continuă transmisia octeților de date, iar la final se dezactivează transmisia și recepția pe UART1 și RS485.
    - Dacă nodul nu este de tip MASTER, se golește buffer-ul pentru a pregăti nodul pentru noi mesaje.

Prin acestea, funcția TxMesaj se asigură că mesajele sunt pregătite și transmise corect mai departe, gestionând eventualele erori de transmisie și asigurând conversia adecvată a datelor din format binar în ASCII HEX. Funcția bin2ascii facilitează această conversie, necesară pentru formatarea corectă a mesajelor în vederea transmiterii lor prin UART.

Schema logică a serviciului de transmisie


Proiect CSC
Funcția TxMesaj(i)
Varianta 3
RS-485
Codificare ASCII-HEX
TxMesaj(i)
Este un mesaj POLL_MES sau JET_MES?
NU
DA
Calculează suma de control:
Inițializează: sc = retea[i].bufbin.adresa_hw_dest;
Adună: adresa_hw_src, tipmes, src, dest, lng, date[]
retea[i].bufbin.sc = sc;
Calculează suma de control:
Inițializează: sc = retea[i].bufbin.adresa_hw_dest;
Adună: adresa_hw_src
retea[i].bufbin.sc = sc;
Inițializează ptr în bufasc:
ptr = retea[i].bufasc;
*ptr++ = retea[i].bufbin.adresa_hw_dest + '0';
bin2ascii(retea[i].bufbin.adresa_hw_src, ptr);
ptr+=2;
bin2ascii(retea[i].bufbin.tipmes, ptr);
ptr+=2;
Este un mesaj util (USER_MES)?
DA
NU
Pune în bufasc:
src, dest, lng, date[]
Pune în bufasc:
sc
0x0d (CR)
0x0a (LF)
UART1_MultiprocMode(MULTIPROC_ADRESA);
UART1_TxRxEN(1, 1);
UART1_RS485_XCVR(1, 1);
ptr = retea[i].bufasc;
UART1_Putch(*ptr);
UART1_Getch_TMO(2) != *ptr? sau Timeout?
NU
DA
UART1_MultiprocMode(MULTIPROC_DATA);
UART1_TxRxEN(1, 0);
UART1_TxRxEN(0, 0);
UART1_RS485_XCVR(0, 0);
Error("Detectie coliziune!");
Delay(WAIT);
RET
UART1_Putch(*(++ptr));
*ptr == 0x0d?
NU
DA
UART1_Putch(*(++ptr));
UART1_TxRxEN(1, 1);
TIP_NOD == MASTER?
NU
DA
retea[i].full = 0
UART1_Getch(0);
UART1_TxRxEN(0, 0);
UART1_RS485_XCVR(0, 0);
RET
retea[i]
bufbin
adresa_hw_dest
adresa_hw_src
1 (USER_MES)
src
dest
lng
date[0]
.....
date[lng-1]
sc
bufasc
adresa_hw_dest
adresa_hw_src
adresa_hw_src
USER_MES
USER_MES
src
src
dest
dest
lng
lng
date[0]
date[0]
.....
date[lng-1]
date[lng-1]
sc
sc
CR
LF
full
retea[i]
bufbin
adresa_hw_dest
adresa_hw_src
0 (POLL_MES)
sc
bufasc
adresa_hw_dest
adresa_hw_src
adresa_hw_src
POLL_MES
POLL_MES
sc
sc
CR
LF
full

