**Implementarea jocului X şi 0 pe placa Dragon12 – Plus2, utilizând comunicaţia prin portul serial**

Talpeș Alexandru-Paul

Stoica Talida Maria

**Sisteme incorporate - An III - CTI**

*2. Enunț*

**Implementarea jocului X şi 0 pe placa Dragon12 – Plus2, utilizând comunicaţia prin portul serial (2 studenţi)**

**Caracteristici:**

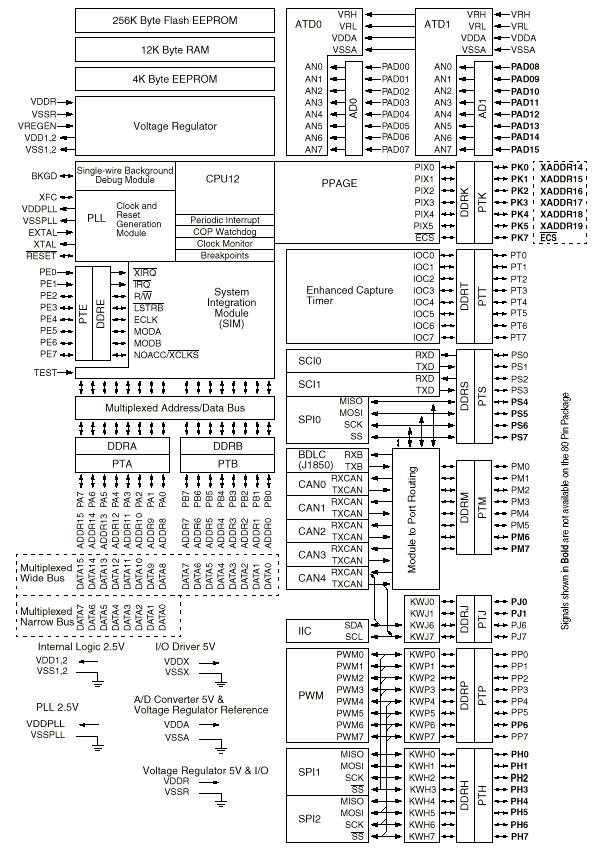
* Cei doi jucători sunt reprezentaţi de utilizatorul PC-ului şi de placa Dragon12.
* Programul va implementa următoarele funcţii:
* Afişare mesaj de întâmpinare cu regulile jocului; poziţiile de pe tabla de joc vor fi notate de la 1 la 9;
* Preluare alegere utilizator – utilizatorul va indica o poziţie de la 1 la 9;
* Alegere Dragon12 – programul va genera aleatoriu o alegere validă;
* Verificare victorie sau remiză – dacă este îndeplinită una dintre condiţii se va afişa un mesaj corespunzător şi jocul va fi reiniţializat;
* Afişare tablă de joc – după fiecare alegere efectuată, se va afişa tabla de joc.
* Derularea jocului se va realiza sub forma unui dialog în monitorul pentru interfața serială, utilizând un program pentru ascultarea portului serial, precum Tera Term, PuTTY, etc.

***3. Descrierea plăcii de dezvoltare utilizate***

Placa Dragon12-Plus conține instalat un microcontroller MC9S12DP256CCPV sau MC9S12DG256CVPE. MC9S12DP256 a fost eliminat de către Freescale fiind înlocuit cu MC9S12DG256. Acesta din urmă prezintă aceleași caracteristici, între cele două fiind o singură diferență: numărul de porturi CAN - DG256 are 2 porturi CAN, iar DP256 are 5 porturi CAN.

**Principalele caracteristici ale lui MC9S12DG256 sunt:**

* un puternic CPU pe 16 biți
* 256K bytes de memorie flash
* 12K bytes de memorie RAM
* 4 K bytes de memorie EEPROM
* Porturi SCI
* Porturi SPI
* Porturi CAN 2.0
* Interfață I2C
* Timere pe 16 biți cu 8 canale
* PWM pe 8biți cu 8 canale/16 biți cu 4 canale.
* Convertor A/D cu 16 canale pe 10 biți
* O viteză de 25 MHz a magistralei pe un chip Phase Lock Loop
* BDM(backround debug mode) pentru programare și depanare
* Pachet LQPF(Quad Flat Package) cu 112 pini ce oferă până la 91 I/O într-un footprint



