**Sisteme încorporate, An III, CTI**

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

Popa Nicolae Cosmin

Popescu Izabela Georgeana

Pospischil Ioana Loredana

1. Descrierea plăcii

Placa Dragon12-Plus conține:

- instalat un microcontroller MC9S12DP256CCPV

- sau MC9S12DG256CVPE.

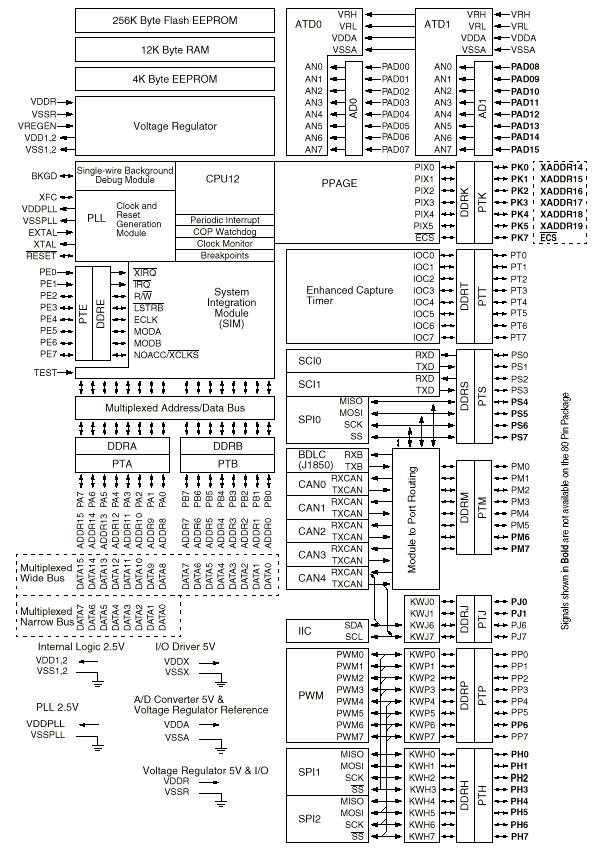
- MC9S12DP256 a fost eliminat de către Freescale fiind înlocuit cu MC9S12DG256.

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



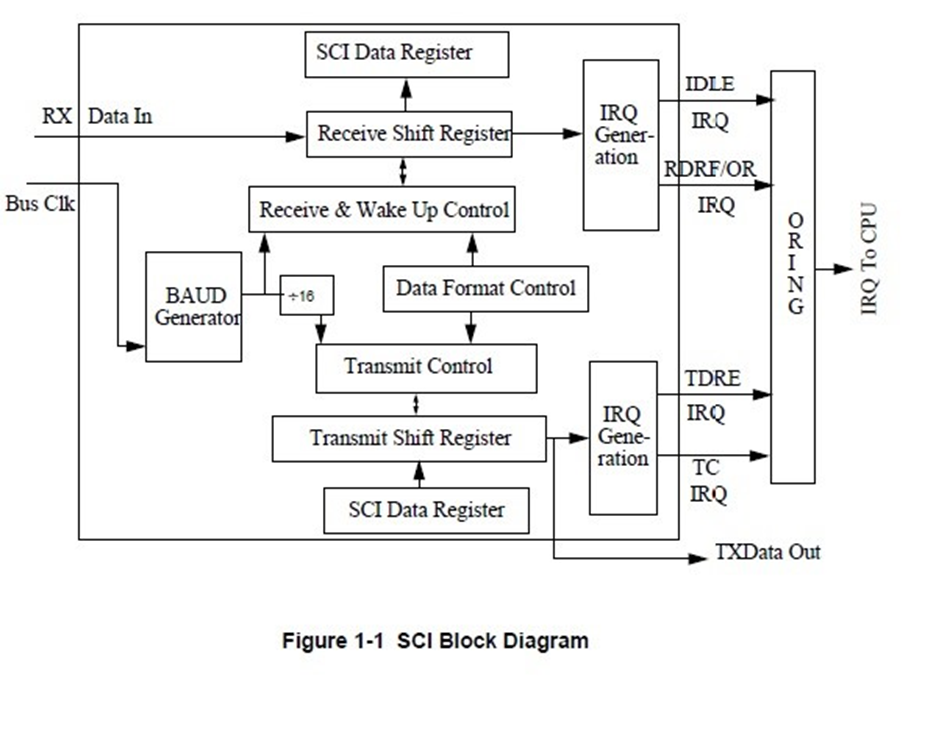
**Caracteristici hardware ale microcontrollerului MC9S12DG256**

* 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"

