Universitatea Tehnică “Gheorghe Asachi” din Iaşi
Facultatea de Automatică şi Calculatoare
Domeniul: Ingineria sistemelor
Specializarea: Automatică şi Informatică Aplicată
Anul universitar 2022-2023

Proiect la disciplina

Comunicaţii în sisteme de conducere

Tema *3*

Student:	**Grupa:**	**Rol (C,T,R,U)**
Mateeșescu Niki	1405B	C
Bucataru Laura-Elena	1405B	T
Agrigoroaie Marian	1405B	R
Iorga Lorena	1405B	U

Cuprins

1. Tema proiectului 2
2. Scurtă prezentare a resurselor HW şi SW utilizate 2
3. Descrierea protocolului 9
4. Interfața I/E și conectarea la mediul de comunicație (nivelul fizic) 11
5. Programul principal 13
6. Pregătirea mesajelor pentru transmisie și servicii de transmisie 17
7. Recepţia mesajelor – descriere, schema logică şi implementare 20
8. Concluzii 26

Bibliografie 26

1. Tema proiectului

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 comunicaţie	115200, 9, N, 1
		Adresă HW nod destinaţie
		Adresă HW nod sursă
		Tipul mesajului: 0 sau 1
		Sursa mesajului (numai mesaje de tip 1)
		Destinaţia 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

Formatul mesajelor:

Binar:

Adresă hardware nod destinaţie	Adresă hardware nod sursă	Tip mesaj	Adresă nod sursă mesaj	Adresă nod destinaţie mesaj	Lungime	Date	Sumă de control
1 octet	1 octet	1 octet	1 octet	1 octet	1 octet	Lungime	1 octet

ASCII hex:

Adresă hardware nod destinaţie	Adresă hardware nod sursă	Tip mesaj	Adresă nod sursă mesaj	Adresă nod destinaţie mesaj	Lungime	Date	Suma de control	Sfârşit mesaj
1 octet	2 octeți	2 octeți	2 octeți	2 octeți	2 octeți	2 x Lungime	2 octeți	0Dh, 0Ah

2. Scurtă prezentare a resurselor HW şi SW utilizate

Microsistemul BIG8051

Microsistemul BIG8051 produs de MikroElektronika oferă o platformă flexibilă pentru programarea și dezvoltarea de aplicații cu microcontrolere 8051. Numeroase module, precum butoane, LED-uri, afișaje LCD, un difuzor (buzzer) piezoelectric, interfețe și controlere de comunicații, convertoare analog /numerice și numeric / analogice etc., permit utilizatorilor să testeze cu ușurință funcționarea aplicațiilor direct pe placa de dezvoltare. O vedere de ansamblu a microsistemului este prezentată în Fig.1, iar o scurtă descriere a blocurilor funcționale este dată în Fig.2.