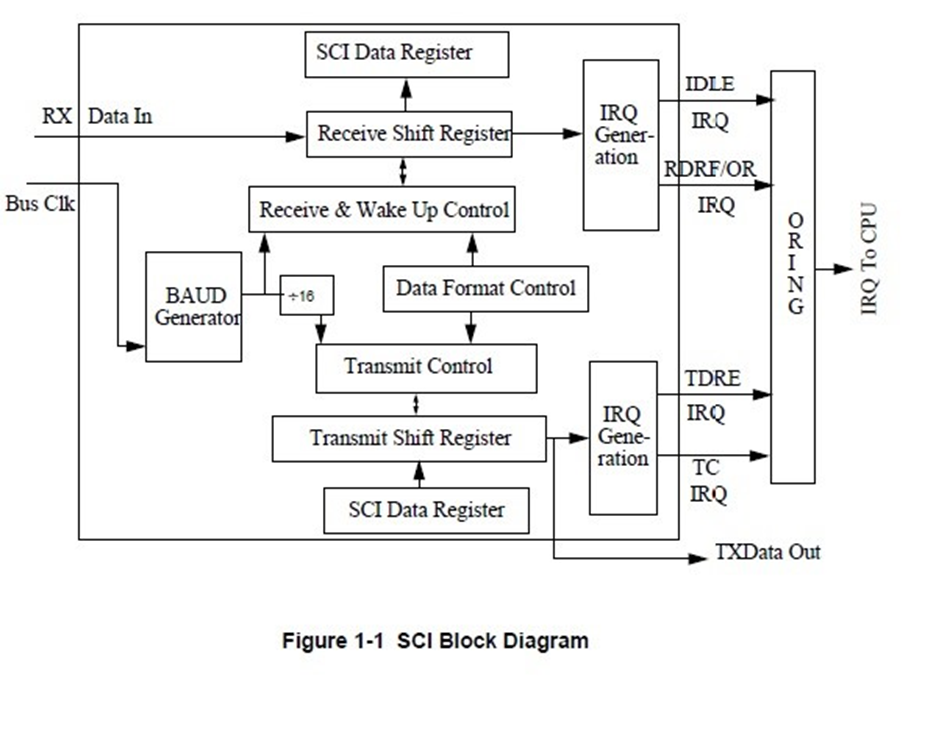
**Caracteristici hardware ale microcontrolerului MC9S12DG256**

* O interfață USB selectabilă pentru SCI0 sau SCI1
* Un LED RGB
* Un port de comunicație RS485
* Un capacitor DS1307 RTC de rezervă inclus pentru a testa interfața I2C
* Un port de expandare I2C pentru a interfața dispozitivele externe I2C
* Port CAN
* Un port de expandare SPI pentru a interfața dispozitivele externe SPI
* Un convertor digital analogic pentru a testa interfața SPI și a genera waveformuri analogice
* 4 servo-controllere robot cu un bloc terminal pentru 5V externi.
* afișaj LED cu 7 segmente cu 4 cifre pentru a învăța tehnici de multiplexare
* 8 leduri
* Switch-uri DIP cu 8 poziții
* 4 switch-uri de tip push-button
* Regulatoare de 3,3 V și 5V
* Microfon condus de timer,sau de convertorul analog digital sau un semnal PWM pentu alarme sau aplicații muzicale.
* O punte H duală care controlează două motoare de curent continuu sau un motor de tip Stepper
* Indicatoare Led Power-On de 3,3V sau 5V
* Detector IR
* Un conector BDM ce va fi conectat cu un BDM de la multiplii distribuitori pentru depanare
* Modul BDM POD pentru a programa alte plăci HCS 12
* Renunțarea la switch pentru a stopa programul când programul este blocat într-o buclă moartă
* Modul de switch pentru a selecta cele 4 moduri de operare:EVB,Jump-to EEPROM,BDM POD și Bootloader
* Tastatură 4\*4
* Form C relay output rated at 3A/30V or 1A/125V
* O interfață de acelerometru X-Y-Z sau o interfață senzor de măsurare a distanței GP2-D12 pentru măsurarea distanței
* Un potențiometru(condensator de reglare) pentru intrări analogie
* Senzor de temperatură
* Senzor de lumină
* Port de comunicație pentru VGA
* Female headere pentru a furniza o distanță mai scurtă de la bread bord la fiecare pin I/O al microcontrollerului.
* Dimensiunea plăcii PC este de 8.4" X 5.3"