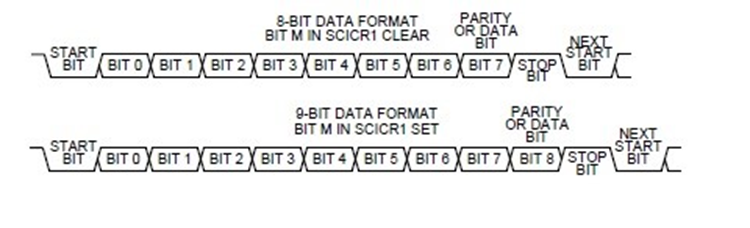
***Descrierea detaliată a modulelor microcontrolerului care au fost implicate în***

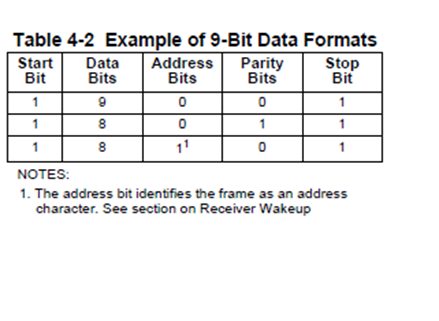
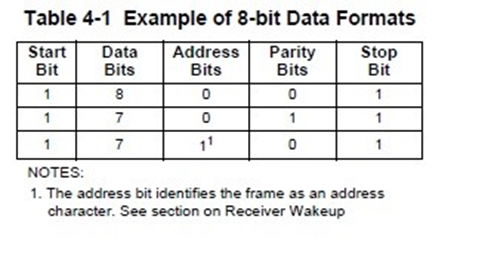
***realizarea proiectului***

**Interfaţa serială SCI**

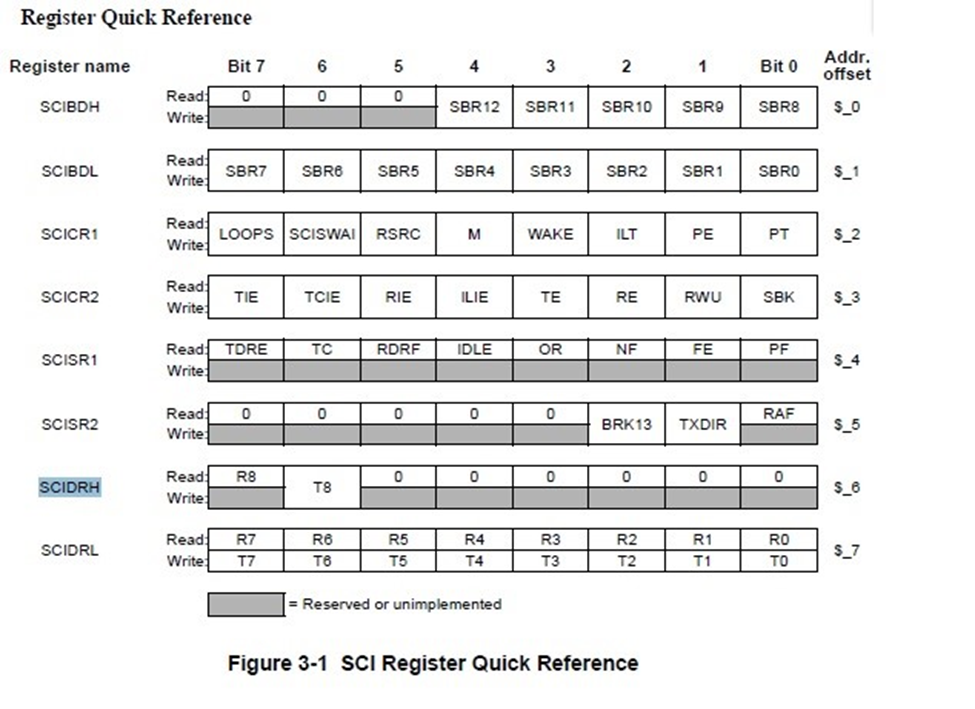


* Operare full duplex;
* Format pe 8 sau 9 biţi, plus biţii de cadrare şi bitul de paritate;
* Transmiţătorul şi receptorul pot fi validaţi separat;
* Generare de întrerupere la: transmitere încheiată, preluare caracter, eroare de ritm, de cadrare, de paritate, zgomote;
* Formatul datelor:





* 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.