7. Recepţia mesajelor – descriere, schema logică şi implementare

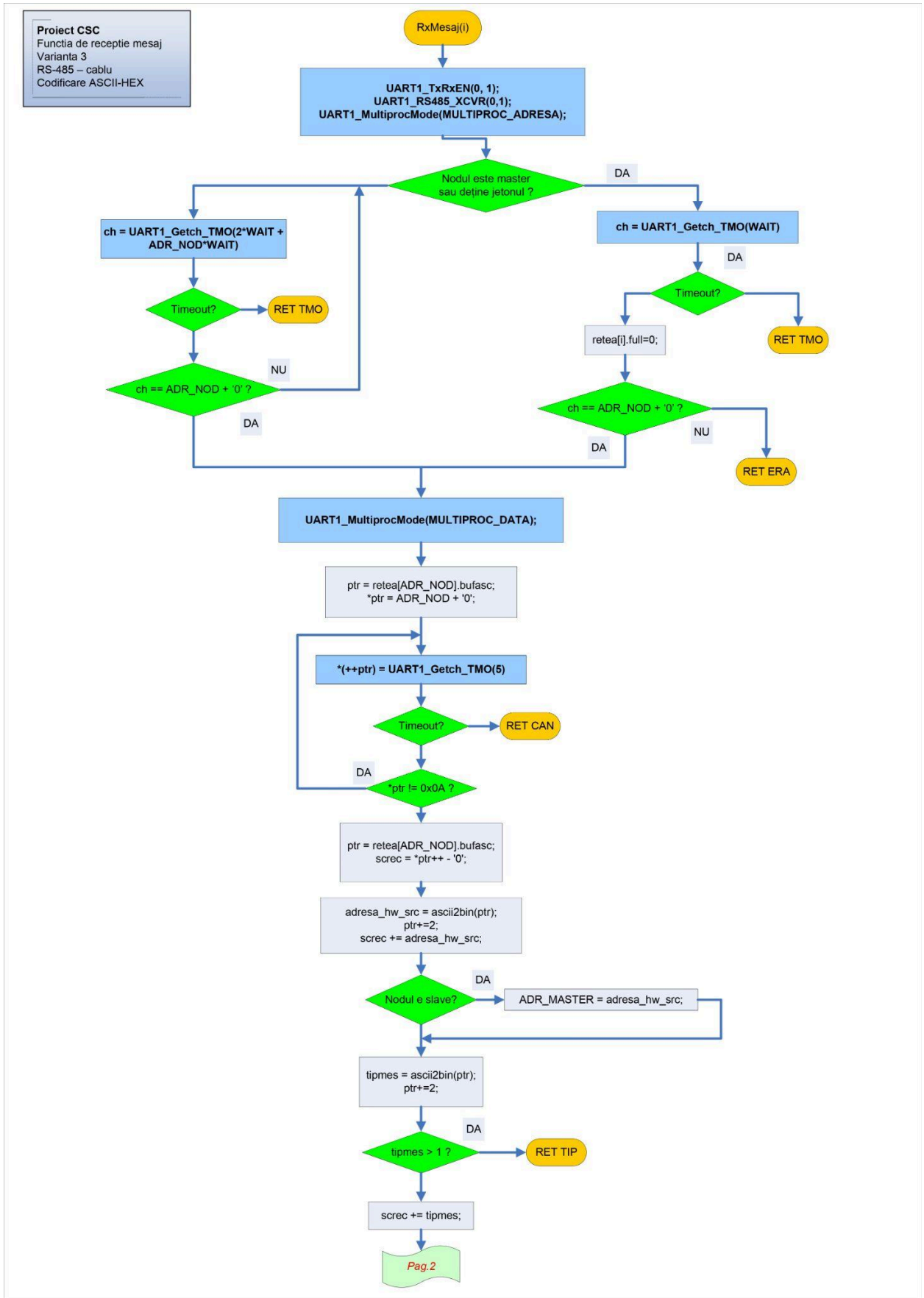
Serviciile de comunicație pentru recepția mesajelor se realizează prin intermediul funcțiilor: ***RxMesaj*** si ***ascii2bin*****.** Funcția ***RxMesaj*** fiind responsabilă pentru recepția mesajelor de la un nod specificat în rețea, verificând adresa nodului destinatar, tipul mesajului și validitatea sumei de control. Aceasta funcționează diferit pentru nodurile master și slave și implică mai multe etape de procesare și validare a datelor primite. Funcția **ascii2bin** este utilizată pentru a converti caracterele ASCII HEX în format binar.