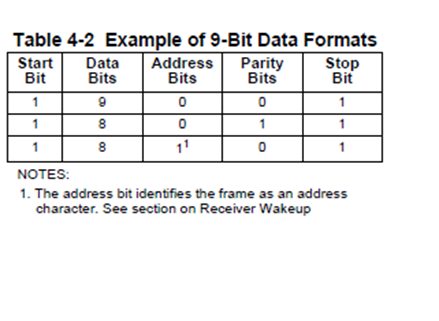
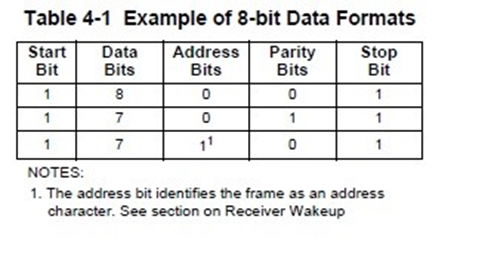
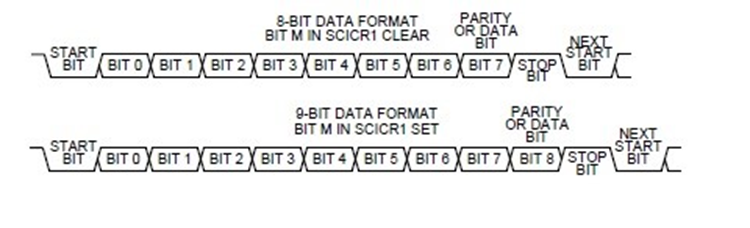
*3. Descrierea detaliată a modulelor microcontrolerului care au fost implicate în*

*realizarea proiectului*

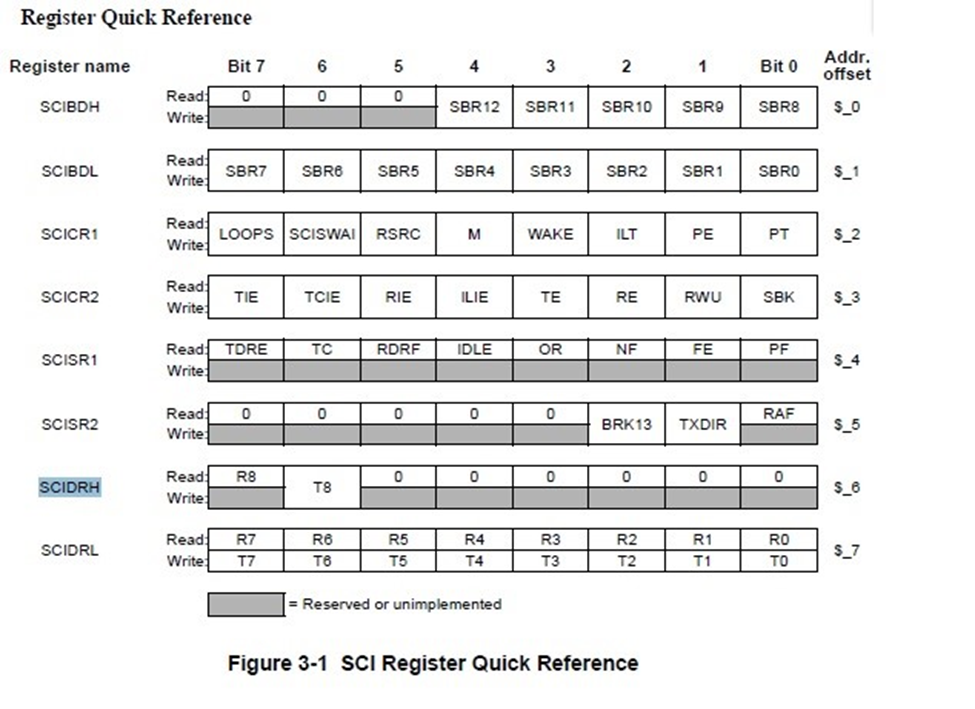
**Interfaţa serială SCI**



* Operare full duplex;
* Format pe 8 sau 9 biţi, plus biţii de cadrare şi bitul de paritate;
* Transmiţătorul şi receptorul pot fi validaţi separat;
* Generare de întrerupere la: transmitere încheiată, preluare caracter, eroare de ritm, de cadrare, de paritate, zgomote;
* Formatul datelor:



* Este controlată de SFR – uri; ex.: SCI Data Registers



**SCIBDH/L(SCI BAUD RATE Register)**

Citirea: oricând.

Scrierea: oricând.

SCI BAUD RATE Register este utilizat de cãtre counter pentru a determina baud rate-ul lui SCI.

BR-este conținutul lui SCI Baud Register(SBR12:SBR0). Poate lua valori de la 1 la 8191.

SCIBDL și SCIBDH trebuie sã fie scriși pentru a returna datele corect.

SBR12:SBR0-determina valoarea baud rate-ului lui SCI.

Baud Rate Generator este dezactivat pânã când TE sau RE este setat prima datã dupã reset. Baud Rate Generator este dezactivat când BR=0.

Scrierea în SCIBDH nu are niciun efect dacã nu s-a scris în SCIBDL, din moment ce scrierea în SCIBDH pune data într-o locație temporarã pânã când SCIBDL este scris.