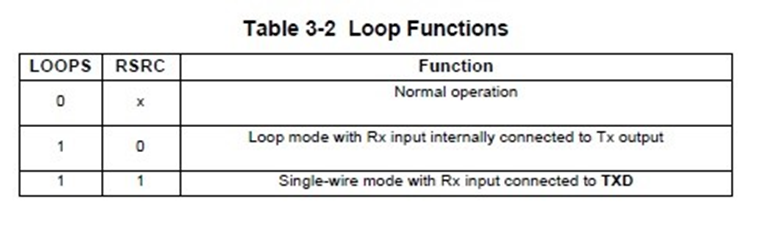
SCISWAI — SCI Stop in Wait Mode Bit

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: irrelevant, 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

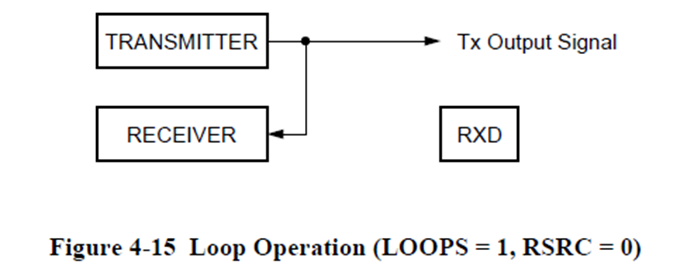
T8 este al nouălea bit de date transmis când SCI este configurat pentru un format de 9 biți(M=1).

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**:



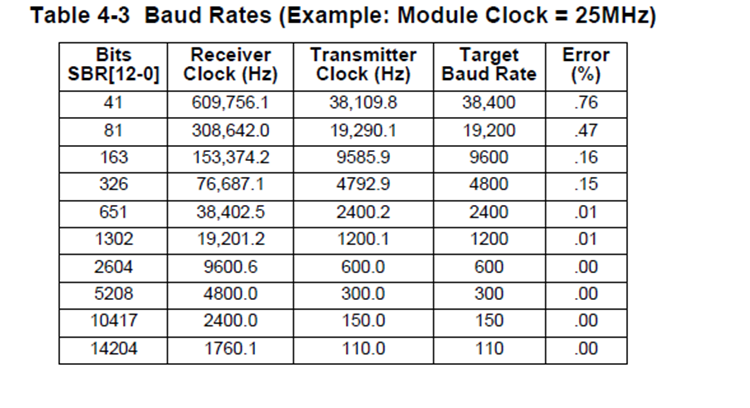
-în scop de test

-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])

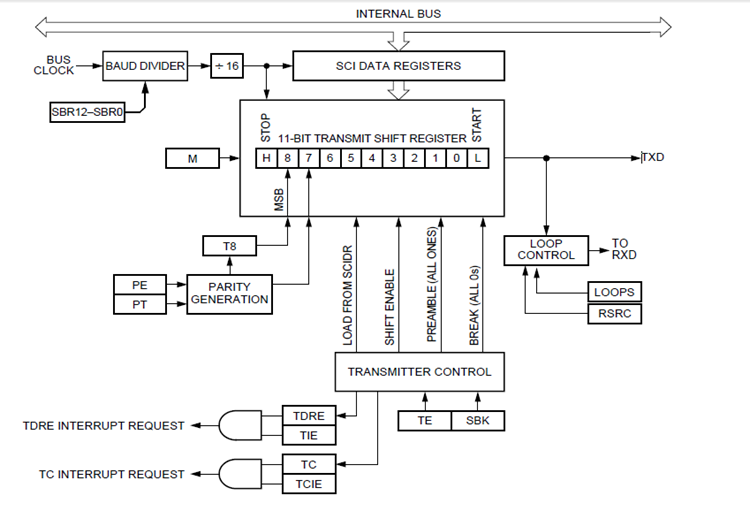
SCI module clock=4MHz

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

SCI0BDH=0x00;

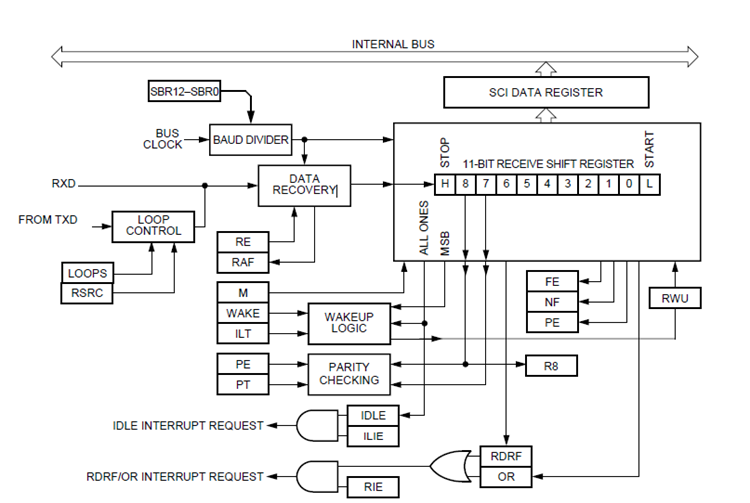
SCI0BDL=0x1A;