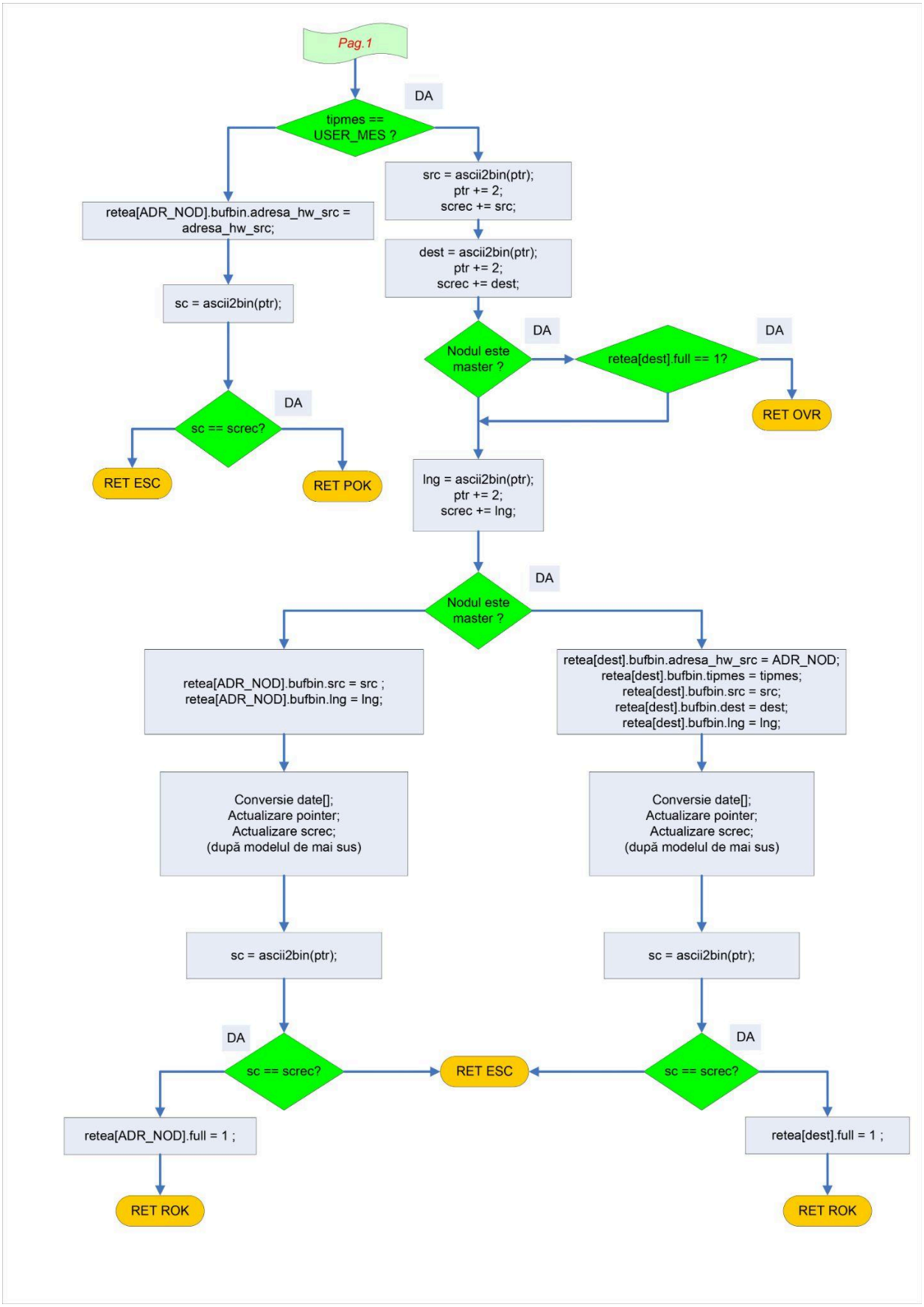
Descrierea funcţiei de recepţie a mesajelor

