ISEL
ADEETC
Área Departamental de Engenharia Electrónica e de Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo da Roleta

SOC

(Roulette Game)

A46378 Berto Barata
A44787 Gonçalo Garcia
A44776 João Gomes

Professores:

Pedro Miguens Matutino (pedro.miguens@isel.pt)

Nuno Sebastião (nuno.sebastiao@isel.pt)

Projeto de
Laboratório de Informática e Computadores
2020 / 2021 inverno

22 de outubro de 2020

Índice

Introdução

O projeto semestral desta unidade curricular como já apresentado anteriormente consiste no desenvolvimento de um jogo da roleta. O seu modo de jogo também explicado anteriormente. O sistema que implementa o jogo será constituído por um computador (módulo de controlo), um teclado de doze teclas, um moedeiro, um mostrador LCD, um mostrador da roleta e uma chave de manutenção. Nesta apresentação iremos explicar o funcionamento e desenvolvimento do módulo SOC. Este SOC é constituído por submódulos, o Serial Receiver e Dispatcher. O bloco Serial Receiver do módulo SOC é constituído por quatro blocos principais: o Serial Control que é o bloco de controlo; o shift register que é o bloco conversor série paralelo; o counter, um contador de bits recebidos; e o parity check, um bloco de validação de paridade. O bloco Dispatcher que é responsável pela entrega ao LCD e ao Roulette Display das tramas válidas recebidas pelo bloco Serial Receiver.

SOC

O módulo Serial Output Controller (SOC) executa a interface com o LCD e com o mostrador da roleta, fazendo a receção da informação enviada pelo módulo de controlo e entregandoa posteriormente ao destinatário, conforme representado no diagrama de blocos do módulo Serial Output Controller.
O módulo SOC recebe, em série, uma mensagem constituída por 6 bits de informação e um bit de paridade, utilizado para detetar erros durante a transmissão. A comunicação com este módulo realiza-se segundo o protocolo de comunicação com o módulo Serial Output Controller, em que o bit LnD identifica o destinatário da mensagem e o último bit (P) contém a informação de paridade ímpar. Nas mensagens para o LCD, o bit RS é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes 4 bits contêm os dados a entregar ao LCD. Nas mensagens para o Roulette Display, os 5 bits constituem os dados a entregar ao dispositivo.
O emissor, realizado em software, quando pretende enviar uma trama para o módulo SOC promove uma condição de início de trama (Start), que corresponde a uma transição ascendente na linha SOCsel. Após a condição de início, o módulo SOC é responsável por armazenar os bits de dados da trama nas transições ascendentes do sinal SCLK.

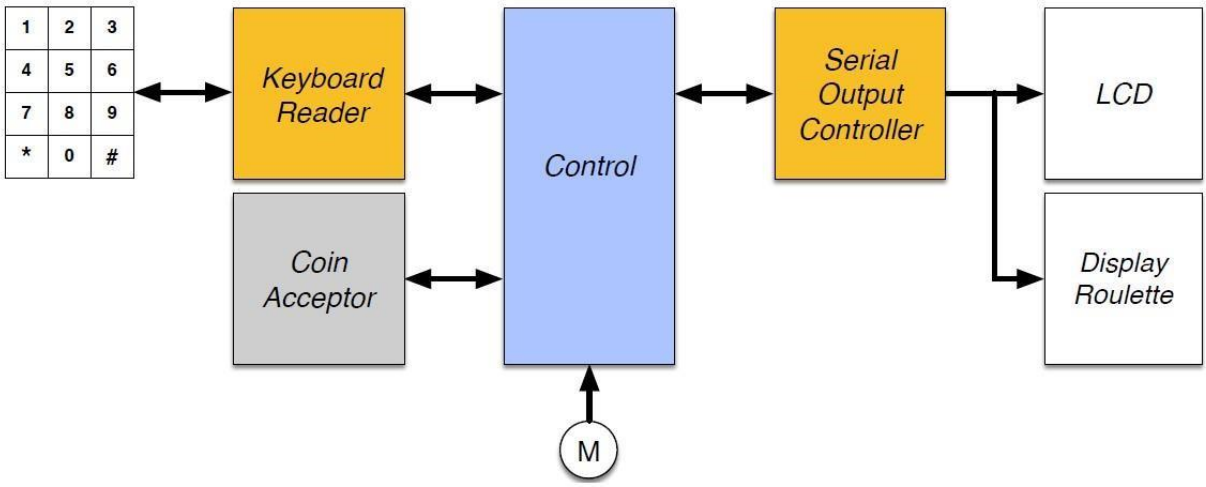


Figura 1 – Diagrama de blocos do módulo de implementação de Roulette Game.