**SCI Control Register 1 (SCICR1)**

Scrierea: oricând.

LOOPS - Loop Select Bit

1 = operație de loop activată

0 = operație normală activată

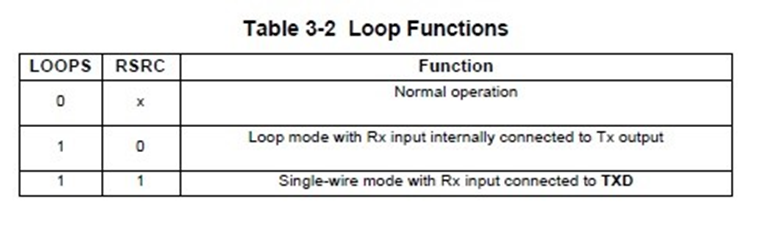
Inputul de recepție este determinat de bitul RSRC.

SCISWAI — SCI Stop in Wait Mode Bit

SCISWAI dezactivează SCI în modul de așteptare

1 = SCI dezactivat în modul de așteptare

0 = SCI activat în modul de așteptare

RSRC — Receiver Source Bit 

Când LOOPS=1, bitul RSRC determină sursa pentru receiver shift register input.

1 = Receiver input connected externally to transmitter

0 = Receiver input internally connected to transmitter output

M — Data Format Mode Bit

MODE determină dacă, caracterele de date sunt pe 8, respectiv 9 biți.

1 = un bit de start, 9 biți de date, un bit de stop

0 = un bit de start, 8 biți de date, un bit de stop

WAKE — Wakeup Condition Bit

WAKE determină ce condiție trezește SCI-ul: un 1 logic (address mark) la cel mai semnificativ bit al unui caracter recepționat sau o condiție de idle pe RXD.

1 = Address mark wakeup

0 = Idle line wakeup

ILT — Idle Line Type Bit

ILT determină când receptorul începe să numere 1 logic ca și biți de caractere idle. Numărarea începe fie după bitul de start, fie după bitul de stop. Dacă numărarea începe după bitul de start, atunci un șir de 1 logic care precede bitul de stop poate cauza o recunoaștere falsă a unui caracter de idle. Începerea numărării după bitul de stop rezolvă această problemă, dar necesită transmisii sincronizate.

1 = Idle character bit count begins after stop bit

0 = Idle character bit count begins after start bit

PE — Parity Enable Bit

PE activează funcția de paritate.Când e activată se inserează un bit de paritate la bitul cel mai semnificativ.

1 = Parity function enabled

0 = Parity function disabled

PT — Parity Type Bit

PT determină dacă SCI generează și verifică o paritate pară sau impară. La o paritate pară, un număr par de 1 resetează bitul de paritate și un număr impar de 1 setează bitul de paritate. La o paritate impară,un număr impar de 1 resetează bitul de paritate și un număr par de 1 setează bitul de paritate.

1 = paritate impară

0 = paritate pară

**SCI Control Register 2(SCICR2)**

Citirea: oricând.

Scrierea: oricând.

TIE — Transmitter Interrupt Enable Bit

TIE activează flagul de transmisie a registrului de date (TDRE) pentru a genera cereri de întrerupere.

1 = TDRE interrupt requests enabled

0 = TDRE interrupt requests disabled

TCIE — Transmission Complete Interrupt Enable Bit

TCIE activează flagul de transmisie completă (TC) pentru a genera cereri de întrerupere .

1 = TC interrupt requests enabled

0 = TC interrupt requests disabled

RIE — Receiver Full Interrupt Enable Bit

RIE activează receive data register full flag, RDRF, sau overrun flag, OR, pentru a genera cereri de întrerupere.

1 = RDRF and OR interrupt requests enabled

0 = RDRF and OR interrupt requests disabled

ILIE — Idle Line Interrupt Enable Bit

ILIE activează idle line flag, IDLE,pentru a genera cereri de înterupere

1 = IDLE interrupt requests enabled

0 = IDLE interrupt requests disabled

TE — Transmitter Enable Bit

TE activează transmițătorul SCI și configurează pinul TXD ca fiind controlat de SCI .

1 = Transmitter enabled

0 = Transmitter disabled

RE — Receiver Enable Bit

RE activează receptorul RE.

1 = Receiver enabled

0 = Receiver disabled

RWU — Receiver Wakeup Bit Standby state

1 = RWU activează funcția de wakeup și inhibă cereri viitoare de întrerupere din partea receptorului.În mod normal,hardware-ul trezește automat receptorul resetâmd RWU.

0 = Normal operation.