**Transmisia:**



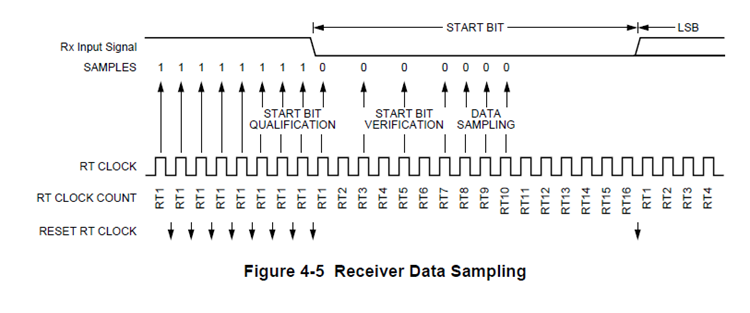
-M stabileşte formatul, cu 8 sau 9 biţi, PE validează paritatea, PT stabileşte tipul parităţii, TC arată încheierea transmisiei, TDRE cere date de la procesor, TIE şi TCIE validează cererile corespunzătoare de întrerupere, TE validează transmisia iar SBK permite generarea caracterului Break;

**Recepția:**

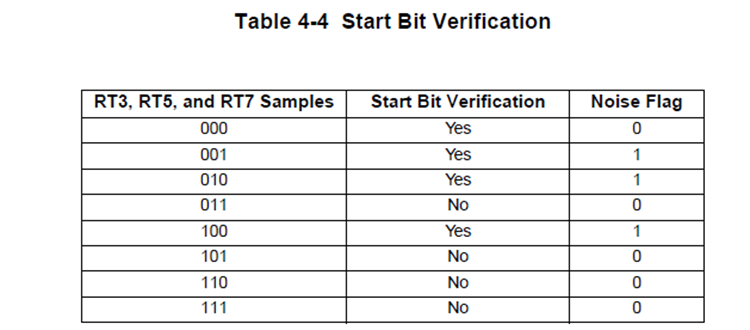


-RE validează recepţia, RAF arată recepţie în lucru, OR (overrun), FE (frame), NF (noise), PE (parrity) arată erori, RDRF arată caracter disponibil, RWU validează operaţia wakeup, WAKE stabileşte cine provoacă wakeup (adresa sau linia idle), ILIE, RIE validează cereri de întrerupere;