- Inițializarea pentru recepție:
    - Dezactivează transmisia și activează recepția pe UART1 și RS485.
    - Configurează modul multiprocesor pentru recepția octeților de adresă.
- Recepția adresei:
    - Dacă nodul curent este master sau de tip jeton, așteaptă primirea caracterului de adresă de la un nod slave.
    - Dacă nodul curent este slave, așteaptă primirea caracterului de adresă de la un nod master.
    - Verifică dacă adresa primită corespunde adresei nodului curent, în caz contrar, termină recepția cu o eroare de adresă (ERA).
- Recepția datelor:
    - Configurează modul multiprocesor pentru recepția octeților de date.
    - Inițializează un pointer către buffer-ul ASCII al nodului curent și stochează adresa nodului destinatar în buffer.
    - Primește restul mesajului până la caracterul LF (0x0A).
- Validarea și procesarea mesajului:
    - Inițializează suma de control recepționată (screc) cu adresa hardware a nodului destinatar.
    - Determină adresa hardware a sursei mesajului și adaugă la suma de control.

    - Dacă nodul curent este slave, actualizează adresa master.
    - Determină tipul mesajului și verifică dacă este valid.
    - Dacă mesajul este de tip USER_MES, procesează și validează adresa sursei, adresa destinatarului și lungimea datelor.
    - Dacă nodul este master, verifică dacă buffer-ul destinatarului este deja plin și termină recepția cu o eroare de buffer plin (OVR).
    - Procesează fiecare octet de date, adaugându-l la suma de control și stocându-l în buffer-ul binar.
    - Verifică suma de control pentru a determina validitatea mesajului.
    - Dacă mesajul este valid, marchează buffer-ul ca plin și returnează ROK, în caz contrar, returnează eroare de sumă de control (ESC).
- Mesajele de tip POLL_MES:
    - Dacă mesajul este de tip POLL_MES, memorează adresa hardware a sursei și verifică suma de control.
    - Dacă suma de control este corectă, returnează POK, în caz contrar, returnează ESC.

Schema logică a serviciului de recepție


Proiect CSC
Functia de receptie mesaj
Varianta 3
RS-485 – cablu
Codificare ASCII-HEX
RxMesaj(i)
UART1_TxRxEN(0, 1);
UART1_RS485_XCVR(0,1);
UART1_MultiprocMode(MULTIPROC_ADRESA);
Nodul este master sau deține jetonul ?
DA
ch = UART1_Getch_TMO(2*WAIT + ADR_NOD*WAIT)
Timeout?
RET TMO
NU
ch == ADR_NOD + '0' ?
DA
ch = UART1_Getch_TMO(WAIT)
DA
Timeout?
RET TMO
retea[i].full=0;
ch == ADR_NOD + '0' ?
NU
DA
RET ERA
UART1_MultiprocMode(MULTIPROC_DATA);
ptr = retea[ADR_NOD].bufasc;
*ptr = ADR_NOD + '0';
*(++ptr) = UART1_Getch_TMO(5)
Timeout?
RET CAN
DA
*ptr != 0x0A ?
ptr = retea[ADR_NOD].bufasc;
screc = *ptr++ - '0';
adresa_hw_src = ascii2bin(ptr);
ptr+=2;
screc += adresa_hw_src;
Nodul e slave?
DA
ADR_MASTER = adresa_hw_src;
tipmes = ascii2bin(ptr);
ptr+=2;
tipmes > 1 ?
DA
RET TIP
screc += tipmes;
Pag.2


Pag.1
tipmes == USER_MES ?
DA
retea[ADR_NOD].bufbin.adresa_hw_src = adresa_hw_src;
sc = ascii2bin(ptr);
sc == screc?
DA
RET ESC
RET POK
src = ascii2bin(ptr);
ptr += 2;
screc += src;
dest = ascii2bin(ptr);
ptr += 2;
screc += dest;
Nodul este master ?
DA
retea[dest].full == 1?
DA
RET OVR
lng = ascii2bin(ptr);
ptr += 2;
screc += lng;
Nodul este master ?
DA
retea[ADR_NOD].bufbin.src = src ;
retea[ADR_NOD].bufbin.lng = lng;
retea[dest].bufbin.adresa_hw_src = ADR_NOD;
retea[dest].bufbin.tipmes = tipmes;
retea[dest].bufbin.src = src;
retea[dest].bufbin.dest = dest;
retea[dest].bufbin.lng = lng;
Conversie date[];
Actualizare pointer;
Actualizare screc;
(după modelul de mai sus)
Conversie date[];
Actualizare pointer;
Actualizare screc;
(după modelul de mai sus)
sc = ascii2bin(ptr);
sc = ascii2bin(ptr);
DA
sc == screc?
RET ESC
DA
sc == screc?
retea[ADR_NOD].full = 1 ;
retea[dest].full = 1 ;
RET ROK
RET ROK