SBK — Send Break Bit

1 = Transmit break characters

0 = No break characters

**SCI Status Register 1 ( SCISR1)**

Citirea: oricând.

Scrierea: irelevant, nu are effect.

TDRE — Transmit Data Register Empty Flag

1 = Byte transferred to transmit shift register; transmit data register empty

0 = No byte transferred to transmit shift register

TC — Transmit Complete Flag

1 = No transmission in progress

0 = Transmission in progress

RDRF — Receive Data Register Full Flag

1 = Received data available in SCI data register

0 = Data not available in SCI data register

IDLE — Idle Line Flag

1 = Receiver input has become idle

0 = Receiver input is either active now or has never become active since the IDLE flag was last cleared

OR — Overrun Flag

1 = Overrun

0 = No overrun

NF — Noise Flag

1 = Noise

0 = No noise

FE — Framing Error Flag

1 = Framing error

0 = No framing error

PF — Parity Error Flag

1 = Parity error

0 = No parity error

**SCI Status Register 2(SCISR2)**

Citirea: oricând.

Scrierea: oricând - scrierea accesează SCI status register 2; are effect doar scrierea pe biții TXDIR și BRK13(SCISR2[1] & [2]).

BRK13 — Break Transmit character length

1 = Break character is 13 or 14 bit long

0 = Break Character is 10 or 11 bit long

TXDIR — Transmitter pin data direction in Single-Wire mode.

1 = TXD pin to be used as an output in Single-Wire mode

0 = TXD pin to be used as an input in Single-Wire mode

RAF — Receiver Active Flag

1 = Reception in progress

0 = No reception in progress

**SCI Data Registers (SCIDRH/L)**

Citirea: oricând – citirea accesează SCI receive data register

Scrierea: oricând – scrierea accsează SCI transmit data register; scrierea pe R8 nu are effect.

R8 — Received Bit 8

R8 este al nouălea bit de date recepționat când SCI este configurat pentru un format de 9 biți(M=1).

T8 — Transmit Bit 8

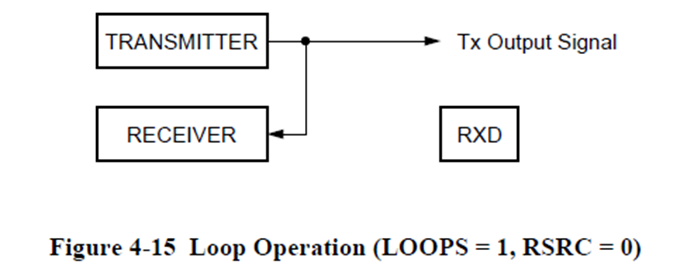
T8 este al nouălea bit de date transmis când SCI este configurat pentru un format de 9 biți(M=1).

R7-R0 — Biții de recepție de la 7 la 0 pentru formate de date pe 9 sau 8 biți.

T7-T0 —Biții de transmisie de la 7 la 0 pentru formate de date pe 9 sau 8 biți.

Terminale: RxD, TxD.

**Modul LOOP**:



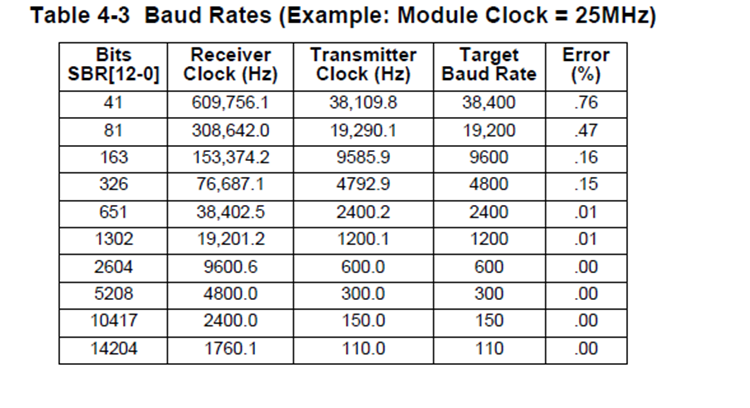
-în scop de test

-ieșirea transmițătorului merge la intrarea receptorului.Intrarea Rx este deconectată de la SCI.

Se activează operația de loop setând biții de loop și resetând bitul RSRC din SCI CR1.

Setarea biților de loop dezactivează calea de la semnalul de input Rx la receptor.Resetarea bitului RSRC conectează ieșirea transmițătorului la intrarea receptorului.Atât transmițătorul,cât și receptorul trebuie să fie activați (TE=1 și RE=1).

**Rata de transfer:**



-Eroarea apare din divizarea cu o valoare întreagă a valorii frecvenței tactului.