**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.

Codul se gasește la urmatorul link: <https://github.com/PopaCosmin/SI.git>

**COD**

**#include <time.h>**

**#include <stdio.h>**

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

**#include <stdlib.h>**

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

**#define COMPUTER 1**

**#define HUMAN 2**

**#define PLAYER1 3**

**#define PLAYER2 4**

**#define SIDE 3 // Lungimea tablei de joc**

**#define COMPUTERMOVE 'O' // Calculatorul va completa cu '0'**

**#define HUMANMOVE 'X' // Omul va completa cu 'X'**

**void MSDelay(unsigned int);**

**void newLine();**

**void sendString(char s[]);**

**void transmitereSeriala(unsigned char);**

**void sendStringWithoutNewLine(char s[]);**

**char scanKeypad();**

**void showBoard(char board[][SIDE]);**

**void showInstructions();**

**void initialise(char board[][SIDE]);**

**int rowCrossed(char board[][SIDE]);**

**int columnCrossed(char board[][SIDE]);**

**int diagonalCrossed(char board[][SIDE]);**

**int gameOver(char board[][SIDE]);**

**void playWithComputer(int whoseTurn);**

**void playWithFriend(int whoseTurn);**

**const unsigned char keypad[4][4] = // Initializarea tastaturii cu caractere**

**{**

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

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

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

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

**};**

**unsigned char column, row, choice;**

**unsigned int nrJocuri, winA = 0, winB = 0;**

**void main(void)**

**{**

**unsigned int i = 0;**

**//setam registri**

**SCI0BDH = 0x00;**

**SCI0BDL = 26;**

**//transmitter enabled**

**SCI0CR1 = 0x00;**

**//reciever enable**

**SCI0CR2 = 0x0C;**

**//The RUN mode (SW7=10) of Dragon12+ works at 8MHz.**

**//Serial Monitor used for LOAD works at 48MHz**

**//8MHz/2=4MHz, 4MHz/16=250,000 and 250,000/9600=26**

**DDRA = 0x0F; // MAKE ROWS INPUT AND COLUMNS OUTPUT**

**while(1) // Afisarea meniului**

**{**

**newLine();**

**sendString("How many games you want to play? ");**

**nrJocuri = scanKeypad(); // citim nr de jocuri**

**if(nrJocuri >= '1' && nrJocuri <= '9')**

**nrJocuri = nrJocuri - '0';//transformam in int**

**newLine();**

**sendString("A) PLAY WITH COMPUTER");**

**newLine();**

**sendString("B) PLAY WITH FRIEND");**

**newLine();**

**while(1)**

**{**

**sendString("Choose the opponent: ");**

**choice = scanKeypad(); // Pentru jocul cu calculatorul, se asteapta apasarea tastei 'A'**

**if(choice == 'A')**

**{**

**while(i < nrJocuri){**

**playWithComputer(HUMAN);**

**i++;**

**}**

**break;**

**}**

**else if(choice == 'B') // Pentru jocul cu calculatorul, se asteapta apasarea tastei 'B'**

**{**

**while(i < nrJocuri){**

**playWithFriend(PLAYER1);**

**i++;**

**}**

**break;**

**}**

**else**

**sendString("You must choose A or B!");**

**}**

**break;**

**}**

**}**

**// Functie care genereaza timpul de asteptare**

**void MSDelay(unsigned int itime)**

**{**

**unsigned int i, j;**

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

**for(j=0; j<4000; j++);**

**}**

**// Functie pentru transmiterea seriala a unui caracter**

**void transmitereSeriala(unsigned char c) // SCI0**

**{**

**while(!(SCI0SR1 & 0x80)); // Make sure the last bit is gone before giving it another byte**

**SCI0DRL = c;**

**}**

**// Use the pair (CR+LF) as a new-line character to send to Tera Term**

**// ASCII code 13 (decimal) is for CR, and 10 is for LF.**

**void newLine()**

**{**

**transmitereSeriala(0x0D);**

**transmitereSeriala(0x0A);**

**}**

**// Functie care afiseaza un string, cu new-line**

**void sendString(char s[])**

**{**

**int i =0;**

**while (s[i] != 0x00) // Se desparte string-ul in caractere**

**{**

**transmitereSeriala(s[i]); // Se transmite serial fiecare caracter din string**

**i++;**

**}**

**newLine();**

**}**

**// Functie care afiseaza un string, fara new-line**

**void sendStringWithoutNewLine(char s[])**

**{**

**int i =0;**

**while (s[i] != 0x00)**

**{**

**transmitereSeriala(s[i]);**

**i++;**

**}**

**}**

**char scanKeypad()**

**{**

**DDRA = 0x0F; // MAKE ROWS INPUT AND COLUMNS OUTPUT**

**while(1)**

**{**

**do**

**{**

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

**row = PORTA & 0xF0; // READ ROWS**

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

**do**

**{**

**do**

**{**

**MSDelay(1); // WAIT**

**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;**

**}**

**row = 0; // KEY NOT FOUND**

**break;**

**}**

**if(row == 0x10) // KEY IS IN ROW 0**

**{**

**return keypad[0][column];**

**}**

**else if(row == 0x20) // KEY IS IN ROW 1**

**{**

**return keypad[1][column];**

**}**

**else if(row == 0x40) // KEY IS IN ROW 2**

**{**

**return keypad[2][column];**

**}**

**else if(row == 0x80) // KEY IS IN ROW 3**

**{**

**return keypad[3][column];**

**}**

**do**

**{**

**MSDelay(15);**

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

**row = PORTA & 0xF0; // READ ROWS**

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

**}**

**}**

**// Functie care afiseaza tabla de joc**

**void showBoard(char board[][SIDE])**

**{**

**newLine();**

**newLine();**

**sendStringWithoutNewLine(" ");**

**transmitereSeriala(board[0][0]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[0][1]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[0][2]);**

**sendString(" ");**

**sendString("-------------");**

**sendStringWithoutNewLine(" ");**

**transmitereSeriala(board[1][0]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[1][1]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[1][2]);**