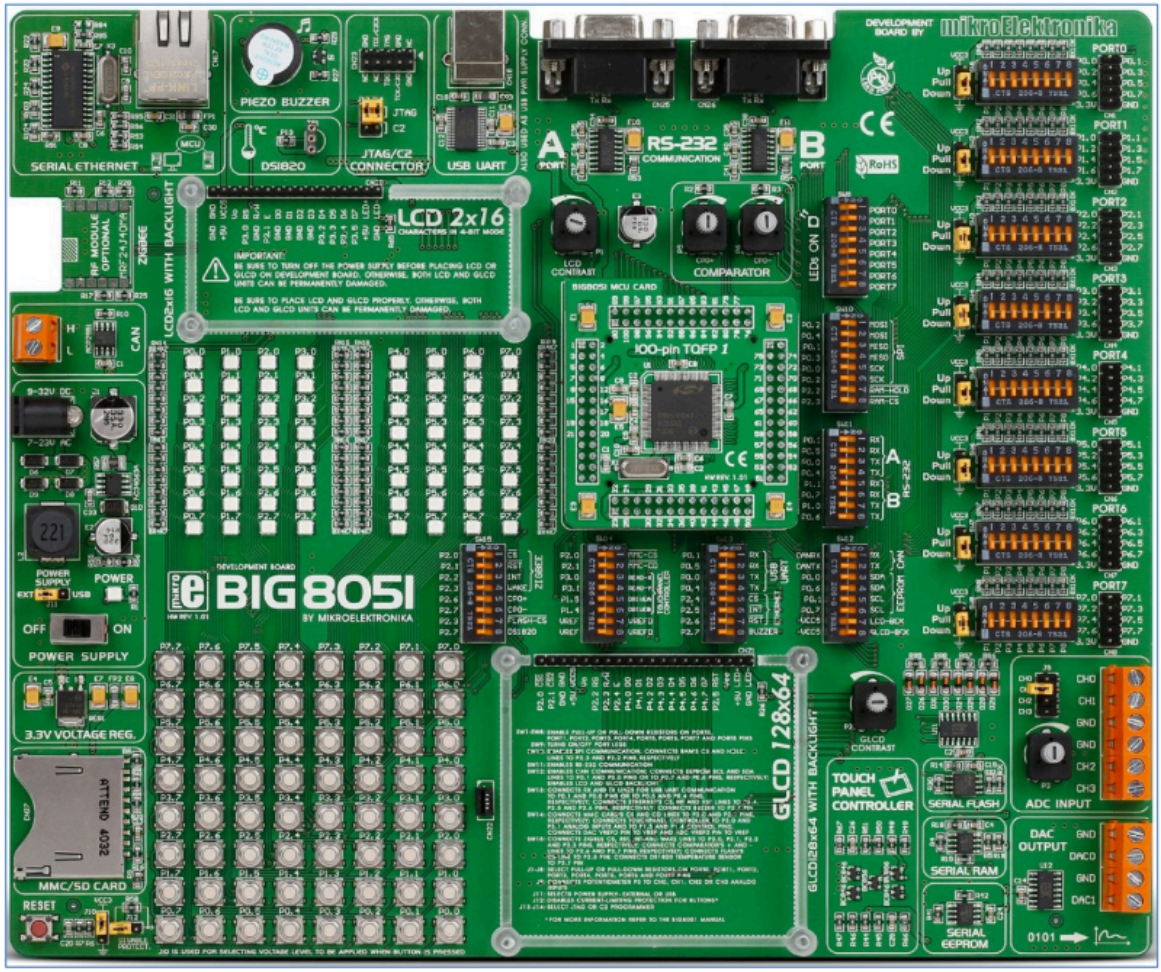


Fig.1. Microsistemul BIG8051

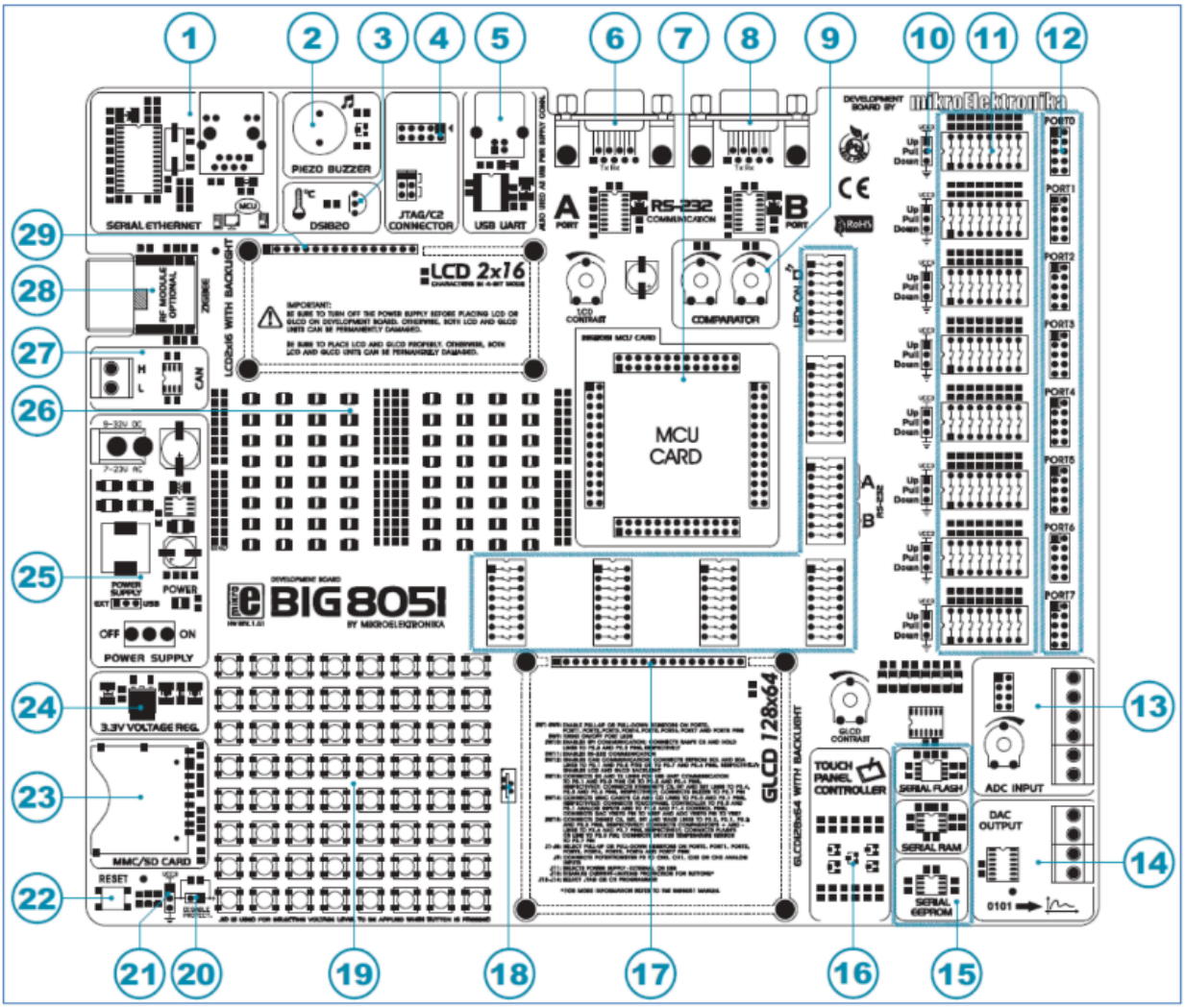


Fig. 2. Microsistemul BIG8051 – blocuri componente

1. Modul interfață Ethernet	8. Modul interfață RS-232 - B
2. Buzzer piezoelectric	9. Comparator
3. Conector pentru senzorul de temperatură	10. Jumperi selectare nivel de fixare VCC | GND
4. Conector pentru programator/debugger	11. DIP switch conectare pini la rezistoare de fixare
5. Modul USB-UART și alimentare USB	12. Conectori porturi I/E și alimentare
6. Modul interfață RS-232 - A	13. Intrări convertor A/D
7. Conector card MCU	14. Ieșiri convertor D/A

15. Modul memorii seriale externe	23. Conector card de memorie MMC/SD
16. Controler touch panel	24. Regulator de tensiune 3,3V c.c.
17. Conector pentru afișaj grafic (GLDC)	25. Conector și selecție sursă de alimentare
18. Conector pentru touch panel	26. LED-uri pentru afișarea stării porturilor
19. Butoane cu revenire conectate la pinii porturilor	27. Modul interfață CAN
20. Jumper scurtcircuitare rezistor de protecție	28. Conector interfață ZigBee
21. Jumper selectare stare logică buton apăsat	29. Conector afișaj LCD alfanumeric
22. Buton RESET	

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. 3).

Caracteristicile sale principale sunt următoarele:

- 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÷24,5 MHz;
- execută până la 25 MIPS cu oscilator de 25 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);
    - SMBus (compatibil I2C).

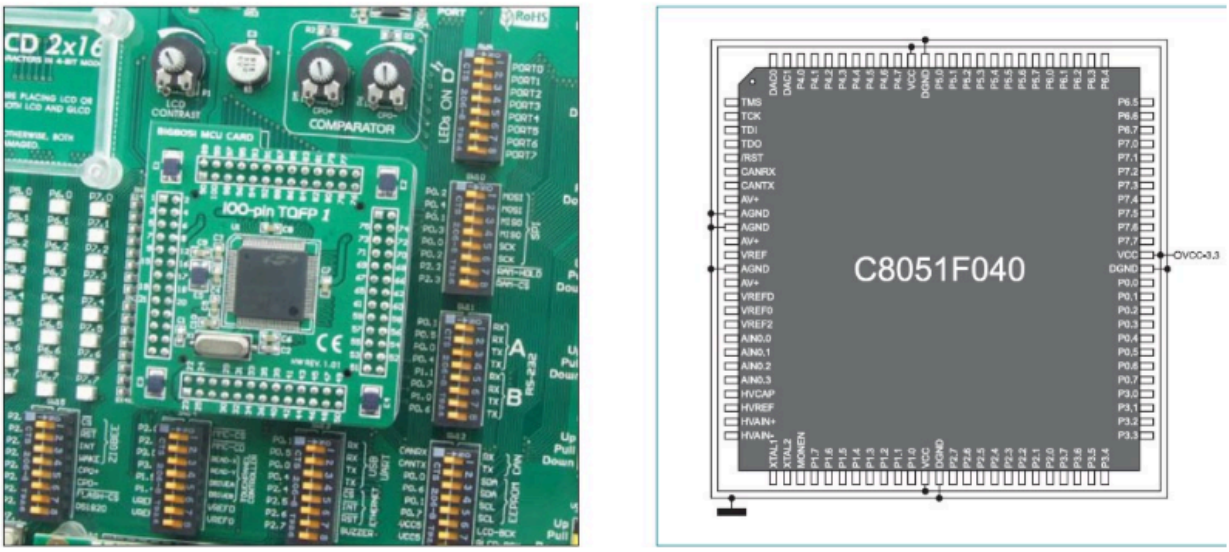


Fig. 3. Microcontrolerul C8051F040 și cardul MCU

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.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. 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. 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.