8. Concluzii

Repartizarea sarcinilor in cadrul echipei proiectului s-a realizat in mod deliberat, fiecare membru al echipei si-a ales sarcinile in functiie de ceea ce stia mai bine sau ceea ce credea ca i-ar prinde bine sa dezvolte, alegand una din cele 4 roluri: coordonator, responsabil transmisie, resonsabil receptie sau responsabil interfata.

Fiind o echipa de 5 oameni, a fost interesant modul de coordonare "dual", totul a decurs mult mai lin decat am fi anticipat, si totodata am abordat astfel două protocoale inca de baza in industria automatizarilor, protocoalele Master-Slave si Jeton. Aici cei doi coordonatori s-au ocupat fiecare de realizarea in programului principal a protocolului specific ales. Pentru partea de comunicație (transmisia și recepția), membrii s-au asigurat de o dezvoltare cât mai utila a programului, care să deserveasca implementarii ambelor protocoale utilizate in proiect. Iar pentru partea de interfata, un coleg s-a ocupat de implementare, garantând funcționalitatea LCD-ului și captarea corectă a intrărilor de la utilizator.

Teste realizate in dezvoltarea proiectului s-au realizat atat modulare (testand componenta cu componenta) cat si integral (imbinand toate componentele spre a urmari comportamentul protocoalelor implementate). Iar dupa multe sesiuni de debug si small-fixes, ambele protocoale au avut ca **rezultat** implementarea cu succes a acestora, functionarea fiind corespunzatoare comportamentului dorit.

Problemele întâmpinate

- Au existat mici probleme la componenta de transmisie a programului, datorate stocarii necorespunzatoare a unor informatii, ceea ce determina trasnmiterea corecta a unui mesaj, insa avand un continut incomplete/incoerent privind mesajul trasnmis de nodul sursa
- Alte probleme au mai fost de natura umana, s-au mai uitat comentat o afisare si aveam impresia ca nodul master nu primeste mesaj de la slave, problema fiind insa doar la afisare :))
- Au mai existat alte mici probleme de incompatibilitate, dar s-au rezolvat destul de rapid acestea

Posibilităţi de îmbunătăţire a protocolului,

- Pentru protocolul Master-Slave:
    - Optimizare a selecției nodurilor slave prin inlocuirea parcurgerii complete a nodurilor cu o listă predefinită de noduri active sau cu date de transmisie pentru a reduce timpul de procesare.
    - Implementarea prioritățiilor mesajelor, prin introducerea unui sistem de prioritizare putem asigura transmiterea mesajelor critice înaintea celor cu prioritate mai mică.

    - Optimizarea dimensiunii și gestionării bufferului pentru a preveni pierderea datelor și pentru a asigura o transmisie eficientă
- Pentru protocolul pe baza de Jeton:
    - Reducerea timpilor de așteptare (WAIT/2) pentru a reduce latența în transmiterea jetonului și accelera comunicarea în rețea.
    - Introducerea unui algoritm care să prioritizeze nodurile pe baza încărcării lor actuale sau a numărului de mesaje în așteptare, pentru a optimiza fluxul de date în rețea.

Bibliografie

1. Cartea tehnică a microsistemul BIG8051.
2. Suportul de proiect - „Comunicaţii în sisteme de conducere”.
3. Îndrumarul de laborator pentru disciplina „Comunicaţii în sisteme de conducere”.