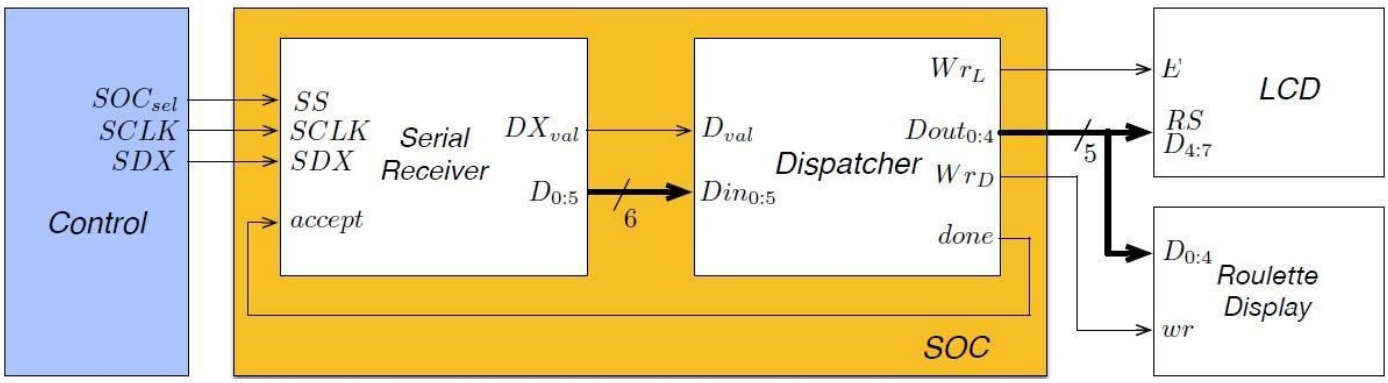


Figura 2 - Diagrama de blocos do módulo Serial Output Controller

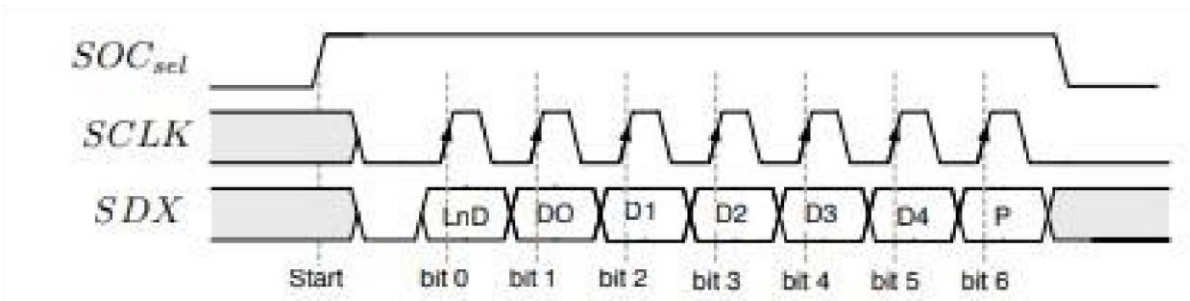


Figura 3 – Protocolo de comunicação com o módulo Serial Output Controller

Interface com o *Control*

Para Implementar o modulo *SOC* em *software* criou-se a classe *SerialEmitter*, recorrendo a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 4.

2.1 Serial Receiver

Como já foi dito anteriormente este módulo está divido em quatro submódulo, Serial Control, Shift Register, Counter e Parity Check. O Serial Control tem como função a verificação dos bits enviados, se foram recebidos de forma correta, e só assim avançar para uma nova trama. O Shift Register tem como principal função o envio dos bit recebidos do Control, e deslocá-los (Shift) para o Dispatcher. O counter é utilizado para regular os bits, ou seja, verificar se foram todos transferidos e indicar ao Serial Control. Por último o Parity Check é utilizado para a deteção de bits errados, verificando o bit de paridade tendo em conta os bits transferidos.

2.2 Dispatcher

O Dispatcher tem como função a receção de uma nova trama válida enviada pelo módulo Serial Receiver, será sinalizada pelo sinal Dval e a sua entrega ao periférico alvo é feita através os sinais WrL e WrD, para o LCD e para o Roulette Display respetivamente. Enviando a trama corretamente, ativa o sinal 'done' que por sua vez, irá indicar ao módulo Serial Receiver que poderá enviar uma nova trama para ser processada.
Podes colocar isto na parta em que está a falar do SOC "O processamento das tramas recebidas pelo SOC, para o LCD ou para o Roulette Display, deverá respeitar a especificação definida pelo fabricante de cada periférico e possibilitar a libertação do canal de receção série o mais rapidamente possível."