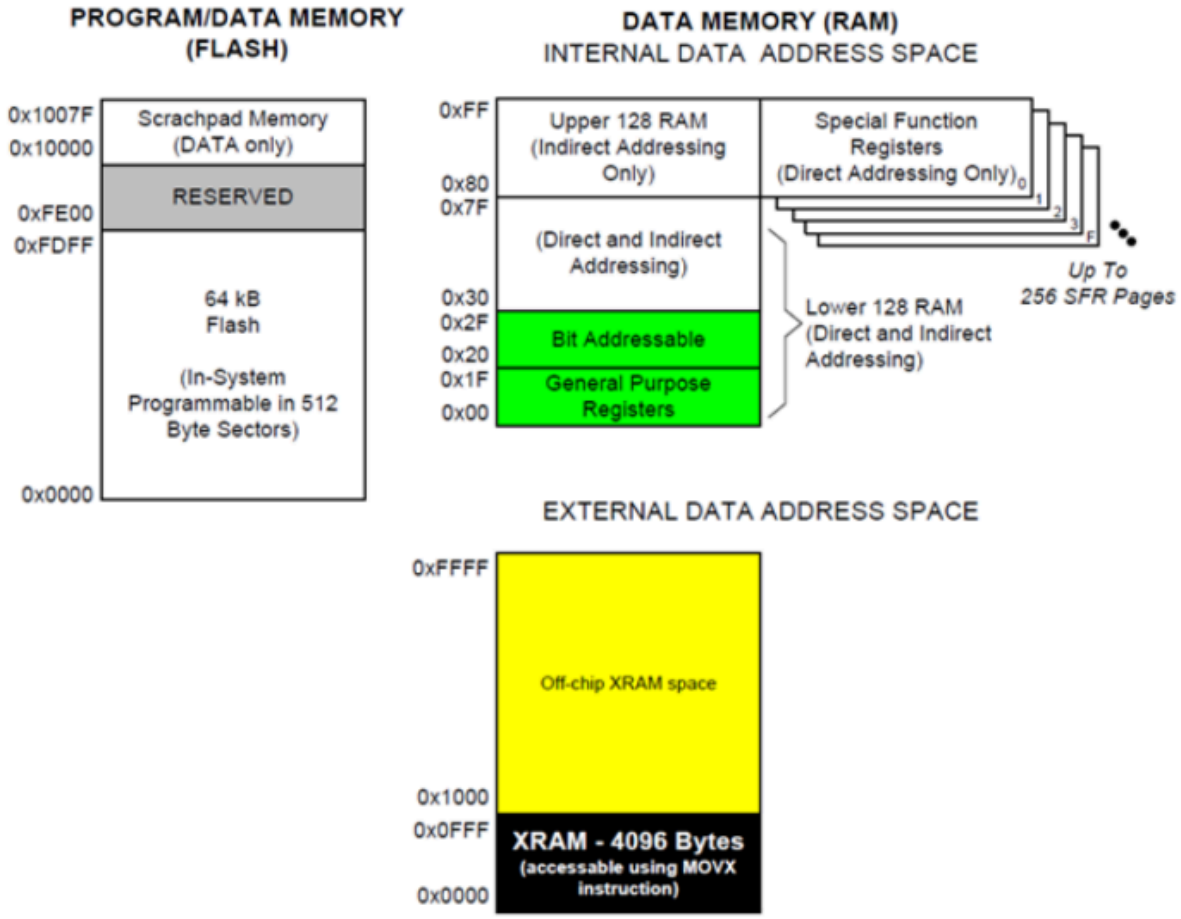


Fig. 5. Organizarea memoriei microcontrolerului C8051F040

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.

Dezvoltarea aplicaţiilor folosind Keil PK51

PK51 este o platformă integrată (IDE – Integrated Development Environment) pentru dezvoltarea aplicațiilor pentru microcontrolere 8051 și 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.

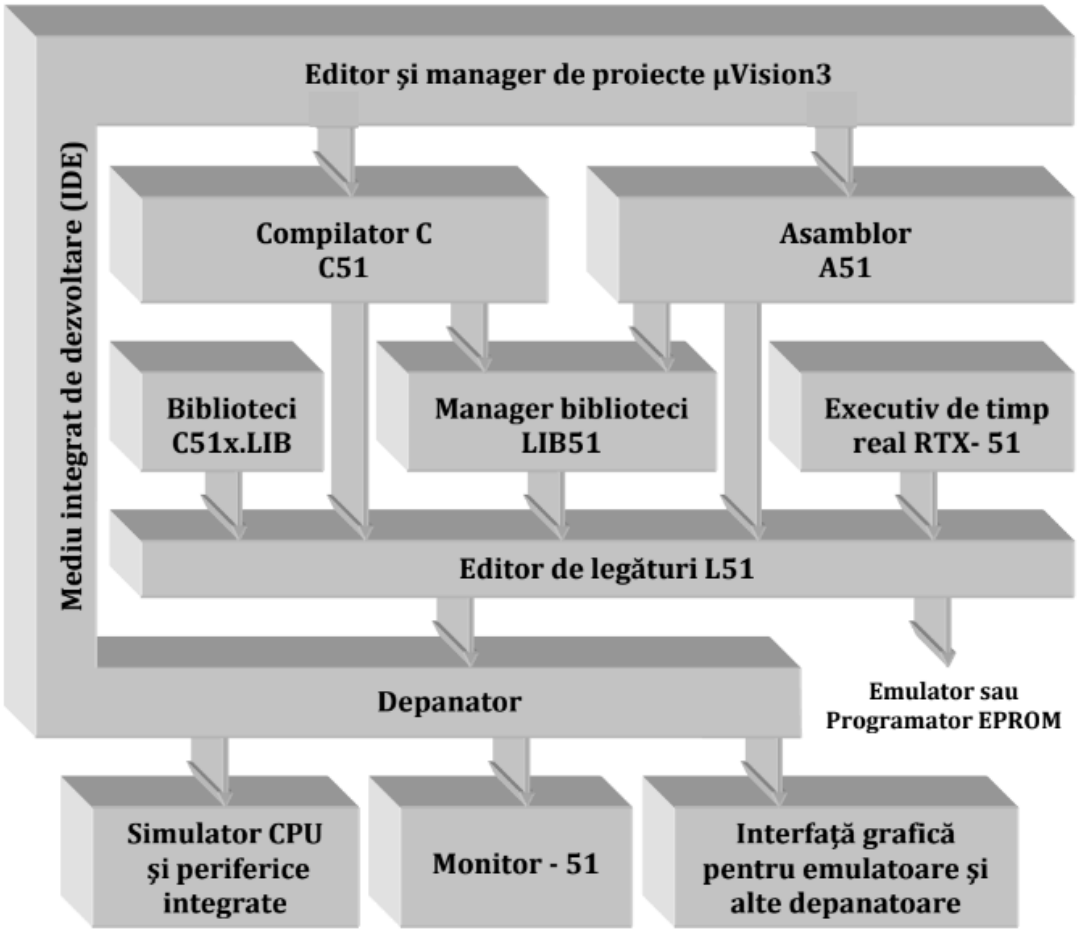


Fig. 6. Structura PK51 - ciclul de dezvoltare al unei aplicaţii

3. Descrierea protocolului Master – Slave

Tehnica de control master-slave prevede existența unui nod arbitrar central (master) care controlează momentele când celelalte noduri pot transmite sau recepționa date.Fiecare mesaj transmis de către nodurile slave va trece mai întâi prin nodul master, ajungând apoi la destinația finală.Dreptul de acces este acordat de către master, pe rând, în mod ciclic, fiecărui nod slave, printr-un mesaj adresat acestuia.

Există două tipuri de mesaje:

- ➢ de interogare (transmis de master pe rând fiecărui slave pentru a determina dacă acesta are mesaj în așteptare)
- ➢ care conține date de transmis (poate apărea ca răspuns al slave-ului la o interogare din partea masterului, urmând să fie trimis către destinația finală).

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.

Masterul 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 raspuns 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 tasta. 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 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.

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 sunt 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).