Relația: SCI baud rate = SCI module clock / (16 \* SCIBR[12:0])

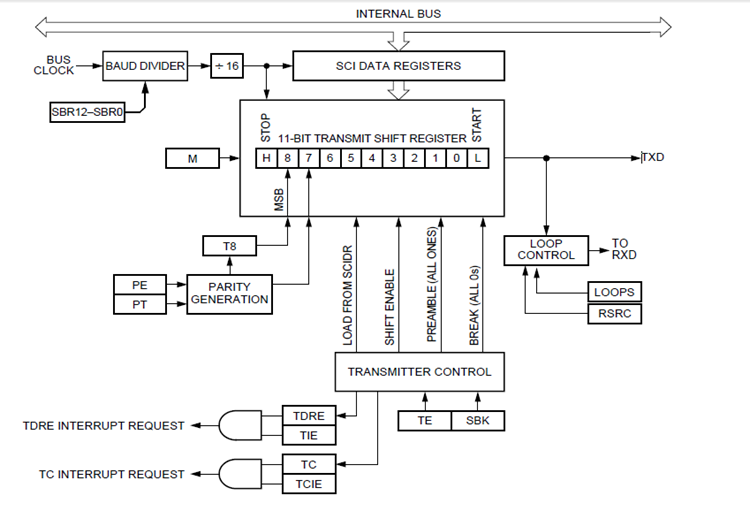
SCI module clock=4MHz

SCI baud rate=9600=0x2580 =>BR=26=0x1A;

SCI0BDH=0x00;

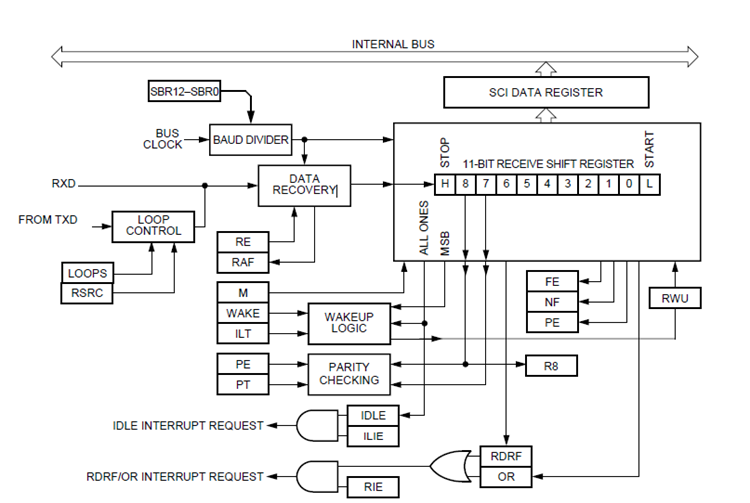
SCI0BDL=0x1A;

**Transmisia:**



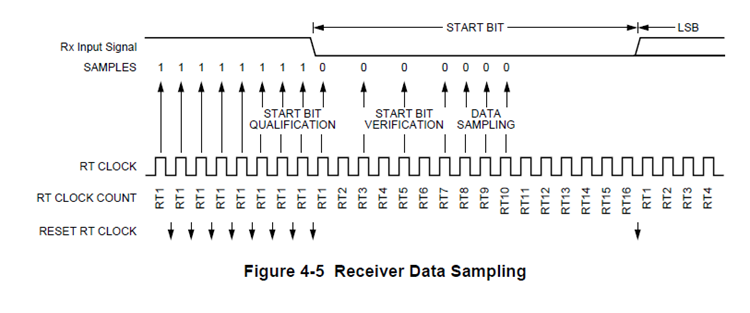
-M stabileşte formatul, cu 8 sau 9 biţi, PE validează paritatea, PT stabileşte tipul parităţii, TC arată încheierea transmisiei, TDRE cere date de la procesor, TIE şi TCIE validează cererile corespunzătoare de întrerupere, TE validează transmisia iar SBK permite generarea caracterului Break;

**Recepția:**

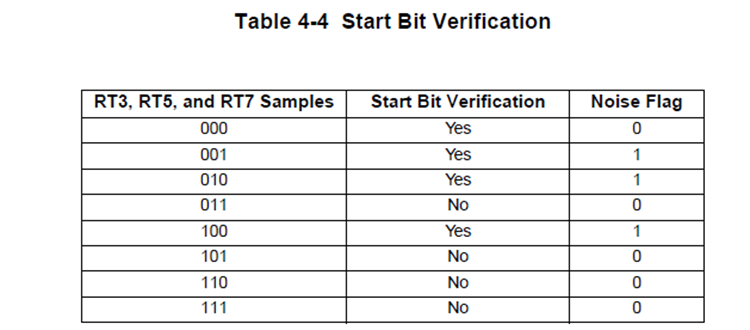


-RE validează recepţia, RAF arată recepţie în lucru, OR (overrun), FE (frame), NF (noise), PE (parrity) arată erori, RDRF arată caracter disponibil, RWU validează operaţia wakeup, WAKE stabileşte cine provoacă wakeup (adresa sau linia idle), ILIE, RIE validează cereri de întrerupere;