**sendString(" ");**

**sendString("-------------");**

**sendStringWithoutNewLine(" ");**

**transmitereSeriala(board[2][0]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[2][1]);**

**sendStringWithoutNewLine(" | ");**

**transmitereSeriala(board[2][2]);**

**sendString(" ");**

**newLine();**

**return;**

**}**

**// Functie care afiseaza instructiunile jocului**

**void showInstructions()**

**{**

**sendString("TIC-TAC-TOE");**

**newLine();**

**sendString("Choose a cell numbered from 1 to 9 as below:");**

**newLine();**

**sendString(" 1 | 2 | 3 ");**

**sendString("-------------");**

**sendString(" 4 | 5 | 6 ");**

**sendString("-------------");**

**sendString(" 7 | 8 | 9 ");**

**newLine();**

**return;**

**}**

**// Functie care initializeaza jocul**

**void initialise(char board[][SIDE])**

**{**

**int j,i;**

**srand(1);**

**// Initial, tabla de joc este goala**

**for (i=0; i<SIDE; i++)**

**{**

**for (j=0; j<SIDE; j++)**

**board[i][j] = ' ';**

**}**

**return;**

**}**

**// Functie care verifica daca un jucator a completat o linie**

**int rowCrossed(char board[][SIDE])**

**{**

**int i;**

**for (i=0; i<SIDE; i++)**

**{**

**if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != ' ')**

**return 1;**

**}**

**return 0;**

**}**

**// Functie care verifica daca un jucator a completat o coloana**

**int columnCrossed(char board[][SIDE])**

**{**

**int i;**

**for (i=0; i<SIDE; i++)**

**{**

**if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != ' ')**

**return 1;**

**}**

**return 0;**

**}**

**// Functie care verifica daca un jucator a completat o diagonala**

**int diagonalCrossed(char board[][SIDE])**

**{**

**if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != ' ')**

**return 1;**

**if (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != ' ')**

**return 1;**

**return 0;**

**}**

**// Functie care verifica daca jocul a fost castigat de unul din jucatori**

**int gameOver(char board[][SIDE])**

**{**

**// Jocul este castigat daca o linie, o coloana sau o diagonala este completata de acelasi jucator**

**return(rowCrossed(board) || columnCrossed(board) || diagonalCrossed(board) );**

**}**

**// Functie pentru jocul om-calculator**

**void playWithComputer(int whoseTurn)**

**{**

**int moveIndex = 0, option, x, y, flag;**

**char c;**

**// Tabla de joc este o matrice de caractere**

**char board[SIDE][SIDE];**

**// Initializarea jocului**

**initialise(board);**

**// Afisarea instructiunilor**

**showInstructions();**

**// Jocul continua pana se ajunge la castigul unuia dintre jucatori sau egal (toata tabla este completata)**

**while (gameOver(board) == 0 && moveIndex != SIDE\*SIDE)**

**{**

**if (whoseTurn == COMPUTER) // Randul calculatorului**

**{**

**while(1) // Se incearca gasirea unei mutari valide**

**{**

**option = rand() % 9 + 1; // Se genereaza o mutare random**

**switch(option) // Se cauta locul din matrice corespunzator mutarii**

**{**

**case 1: {**

**x = 0;**

**y = 0;**

**break;**

**}**

**case 2: {**

**x = 0;**

**y = 1;**

**break;**

**}**

**case 3: {**

**x = 0;**

**y = 2;**

**break;**

**}**

**case 4: {**

**x = 1;**

**y = 0;**

**break;**

**}**

**case 5: {**

**x = 1;**

**y = 1;**

**break;**

**}**

**case 6: {**

**x = 1;**

**y = 2;**

**break;**

**}**

**case 7: {**

**x = 2;**

**y = 0;**

**break;**

**}**

**case 8: {**

**x = 2;**

**y = 1;**

**break;**

**}**

**case 9: {**

**x = 2;**

**y = 2;**

**break;**

**}**

**}**

**if(board[x][y] == ' ') // Daca locul este liber**

**{**

**board[x][y] = COMPUTERMOVE; // Se completeaza cu 0, altfel se genereaza mutari pana la gasirea unui loc liber**

**break;**

**}**

**}**

**// Se afiseaza mutarea**

**sendStringWithoutNewLine("COMPUTER has put ");**

**transmitereSeriala(COMPUTERMOVE);**

**sendStringWithoutNewLine(" in cell ");**

**c = option + '0';**

**transmitereSeriala(c);**

**showBoard(board); // Se afiseaza tabla de joc