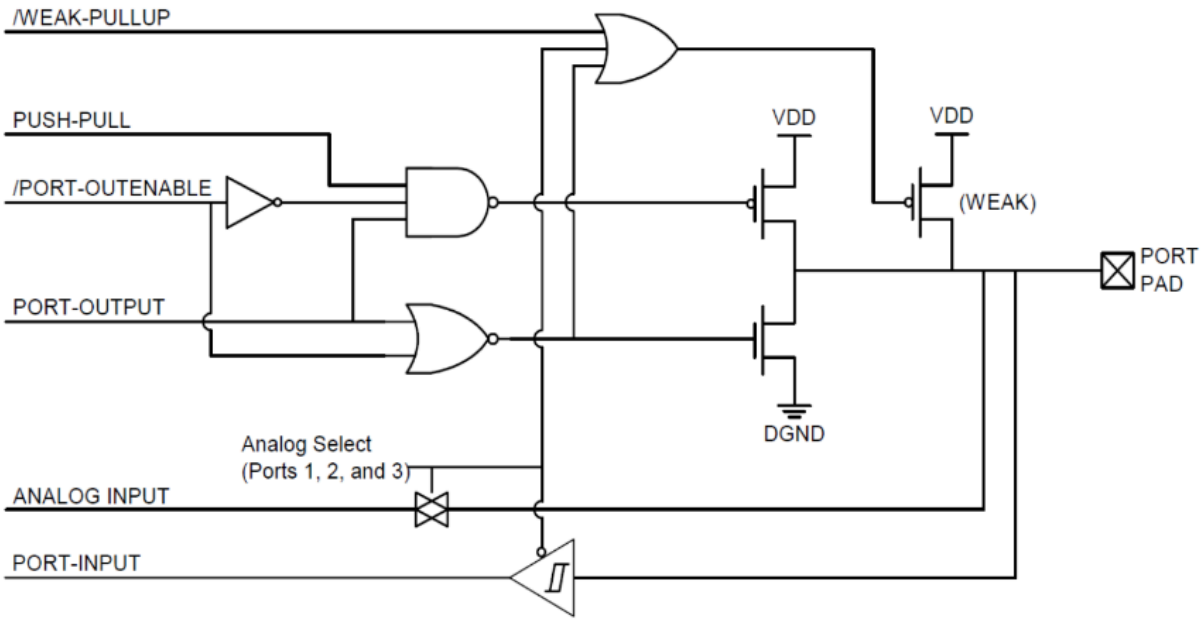


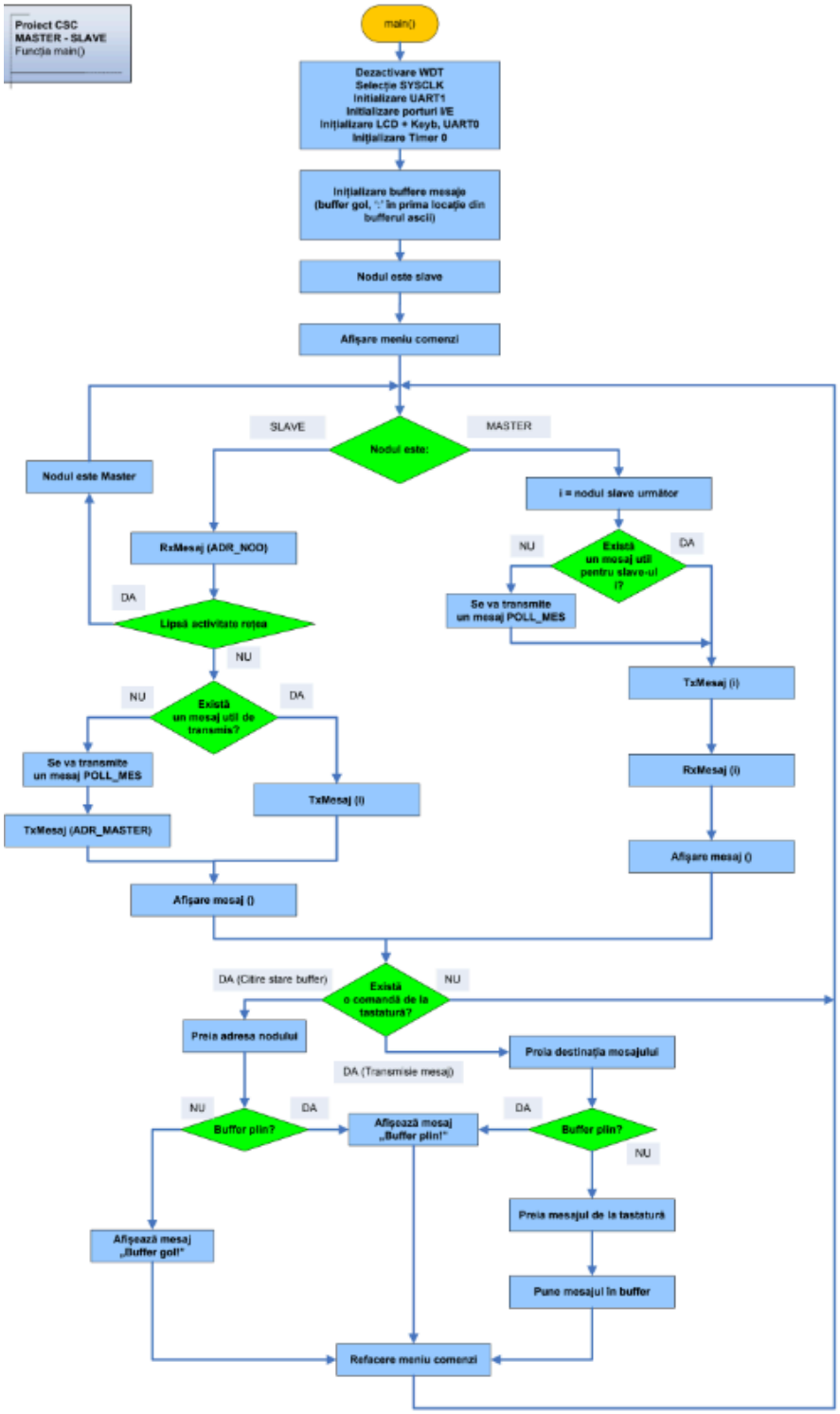
Fig.7. Schema bloc a etajului de I/E al unui pin

- **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 pull-up este automat dezactivat.
    - Rezistorii de pull-up pot fi toți dezactivați, indiferent de nivelul comandat pe linia PORT-OUTPUT, dacă XBR2.7 = 1 (bitul Weak Pull-up Disable).