**Citirea liniei:**



-RT este tactul de recepţie cu frecvenţa Baud rate x 16; pentru a găsi bitul START caută un 0 precedat de trei de 1; apoi numără pînă la 16;



-Dacă verificarea bitului de start nu a avut succes, clockul RT este resetat și începe o nouă căutare pentru bitul de start.

*4. Program – utilizare și descriere*

* **Utilizarea programului**

La rularea programului se foloseste ca și IDE mediul de dezvoltare CodeWarrior, respectiv TeraTerm,un terminal ce asigură transmiterea,respectiv recepția serială a datelor. După rularea codului, se schimbă poziția switchului de pe placă, se deschide terminalul și se resetează placa.

Îndată ce terminalul TeraTerm este deschis se va afișa dispunerea inițială a tablei de joc și i se va cere utilizatorului să selecteze poziția pe care dorește să își plaseze simbolul(X) de pe tastatura plăcii Dragon12. Apoi placa Dragon12 generează aleator un număr între 1 și 9, face verificări ca acea poziție să fie validă(liberă) și își plasează simbolul(0). Apoi se reia procedeul.

Jocul se termină atunci când unul dintre cei doi participanți, jucătorul sau computerul, reușește să plaseze 3 simboluri de același fel(X, respective 0) în linie. Jocul va arăta un mesaj de final, apoi va relua de la început după câteva secunde.

* **Descrierea algoritmului**

La inceputul jocului, se setează regiștrii SCI0BDH/L, corespunzători unui baud rate = 9600, și SCI0CR1/2(transmițător și receptor activat).

În interiorul unei bucle infinite se realizează o succesiune principal de apeluri de funcții: disp\_matrix(), ce printează matricea curentă, get\_player\_move(), ce receptează de pe serial butonul apăsat pe placa de dezvoltare Dragon12 și check(), care verifică constant terminarea jocului.

* **Codul**

#include <hidef.h> /\* common defines and macros \*/

#include "mc9s12dg256.h" /\* derivative-specific definitions \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#pragma LINK\_INFO DERIVATIVE "mc9s12dg256b"

void MSDelay(unsigned int itime){

unsigned int i; unsigned int j;

for(i=0;i<itime;i++)

for(j=0;j<4000;j++); //1 msec. tested using Scope

}

int turn;

char matrix[3][3];

unsigned char column, row, myRow; //to read the keypad

char check(void);

void init\_matrix(void);

void get\_player\_move(void);

void get\_computer\_move(void);

void disp\_matrix(void);

void printing\_serial(char \*p);

void printing\_char(char p) ;

void readKeypad(void);

void main(void) {

int state = 0;

char \*ptr = &matrix[0][0];

char done;

int turn = 1;

char \*stringPlayer = "Your move.\r\n";

SCI0BDH = 0x00;

SCI0BDL = 26;

SCI0CR1 = 0x00;

SCI0CR2 = 0x0C;

DDRA = 0x0F; //set as output

for(;;){

if (state == 0){

init\_matrix();

disp\_matrix();

state = 1;

}

if (state == 1){

printing\_serial(stringPlayer);

do {

disp\_matrix();

get\_player\_move();

turn++;

done = check(); /\* check winner \*/

if(done != ' ')

break; /\* winner!\*/

get\_computer\_move();

disp\_matrix();

turn++;

done = check(); /\* check winner \*/

} while(done == ' ');

state = 2;

}

if (state == 2){

if(done == 'X')

printing\_serial("You won!\r\n");

else

printing\_serial("Computer won!!!!\r\n");

disp\_matrix(); /\* show final positions \*/

break;

}

}

}

void disp\_matrix(void){

int t;

for(t=0; t<3; t++) {

printing\_serial(" ");

printing\_char(matrix[t][0]);

printing\_serial(" | ");

printing\_char(matrix[t][1]);

printing\_serial(" | ");

printing\_char(matrix[t][2]);

if(t!=2){

printing\_serial("\r\n---|---|---\r\n");

}

}

printing\_serial("\r\n");

}

const unsigned char keypad[4][4] =

{

'1','2','3','A',

'4','5','6','B',

'7','8','9','C',

'\*','0','#','D'