actualizata**

**moveIndex ++; // Creste numarul mutarilor**

**whoseTurn = HUMAN; // Omul face urmatoarea mutare**

**}**

**else if (whoseTurn == HUMAN) // Randul omului**

**{**

**while(1)**

**{**

**flag = 1;**

**sendString("Choose the cell: ");**

**choice = scanKeypad(); // Se citeste mutarea de la tastatura sub forma de caracter**

**if((choice>='1') && (choice<='9')) // Se verifica daca reprezinta o cifra de la 1 la 9**

**{**

**option = choice - '0'; // Conversie la intreg**

**}**

**else**

**{**

**sendString("Invalid option, please try again!");**

**flag = 0; // In cazul unei mutari invalide, se reia citirea pana la alegerea unei mutari valide**

**}**

**switch(option) // Se cauta locul din matrice corespunzator mutarii**

**{**

**case 1: {**

**x = 0;**

**y = 0;**

**break;**

**}**

**case 2: {**

**x = 0;**

**y = 1;**

**break;**

**}**

**case 3: {**

**x = 0;**

**y = 2;**

**break;**

**}**

**case 4: {**

**x = 1;**

**y = 0;**

**break;**

**}**

**case 5: {**

**x = 1;**

**y = 1;**

**break;**

**}**

**case 6: {**

**x = 1;**

**y = 2;**

**break;**

**}**

**case 7: {**

**x = 2;**

**y = 0;**

**break;**

**}**

**case 8: {**

**x = 2;**

**y = 1;**

**break;**

**}**

**case 9: {**

**x = 2;**

**y = 2;**

**break;**

**}**

**}**

**if(flag == 1)**

**{**

**if(board[x][y] == ' ') // Daca locul este liber**

**{**

**board[x][y] = HUMANMOVE; // Se completeaxa cu 'X'**

**break;**

**}**

**else // Daca locul corespunzator mutarii este ocupat, se reia citirea pana la alegerea unei mutari valide**

**{**

**sendString("Cell already occupied, choose another!");**

**newLine();**

**}**

**}**

**}**

**// Se afiseaza mutarea**

**sendStringWithoutNewLine("HUMAN has put ");**

**transmitereSeriala(HUMANMOVE);**

**sendStringWithoutNewLine(" in cell ");**

**c = option + '0';**

**transmitereSeriala(c);**

**showBoard(board); // Se afiseaza tabla de joc actualizata**

**moveIndex ++; // Creste numarul mutarilor**

**whoseTurn = COMPUTER; // Calculatorul face urmatoarea mutare**

**}**

**}**

**// Daca s-a ajuns la egal, se afiseaza mesajul corespunzator**

**if (gameOver(board) == 0 && moveIndex == SIDE \* SIDE)**

**{**

**sendString("It's a draw!");**

**newLine();**

**}**

**else // Daca jocul a fost castigat, se afiseaza castigatorul**

**{**

**if (whoseTurn == COMPUTER){**

**winA++;**

**if(winA == nrJocuri){**

**sendString("HUMAN has won the matchs!");**

**//break;**

**}**

**else**

**sendString("HUMAN has won the game!");**

**}**

**else if (whoseTurn == HUMAN) {**

**winB++;**

**if(winB == nrJocuri){**

**sendString("COMPUTER has won the matchs!");**

**//break;**

**}**

**else**

**sendString("COMPUTER has won the game!");**

**}**

**}**

**return;**

**}**

**// Functie pentru joc intre 2 persoane**

**void playWithFriend(int whoseTurn)**

**{**

**int moveIndex = 0, option, x, y, flag;**

**char c;**

**// Tabla de joc este o matrice de caractere**

**char board[SIDE][SIDE];**

**// Initializarea jocului**

**initialise(board);**

**// Afisarea instructiunilor**

**showInstructions();**

**// Jocul continua pana se ajunge la castigul unuia dintre jucatori sau egal (toata tabla este completata)**

**while (gameOver(board) == 0 && moveIndex != SIDE\*SIDE)**

**{**

**if (whoseTurn == PLAYER1) // Randul jucatorului 1**

**{**

**sendString("PLAYER 1");**

**while(1)**

**{**

**flag = 1;**

**sendString("Choose the cell: ");**

**choice = scanKeypad(); // Se citeste mutarea de la tastatura sub forma de caracter**

**if((choice>='1') && (choice<='9')) // Se verifica daca reprezinta o cifra de la 1 la 9**

**{**

**option = choice - '0'; // Conversie la intreg**

**}**

**else**

**{**

**sendString("Invalid option, please try again!");**

**flag = 0; // In cazul unei mutari invalide, se reia citirea pana la alegerea unei mutari valide**

**}**

**switch(option) // Se cauta locul din matrice corespunzator mutarii**

**{**

**case 1: {**

**x = 