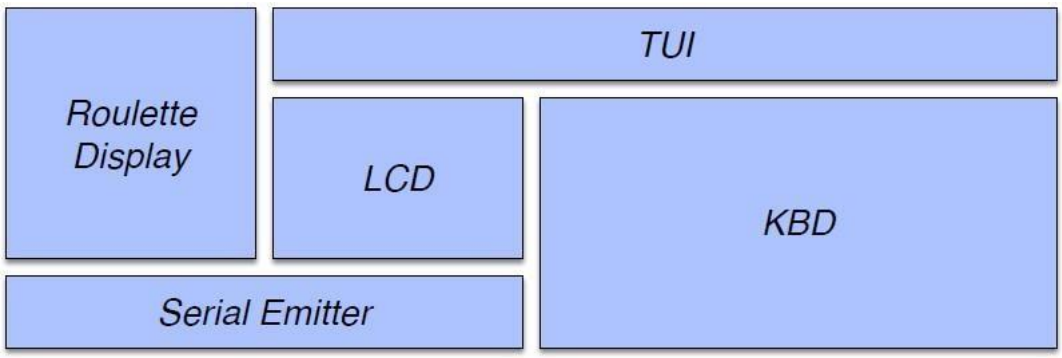


Figura 4 – Diagrama lógico do módulo *Control* de interface com o módulo *Serial Emitter*