5. Programul principal

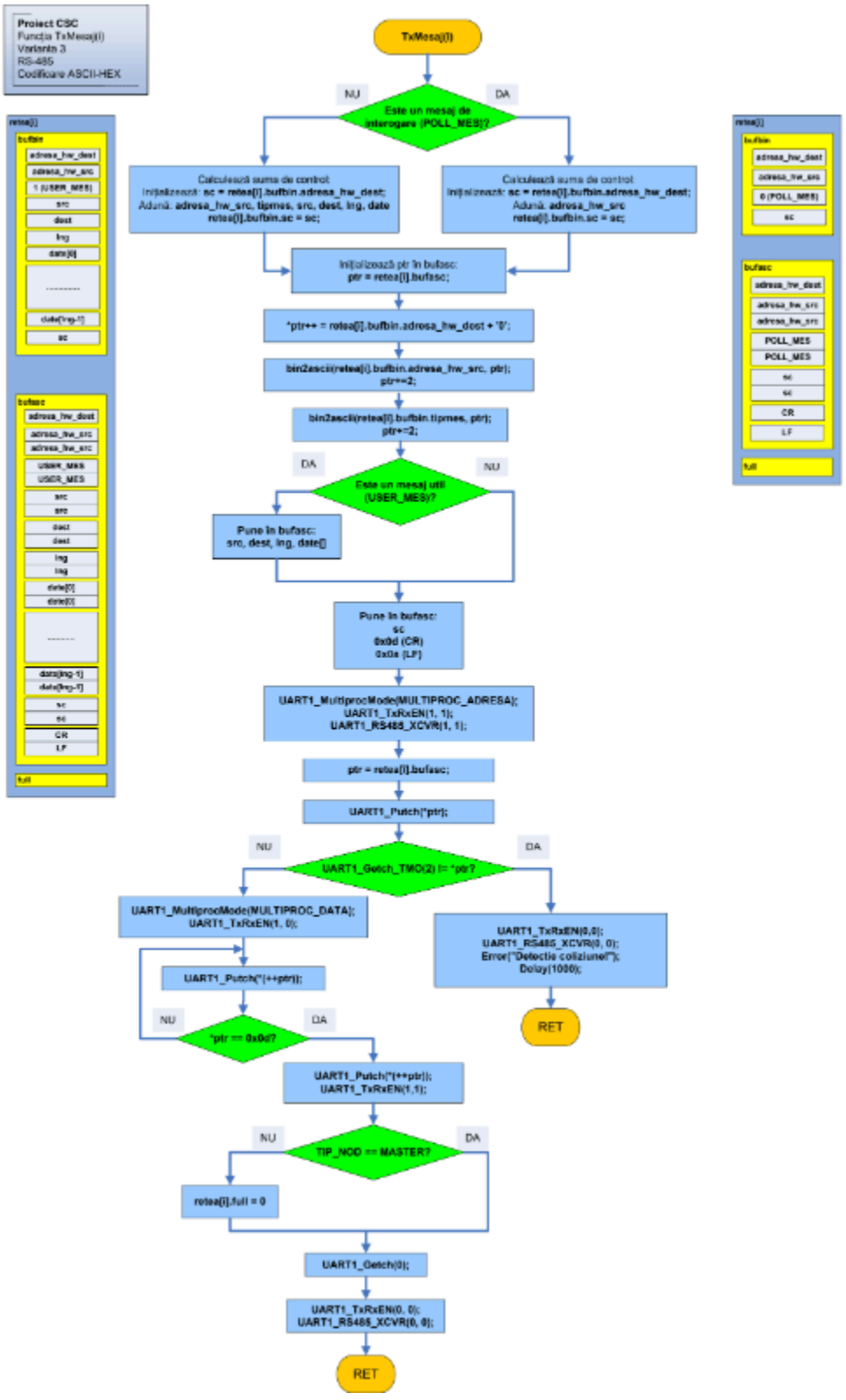


```
#include <c8051F040.h>        // declaratii SFR
#include <wdt.h>
#include <osc.h>
#include <port.h>
#include <uart1.h>
#include <lcd.h>
#include <keyb.h>
#include <Protocol.h>
#include <UserIO.h>

nod retea[NR_NODURI];                                           // reteaua Master-Slave, cu 5 noduri

unsigned char STARE_NOD = 0;                    // starea initiala a nodului curent
unsigned char TIP_NOD       =0;                 // tip nod initial: Slave sau No JET
unsigned char STARE_IO      = 0;                // stare interfata IO - asteptare comenzi
unsigned char ADR_MASTER;                                       // adresa nod master - numai MS
extern unsigned char AFISARE;

//***************************************************************************************************
void TxMesaj(unsigned char i);              // transmisie mesaj destinat nodului i
unsigned char RxMesaj(unsigned char i);                 // primire mesaj de la nodul i

//***************************************************************************************************
void main (void) {
        unsigned char i, found;         // variabile locale

        WDT_Disable();
        // dezactiveaza WDT
        SYSCLK_Init();                                          // initializeaza si selecteaza oscilatorul ales in osc.h
        UART1_Init(NINE_BIT, BAUDRATE_COM);                     // initilizare UART1  - conectata la RS232-B (P1.0 si P1.1)

        PORT_Init ();                           // conecteaza perifericele la pini (UART0, UART1) si stabileste tipul pinilor

        LCD_Init();                                             // 2 linii, display ON, cursor OFF, pozitia initiala (0,0)
        KEYB_Init();
        UART0_Init(EIGHT_BIT, BAUDRATE_IO);         // initializare UART0  - conectata la USB-UART (P0.0 si P0.1)

        Timer0_Init();          // initializare Timer 0

        EA = 1;          // valideaza intreruperile
        SFRPAGE = LEGACY_PAGE;                                  // selecteaza pagina 0 SFR

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

        Afisare_meniu();                // Afiseaza meniul de comenzi

        while(1){                       // bucla infinita


                switch(STARE_NOD){
                        case 0:                                         // nodul este slave, asteapta mesaj de la master
                                switch(RxMesaj(ADR_NOD)){               // asteapta un mesaj de la master
                                        case TMO:
                                                Error("\n\rSL -> MS."); // anunta ca nodul curent devine master
                                                TIP_NOD = MASTER; // nodul curent devine master
                                                STARE_NOD = 2;    // trece in starea 2
                                                i = ADR_NOD;// primul slave va fi cel care urmeaza dupa noul master
                                                break;

                                        case ROK:
                                                Afisare_mesaj();
                                                STARE_NOD = 1;

                                                break;// a primit un mesaj de la master, il afiseaza si trece in starea 1
                                        case CAN:
                                                Error("\n\rMsj incom");
```

```
                                        break;      // afiseaza eroare Mesaj incomplet
                                    case TIP: Error("\n\rMsj nec");

                                        break;      // afiseaza eroare Tip mesaj necunoscut
                                    case ESC:       Error("\n\rEroare SC");

                                        break;      // afiseaza Eroare SC
                                    default:    Error("\n\rErr nec");

                                        break;      // afiseaza cod eroare necunoscut
                                }
                                break;

                            case 1:
                    // nodul este slave, transmite mesaj catre master
                                    found = 0;
                                    for(i = 0; i < NR_NODURI; i++){// cauta sa gaseasca un mesaj util de transmis
                                            if(retea[i].full == 1){
                                                found = 1;

                                                break;

                                            }

                                    }

                                    if(found == 1){                 // daca gaseste un nod i
                                        retea[i].bufbin.adresa_hw_dest = ADR_MASTER;// pune adresa HW
dest ADR_MASTER
                                        TxMesaj(i);                 // transmite mesajul catre nodul i
                                    }

                                    else{       // daca nu gaseste, construieste un mesaj in bufferul ADR_MASTER
                                        retea[ADR_MASTER].bufbin.adresa_hw_dest     = ADR_MASTER;
// adresa HW dest este ADR_MASTER
                                        retea[ADR_MASTER].bufbin.adresa_hw_src      = ADR_NOD;
                    // adresa HW src este ADR_NOD
                                        retea[ADR_MASTER].bufbin.tipmes = POLL_MES;         // tip
mesaj = POLL_MES
                                        TxMesaj(ADR_MASTER); // transmite mesajul din bufferul
ADR_MASTER
                                    }

                                    STARE_NOD = 0;  // trece in starea 0, sa astepte un nou mesaj de la master
                                break;

                            case 2:                             // tratare stare 2 si comutare stare
                                    do{
                                            i++;        // selecteaza urmatorul slave (incrementeaza i)
                                            if(i == NR_NODURI)
                                                                    i = 0;
                                        }
                                        while(i == ADR_NOD);
                                    retea[i].bufbin.adresa_hw_dest = i; // adresa HW dest este i
                                    if(retea[i].full == 1) TxMesaj(i); // daca in bufferul i se afla un mesaj util, il
transmite
                                    else{ // altfel, construieste un mesaj de interogare in bufferul i
                                            retea[i].bufbin.adresa_hw_src = ADR_NOD; // adresa HW
src este ADR_NOD
                                            retea[i].bufbin.tipmes = POLL_MES; // tip mesaj =
POLL_MES
                                            TxMesaj(i); // transmite mesajul din bufferul i
                                    }
                                    STARE_NOD = 3; // trece in starea 3, sa astepte raspunsul de la slave-ul i
                                    break;

                            case 3:                             // nodul este slave, asteapta mesaj de la master
                                switch(RxMesaj(i)){             // asteapta un raspuns de la slave i
                                    case TMO:
```

```
                                                                Error("\n\rTimeout ");              //
afiseaza eroare Timeout nod i
                                                                if(AFISARE){
                                                                        UART0_Putch( i + '0');
                                                                        LCD_Putch(i + '0');
                                                                }
                                                                        break;
                                                        case ROK:
                                                                Afisare_mesaj();

                                                                break;      // a primit un mesaj, il afiseaza
                                                        case ERI:
                                                                Error("\n\rErr incad");

                                                                break;      // afiseaza Eroare incadrare
                                                        case ERA:
                                                                Error("\n\rErr adr");

                                                                break;      // afiseaza Eroare adresa
                                                        case TIP:
                                                                Error("\n\rMsj nec");

                                                                break;      // afiseaza Tip mesaj
necunoscut
                                                        case OVR:
                                                                Error("\n\rErr suprap");

                                                                break;      // afiseaza Eroare
suprapunere
                                                        case ESC:
                                                                Error("\n \r Eroare SC");

                                                                break;      // afiseaza Eroare SC
                                                        default:
                                                                Error("\n\r Err nec");

                                                                break;      // afiseaza Eroare
necunoscuta
                                        }
                                        STARE_NOD = 2; // revine in starea 2 (a primit sau nu un raspuns de la slave, trece la
urmatorul slave)
                                        break;
                        }
                        UserIO();               // apel functie interfata
                }
}
```