0;**

**y = 0;**

**break;**

**}**

**case 2: {**

**x = 0;**

**y = 1;**

**break;**

**}**

**case 3: {**

**x = 0;**

**y = 2;**

**break;**

**}**

**case 4: {**

**x = 1;**

**y = 0;**

**break;**

**}**

**case 5: {**

**x = 1;**

**y = 1;**

**break;**

**}**

**case 6: {**

**x = 1;**

**y = 2;**

**break;**

**}**

**case 7: {**

**x = 2;**

**y = 0;**

**break;**

**}**

**case 8: {**

**x = 2;**

**y = 1;**

**break;**

**}**

**case 9: {**

**x = 2;**

**y = 2;**

**break;**

**}**

**}**

**if(flag == 1)**

**{**

**if(board[x][y] == ' ') // Daca locul este liber**

**{**

**board[x][y] = HUMANMOVE; // Se completeaxa cu 'X'**

**break;**

**}**

**else // Daca locul corespunzator mutarii este ocupat, se reia citirea pana la alegerea unei mutari valide**

**{**

**sendString("Cell already occupied, choose another!");**

**newLine();**

**}**

**}**

**}**

**// Se afiseaza mutarea**

**sendStringWithoutNewLine("PLAYER 1 has put ");**

**transmitereSeriala(HUMANMOVE);**

**sendStringWithoutNewLine(" in cell ");**

**c = option + '0';**

**transmitereSeriala(c);**

**showBoard(board); // Se afiseaza tabla de joc actualizata**

**moveIndex ++; // Creste numarul mutarilor**

**whoseTurn = PLAYER2; // Celalalt jucator face urmatoarea mutare**

**}**

**else if (whoseTurn == PLAYER2) // Randul jucatorului 1**

**{**

**sendString("PLAYER 2");**

**while(1)**

**{**

**flag = 1;**

**sendString("Choose the cell: ");**

**choice = scanKeypad(); // Se citeste mutarea de la tastatura sub forma de caracter**

**if((choice>='1') && (choice<='9')) // Se verifica daca reprezinta o cifra de la 1 la 9**

**{**

**option = choice - '0'; // Conversie la intreg**

**}**

**else**

**{**

**sendString("Invalid option, please try again!");**

**flag = 0; // In cazul unei mutari invalide, se reia citirea pana la alegerea unei mutari valide**

**}**

**switch(option) // Se cauta locul din matrice corespunzator mutarii**

**{**

**case 1: {**

**x = 0;**

**y = 0;**

**break;**

**}**

**case 2: {**

**x = 0;**

**y = 1;**

**break;**

**}**

**case 3: {**

**x = 0;**

**y = 2;**

**break;**

**}**

**case 4: {**

**x = 1;**

**y = 0;**

**break;**

**}**

**case 5: {**

**x = 1;**

**y = 1;**

**break;**

**}**

**case 6: {**

**x = 1;**

**y = 2;**

**break;**

**}**

**case 7: {**

**x = 2;**

**y = 0;**

**break;**

**}**

**case 8: {**

**x = 2;**

**y = 1;**

**break;**

**}**

**case 9: {**

**x = 2;**

**y = 2;**

**break;**

**}**

**}**

**if(flag == 1)**

**{**

**if(board[x][y] == ' ') // Daca locul este liber**

**{**

**board[x][y] = COMPUTERMOVE; // Se completeaxa cu '0'**

**break;**

**}**

**else // Daca locul corespunzator mutarii este ocupat, se reia citirea pana la alegerea unei mutari valide**

**{**

**sendString("Cell already occupied, choose another!");**

**newLine();**

**}**

**}**

**}**

**// Se afiseaza mutarea**

**sendStringWithoutNewLine("PLAYER 2 has put ");**

**transmitereSeriala(COMPUTERMOVE);**

**sendStringWithoutNewLine(" in cell ");**

**c = option + '0';**

**transmitereSeriala(c);**

**showBoard(board); // Se afiseaza tabla de joc actualizata**

**moveIndex ++; // Creste numarul mutarilor**

**whoseTurn = PLAYER1; // Celalalt jucator face urmatoarea mutare**

**}**

**}**

**// Daca s-a ajuns la egal, se afiseaza mesajul corespunzator**

**if (gameOver(board) == 0 && moveIndex == SIDE \* SIDE)**

**{**

**sendString("It's a draw!");**

**newLine();**

**}**

**else // Daca jocul a fost castigat, se afiseaza castigatorul**

**{**

**if (whoseTurn == PLAYER1){**

**winB++;**

**if(winB == nrJocuri){**

**sendString("PLAYER 2 has won the matchs!");**

**// break;**

**}**

**else**

**sendString("PLAYER 2 has won the game!");**

**}**

**else if (whoseTurn == PLAYER2){**

**winA++;**

**if(winA == nrJocuri){**

**sendString("PLAYER 1 has won the matchs!");**

**// break;**

**}**

**else**

**sendString("PLAYER 1 has won the game!");**

**}**

**}**

**return;**

**}**