};

void readKeypad(){

do{

PORTA = PORTA | 0x0F; //COLUMNS SET HIGH

row = PORTA & 0xF0; //READ ROWS

}while(row == 0x00); //WAIT UNTIL KEY PRESSED

do{

do{

MSDelay(1);

row = PORTA & 0xF0; //READ ROWS

}while(row == 0x00); //CHECK FOR KEY PRESS

MSDelay(15); //WAIT FOR DEBOUNCE

row = PORTA & 0xF0;

}while(row == 0x00); //FALSE KEY PRESS

while(1){

PORTA &= 0xF0; //CLEAR COLUMN

PORTA |= 0x01; //COLUMN 0 SET HIGH

row = PORTA & 0xF0; //READ ROWS

if(row != 0x00){ //KEY IS IN COLUMN 0

column = 0;

break;

}

PORTA &= 0xF0; //CLEAR COLUMN

PORTA |= 0x02; //COLUMN 1 SET HIGH

row = PORTA & 0xF0; //READ ROWS

if(row != 0x00){ //KEY IS IN COLUMN 1

column = 1;

break;

}

PORTA &= 0xF0; //CLEAR COLUMN

PORTA |= 0x04; //COLUMN 2 SET HIGH

row = PORTA & 0xF0; //READ ROWS

if(row != 0x00){ //KEY IS IN COLUMN 2

column = 2;

break;

}

PORTA &= 0xF0; //CLEAR COLUMN

PORTA |= 0x08; //COLUMN 3 SET HIGH

row = PORTA & 0xF0; //READ ROWS

if(row != 0x00){ //KEY IS IN COLUMN 3

column = 3;

break;

} //KEY NOT FOUND

}

if(row == 0x10){

myRow=0;

}

else if(row == 0x20){

myRow=1;

}

else if(row == 0x40){

myRow=2;

}

else if(row == 0x80){

myRow=3;

}

do{

MSDelay(15);

PORTA = PORTA | 0x0F; //COLUMNS SET HIGH

row = PORTA & 0xF0; //READ ROWS

}while(row != 0x00); //MAKE SURE BUTTON IS NOT STILL HELD

}

void get\_player\_move(void){

int x, y;

printing\_serial("Press keypad to select your move: \r\n");

readKeypad();

printing\_serial("\r\n");

x = myRow;

y = column;

if(matrix[x][y] != ' '){

printing\_serial("Invalid move, try again.\r\n");

get\_player\_move();

}else{

matrix[x][y] = 'X';

}

}

void get\_computer\_move(void){

int i, j;

for(i=0; i<3; i++){

for(j=0; j<3; j++)

if(matrix[i][j] == ' ') break;

if(matrix[i][j] == ' ') break;

}

if(i\*j==9) {

printing\_serial("Draw!\r\n");

}else

matrix[i][j]='O';

}

void printing\_serial(char \*p){

int i = 0;

char str[200];

strcpy(str, p);

while (str[i] != ''){

if (SCI0SR1 & 0x80){

SCI0DRL = str[i++];

}

}

return;

}

void printing\_char(char p){

int i = 0;

while (i < 1){

if (SCI0SR1 & 0x80){

SCI0DRL = p;

i++;

}

}

return;

}

char reading\_serial(){

while (!(SCI0SR1 & 0x20));

return SCI0DRL;

}

void init\_matrix(){

int i, j;

for(i=0; i< 3; i++) {

for(j=0; j< 3; j++){

matrix[i][j] = ' ';

}

printing\_serial("\r\n");

}

return;

}

char check(void){

int i;

for(i=0; i<3; i++)

if(matrix[i][0]==matrix[i][1] && matrix[i][0]==matrix[i][2] )

return matrix[i][0];

for(i=0; i<3; i++)

if(matrix[0][i]==matrix[1][i] && matrix[0][i]==matrix[2][i])

return matrix[0][i];

if(matrix[0][0]==matrix[1][1] && matrix[1][1]==matrix[2][2])

return matrix[0][0];

if(matrix[0][2]==matrix[1][1] && matrix[1][1]==matrix[2][0])

return matrix[0][2];

return ' ';

}

*Link-ul de GitHub unde se poate găsi codul și documentația:* <https://github.com/YettySpaghetty/SIproiect>

*5. Bibliografie*

* M.Popa, *Sisteme cu microcontrollere orientate pe aplicații* ; Orizonturi Universitare, Timișoara , 2003
* Dragon12-Plus2 Trainer For Freescale HCS12 microcontroller family , User’s Manual for Rev. A board Version 1.01
* MC9S12DP256B Device User Guide V02.15
* HCS12 Serial Communications Interface (SCI)-Block Guide V02.08