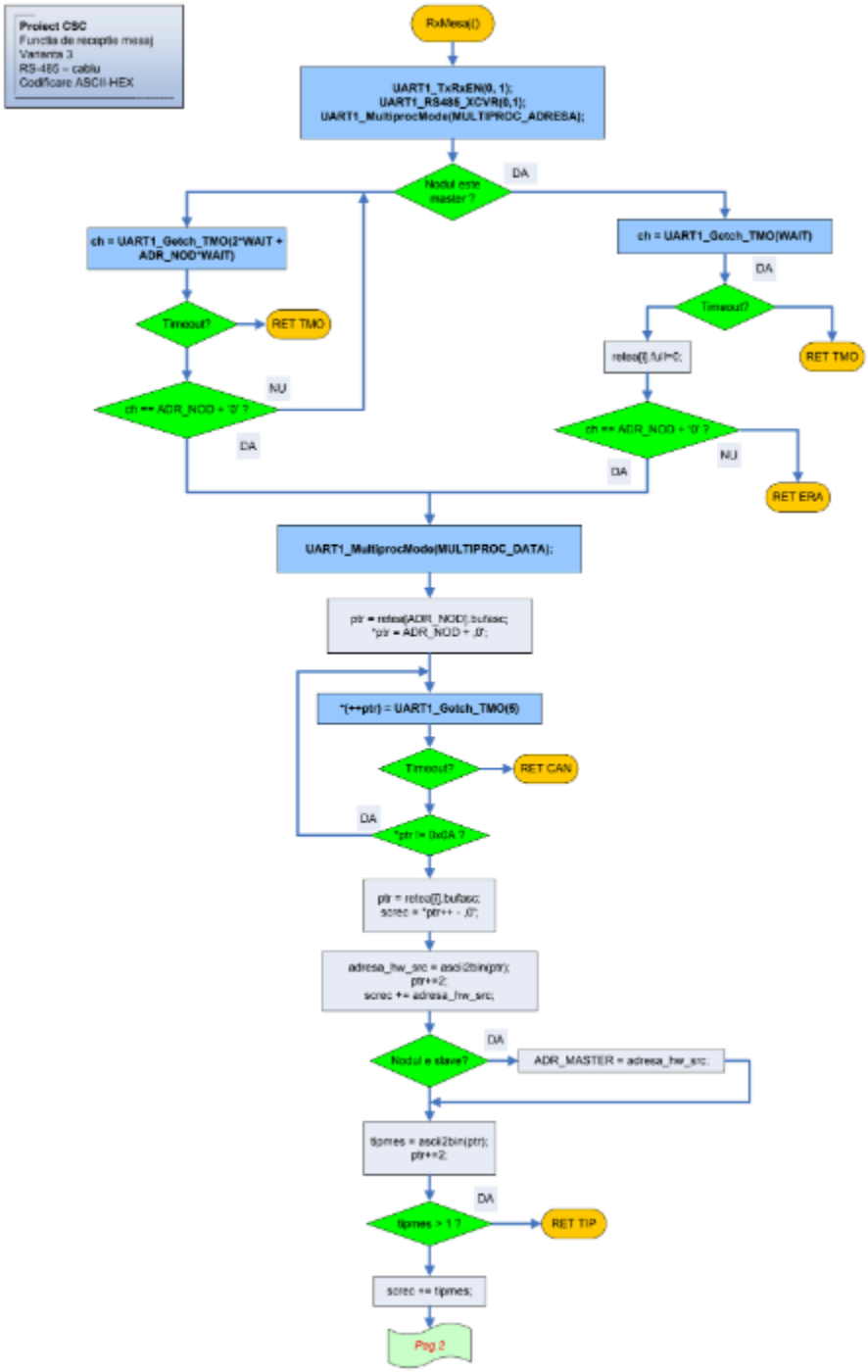
6. Pregătirea mesajelor pentru transmisie și servicii de transmisie

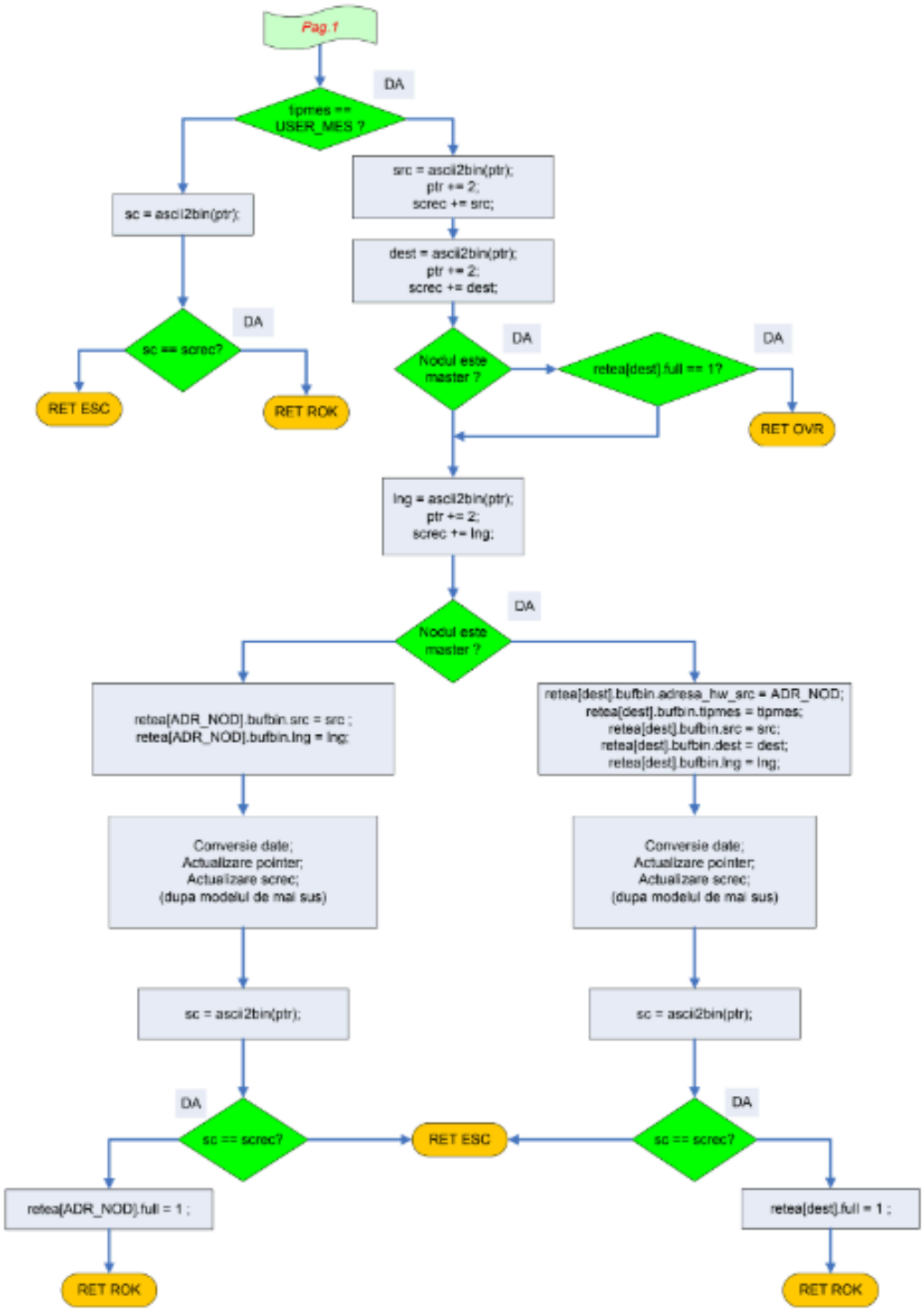


```
#include <c8051F040.h>         // declaratii SFR
#include <uart1.h>
#include <Protocol.h>
#include <UserIO.h>

extern unsigned char STARE_NOD; // starea initiala a nodului curent
extern unsigned char TIP_NOD; // tip nod initial: Nu Master, Nu Jeton

extern nod retea[];

extern unsigned char timeout; // variabila globala care indica expirare timp de asteptare eveniment
//*****************************************************************************************************
void TxMesaj(unsigned char i); // transmisie mesaj destinat nodului i
void bin2ascii(unsigned char ch, unsigned char *ptr); // functie de conversie octet din binar in ASCII HEX

//*****************************************************************************************************
void TxMesaj(unsigned char i){ // transmite mesajul din buffer-ul i
        unsigned char sc, *ptr, j;

        if(retea[i].bufbin.tipmes ==POLL_MES) // daca este un mesaj de interogare (POLL=0)
        {
                        // calculeaza direct sc
                        retea[i].bufbin.sc=retea[i].bufbin.adresa_hw_dest + retea[i].bufbin.adresa_hw_src ;
        }   // altfel...
        else
        {
                        sc=retea[i].bufbin.adresa_hw_dest;   // initializeaza SC cu adresa HW a nodului destinatie
                        sc+=retea[i].bufbin.adresa_hw_src;   // ia in adresa_hw_src
                        sc+=retea[i].bufbin.tipmes; // ia in calcul tipul mesajului
                        sc+=retea[i].bufbin.src; // ia in calcul adresa nodului sursa al mesajului
                        sc+=retea[i].bufbin.dest;  // ia in calcul adresa nodului destinatie al mesajului
                        sc+=retea[i].bufbin.lng;
                        for(j=0;j<retea[i].bufbin.lng;j++)    // ia in calcul lungimea datelor
                        {
                                sc+=retea[i].bufbin.date[j];   // ia in calcul datele
                // stocheaza suma de control
                        }
                        retea[i].bufbin.sc=sc;
        }


        ptr=retea[i].bufasc; // initializare pointer pe bufferul ASCII
        *ptr++=retea[i].bufbin.adresa_hw_dest+'0'; // pune in bufasc adresa HW dest + '0'
        bin2ascii(retea[i].bufbin.adresa_hw_src,ptr); // pune in bufasc adresa HW src in ASCII HEX
        ptr+=2;
        bin2ascii(retea[i].bufbin.tipmes,ptr); // pune in bufasc tipul mesajului
        ptr+=2;
                if(retea[i].bufbin.tipmes ==USER_MES) // daca este un mesaj de date (USER_MES)
        {
                        bin2ascii(retea[i].bufbin.src,ptr); // pune in bufasc src
                        ptr+=2;
                        bin2ascii(retea[i].bufbin.dest,ptr); // pune in bufasc dest
                        ptr+=2;
                        bin2ascii(retea[i].bufbin.lng,ptr); // pune in bufasc lng date
                        ptr+=2;
                        for(j=0;j<retea[i].bufbin.lng;j++)
                        {
                                bin2ascii(retea[i].bufbin.date[j],ptr);
                                ptr+=2; // pune in bufasc datele
                        }
        }
        bin2ascii(retea[i].bufbin.sc,ptr); // pune in bufasc SC
        ptr+=2;
        // de vazut

        *ptr++=0x0d; // pune in bufasc CR
        *ptr++=0x0a; // pune in bufasc LF
```

```
		UART1_MultiprocMode(MULTIPROC_ADRESA); // urmeaza tranmisia octetului de adresa
		UART1_RS485_XCVR(1,1); // validare Tx si Rx RS485
		UART1_TxRxEN(1, 1);

		ptr=retea[i].bufasc; // reinitializare pointer
		UART1_Putch(*ptr); // transmite adresa HW dest

		if(UART1_Getch_TMO(2)!=*ptr) // daca caracterul primit e diferit de cel transmis ...
		{
				UART1_TxRxEN(0,0);
				UART1_RS485_XCVR(0,0); // dezactivare Tx RS485
				Error("\n\rColiziune!"); // afiseaza Eroare coliziune
				Delay(1000);
				return; // asteapta 1 secunda // termina transmisia (revine)
		}

		UART1_MultiprocMode(MULTIPROC_DATA); // urmeaza tranmisia octetilor de date
		UART1_TxRxEN(1, 0);
		do{
			UART1_Putch(*(++ptr));
		}
		while(*ptr!=0x0d);		// transmite restul caracterelor din bufferul ASCII

		UART1_Putch(*(++ptr));
		UART1_TxRxEN(1,1);

		if(TIP_NOD!=MASTER)
		{
			retea[i].full=0;	// slave-ul considera acum ca a transmis mesajul
		}
		UART1_Getch(0); // asteapta terminarea transmisie ultimului caracter
		UART1_TxRxEN(0, 0);
		UART1_RS485_XCVR(0, 0); // dezactivare Tx RS485
		return;

}

//*************************************************************************************************************
void bin2ascii(unsigned char ch, unsigned char *ptr){ // converteste octetul ch in doua caractere ASCII HEX puse la adresa ptr
unsigned char first, second;
first = (ch & 0xF0)>>4; // extrage din ch primul digit
second = ch & 0x0F; // extrage din ch al doilea digit
if(first > 9) *ptr++ = first - 10 + 'A'; // converteste primul digit daca este litera
else *ptr++ = first + '0'; // converteste primul digit daca este cifra
  if(second > 9) *ptr++ = second - 10 + 'A'; // converteste al doilea digit daca este litera
else *ptr++ = second + '0'; // converteste al doilea digit daca este cifra
}
```

7. Recepţia mesajelor – descriere, schema logică şi implementare




Pag.1
tipmes == USER_MES ?
DA
sc = ascii2bin(ptr);
src = ascii2bin(ptr);
ptr += 2;
screc += src;
dest = ascii2bin(ptr);
ptr += 2;
screc += dest;
sc == screc?
DA
RET ESC
RET ROK
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;
(dupa modelul de mai sus)
Conversie date;
Actualizare pointer;
Actualizare screc;
(dupa modelul de mai sus)
sc = ascii2bin(ptr);
sc = ascii2bin(ptr);
DA
sc == screc?
RET ESC
sc == screc?
DA
retea[ADR_NOD].full = 1 ;
retea[dest].full = 1 ;
RET ROK
RET ROK

```
#include <c8051F040.h>         // declaratii SFR
#include <uart1.h>
#include <Protocol.h>
#include <UserIO.h>

extern nod retea[];                                                    // reteaua Master-Slave, cu 5 noduri

extern unsigned char STARE_NOD;                    // starea initiala a nodului curent
extern unsigned char TIP_NOD;                                  // tip nod
extern unsigned char ADR_MASTER;     // adresa nodului master

extern unsigned char timeout;                // variabila globala care indica expirare timp de asteptare eveniment
//***************************************************************************************************
unsigned char RxMesaj(unsigned char i);                                        // primire mesaj de la nodul i
        unsigned char ascii2bin(unsigned char *ptr);                                    // functie de conversie 2
caractere ASCII HEX in binar

        //***************************************************************************************************
        unsigned char RxMesaj(unsigned char i){                                    // receptie mesaj

        unsigned char j, sc, ch, adresa_hw_src, screc, src, dest, lng, tipmes, *ptr;

        UART1_TxRxEN(0, 1);
                                                                                                    // dezactivare Tx,
validare RX UART1
        UART1_RS485_XCVR(0,1);
                                                                // dezactivare Tx, validare RX RS485
        UART1_MultiprocMode(MULTIPROC_ADRESA);
        // receptie doar octeti de adresa

        if(TIP_NOD == MASTER){
                        // Daca nodul este master...
                ch = UART1_Getch_TMO(WAIT);
                        // M: asteapta cu timeout primul caracter al raspunsului de la slave
                if (timeout) return TMO;
                  // M: timeout, terminare receptie
                else {
                                        // M: raspunsul de la slave vine, considera ca mesajul anterior a fost transmis
cu succes
                        retea[i].full = 0;
                        if (ch != ADR_NOD + '0') return ERA;
// M: adresa HW ASCII gresita, terminare receptie
                }

        }
        else{
                                        // Daca nodul este slave...
                do{
                        ch = UART1_Getch_TMO(2*WAIT + ADR_NOD*WAIT);        // S: asteapta cu timeout primirea
primului caracter al unui mesaj de la master
                        if (timeout) return TMO;          // S: timeout, terminare receptie
                }while(ch != ADR_NOD + '0');  // S: iese doar cand mesajul era adresat acestui slave
        }

        UART1_MultiprocMode(MULTIPROC_DATA);
// receptie octeti de date

        ptr = retea[ADR_NOD].bufasc;
                // M+S: pune in bufasc restul mesajului ASCII HEX
        *ptr = ADR_NOD + '0';

        do{
                *(++ptr) = UART1_Getch_TMO(5);
```

```
                        if(timeout) return CAN;
                          // M+S: timeout, terminare receptie
                }while(*ptr != 0x0A);



                //ptr = retea[i].bufasc;
                ptr = retea[ADR_NOD].bufasc;              // M+S: reinitializare pointer in bufferul ASCII

                                                                                        // M+S: initializeaza screc cu adresa HW dest
                screc = *ptr++ - '0';

                adresa_hw_src = ascii2bin(ptr);                         // M+S: determina adresa HW src
                ptr+=2;
                screc += adresa_hw_src;                                         // M+S: aduna adresa HW src



                if(TIP_NOD == SLAVE){
                        ADR_MASTER = adresa_hw_src;                                                     // Slave actualizeaza adresa
Master
                }

                tipmes = ascii2bin(ptr);          // M+S: determina tipul mesajului
                ptr+=2;

                if(tipmes > 1)return TIP;         // M+S: cod functie eronat, terminare receptie

                screc += tipmes;// M+S: ia in calcul in screc codul functiei

                if(tipmes == USER_MES){
                                        // M+S: Daca mesajul este unul de date

                                        src = ascii2bin(ptr);
                                // M+S: determina sursa mesajului
                                        ptr += 2;
                                        screc += src;
                                                // M+S: ia in calcul in screc adresa src

                                        dest = ascii2bin(ptr);
                                          // M+S: determina destinatia mesajului
                                        ptr += 2;
                                        screc += dest;
                                                // M+S: ia in calcul in screc adresa dest

                                        if(TIP_NOD == MASTER){
                                                                                                // Daca nodul este master...
                                                if(retea[dest].full == 1) return OVR;
                                                // M: bufferul destinatie este deja plin, terminare receptie
                                        }

                                        lng = ascii2bin(ptr);
                                                                                                                        // M+S:
determina lng
                                        ptr += 2;
                                        screc += lng;

                                        if(TIP_NOD == MASTER){

// Daca nodul este master...
                                                retea[dest].bufbin.adresa_hw_src = ADR_NOD;
                                                                                                // M: stocheaza in bufbin
adresa HW src
                                                retea[dest].bufbin.tipmes = tipmes;
                                                                                                        // M: stocheaza in
bufbin tipul mesajului
                                                retea[dest].bufbin.src = src;
                                                                                                        // M: stocheaza in
bufbin adresa nodului sursa al mesajului
```

```
                                        retea[dest].bufbin.dest = dest;
                                                                                                        // M: stocheaza in
bufbin adresa nodului destinatie al mesajului
                                        retea[dest].bufbin.lng = lng;
                                                                                                         // M: stocheaza lng


                                        for(j=0;j<retea[dest].bufbin.lng;j++)
                                        {

                                                retea[dest].bufbin.date[j]=ascii2bin(ptr);

                                                ptr+=2;
                                                                        // pune in bufasc datele
                                                screc = screc + retea[dest].bufbin.date[j];
                                        }


                                        sc = ascii2bin(ptr);
                                                                                                                // M:
determina suma de control

                                        if(sc == screc){
                                                retea[dest].full = 1 ;
                                                                                                                 // M:
pune sc in bufbin
                                                return ROK;

                        // M: mesaj corect, marcare buffer plin
                                        }
                                        else return ESC;

// M: eroare SC, terminare receptie

                                }
                                else {
                                retea[ADR_NOD].bufbin.src = src ;                                       // S: stocheaza la
destsrc codul nodului sursa al mesajului
                                retea[ADR_NOD].bufbin.lng = lng;                                        // S: stocheaza lng

                                for(j=0;j<retea[ADR_NOD].bufbin.lng;j++)
                                {

                                        retea[ADR_NOD].bufbin.date[j]=ascii2bin(ptr);

                                        ptr+=2;
                                                                // pune in bufasc datele
                                        screc = screc + retea[ADR_NOD].bufbin.date[j];
                                }
                                                                                                                // S: ia in
calcul in screc octetul de date

                                sc = ascii2bin(ptr);
                                                                                                        // S: determina
suma de control
                                if(sc == screc){
                                                                                                                 // S:
mesaj corect, marcare buffer plin
                                        retea[ADR_NOD].full = 1 ;
                                        return ROK;
                                }
                                else return ESC;
                                                                                                                 // S:
eroare SC, terminare receptie
                                }
        }
                        else{
```

```
                        sc = ascii2bin(ptr);
                                                                                                                // S: determina
suma de control
                        if(sc == screc) return ROK;
                        else return ESC; // M+S: eroare SC, terminare receptie
                        }
                        //return TMO;
                }

//*************************************************************************************************************
unsigned char ascii2bin(unsigned char *ptr){                    // converteste doua caractere ASCII HEX de la adresa ptr
                unsigned char bin;

                if(*ptr > '9') bin = (*ptr++ - 'A' + 10) << 4; // contributia primului caracter daca este litera
                else bin = (*ptr++ - '0') << 4;                                                                 //
contributia primului caracter daca este cifra
                if(*ptr > '9') bin  += (*ptr++ - 'A' + 10);             // contributia celui de-al doilea caracter daca este litera
                else bin += (*ptr++ - '0');
                // contributia celui de-al doilea caracter daca este cifra
                return bin;
}
```

8. Concluzii

Ca și echipă, ne-am împărțit taskurile încă din prima săptămână de proiect și fiecare a cerut ajutorul când întâmpina dificultăți.

Pe parcursul laboratorului, am realizat două testări : una în simulare, iar cealaltă la nivel fizic.

Testarea în simulare am realizat-o folosind mediul $\mu Vision5$, utilizând ferestrele UART1 și UART2.Rezultatele au fost cele dorite.

Dezvoltarea aplicației la nivel fizic a fost dificilă. Ne-am confruntat cu o comunicare nesatisfăcătoare între nodul master și cel slave, însă am reușit să remediem problema, programul fiind funcționabil.

Bibliografie

- ★ BIG8051 User Manual by MikroElektronika
- ★ CSC - Microsistemul BIG8051 – MikroElektronika
- ★ CSC - Mediul integrat de dezvoltare Keil PK51
- ★ Suportul de curs - “Comunicaţii în sisteme de conducere”