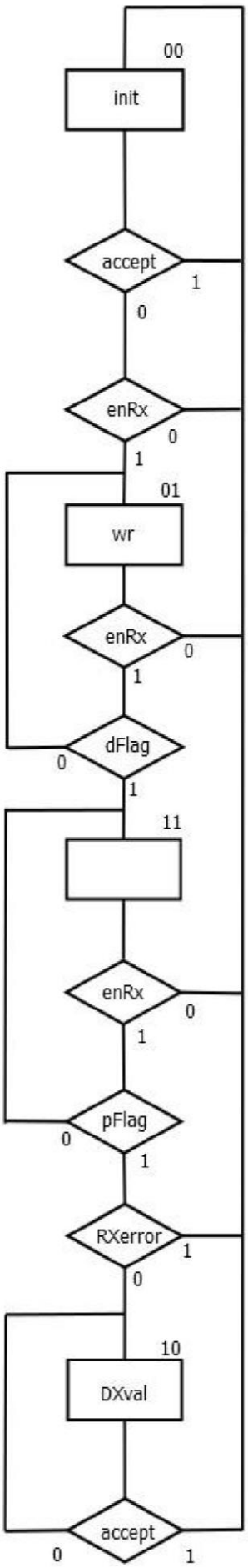


Figura 5 Máquina de estados do bloco *Serial Reciever*

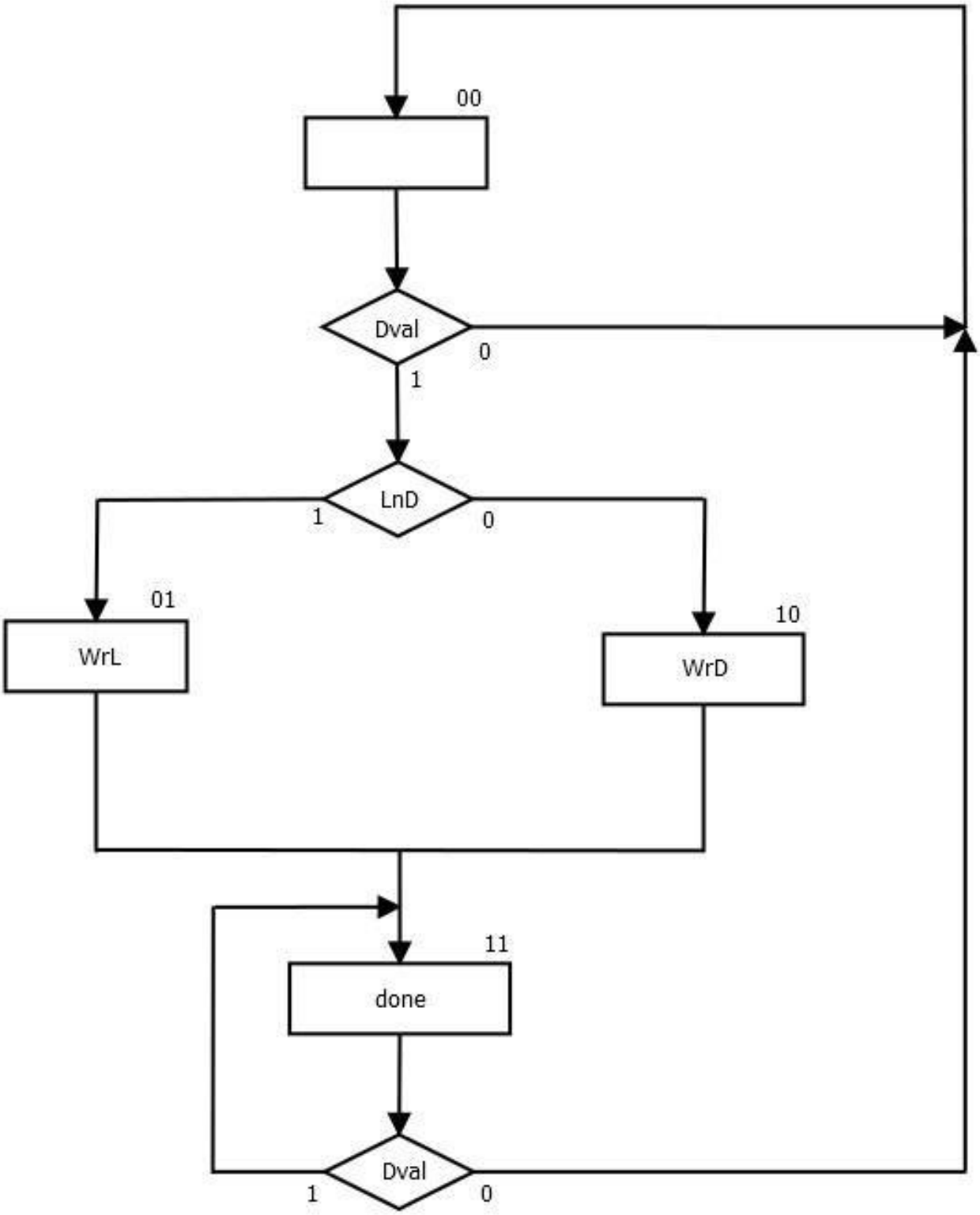


Figura 6 - Máquina de estados do bloco *Dispatcher*

Conclusões

Os objetivos solicitados foram alcançados. Percorrendo todos os passos necessário e imprescindíveis para a conclusão com sucesso deste módulo. Melhorámos as nossas capacidades de programação em Java, no desenvolvimento da descrição de hardware, bem como na elaboração de esquemas elétricos. Concluindo esta etapa a transmissão dos bits passou a ser feita em série, devido a realização da classe Serial Emitter e do submódulo Serial Receiver, ao contrário do que acontecia nas duas etapas anteriores onde a transmissão era feita em paralelo. Algumas etapas foram mais trabalhosas que outras, nomeadamente a resolução do submódulo Parity Check e o desenvolvimento da classe Serial Emitter que passou por várias versões e processos de simplificação.

A.1. Descrição CUPL do módulo SOC

```
Name        SOC;
PartNo      ;
Date        13/01/21;
Revision    ;
Designer    GG;
Company     ;
Assembly    ;
Location    ;
Device      V750C;

/* Input Pins */

PIN 1 = SCLK;
PIN 2 = SS;
PIN 3 = SDX;
PIN 4 = MCLK;

/* Output Pins */

PIN 23 = WrL;
PIN [22..18] = [D4..0]; /* [Dout0..4] */
PIN 17 = WrD;

/* Pin Nodes */

PINNODE [34..32] = [CQ0..2]; /* Flip-flops for the counter. */
PINNODE 31 = D5; /* Flip-flop for the shift register. */
PINNODE 30 = err; /* Flip-flop for the parity checker. */
PINNODE [29..28] = [SCQ0..1]; /* Flip-flops for the Serial Control ASM. */
PINNODE [27..26] = [DQ0..1]; /* Flip-flops for the Dispatcher ASM. */

/* Up Counter (3-Bit, Synchronous, Counts From 0 To 7) */

clr = init;

[CQ0..2].SP = 'b'0; /* Not used. */
[CQ0..2].AR = clr;
[CQ0..2].CKMUX = SCLK;

Six = CQ2 & CQ1 & !CQ0;
Seven = CQ2 & CQ1 & CQ0;

CQ0.D = !CQ0;
CQ1.D = CQ1 $ CQ0;
CQ2.D = CQ2 $ CQ1 & CQ0;
```

```
/* Shift Register (6-Bit) */

data = SDX;
enableShift = wr;

[D0..5].SP = 'b'0; /* Not used. */
[D0..5].AR = 'b'0; /* Not used. */
[D0..5].CKMUX = SCLK;

[D0..5].D = [data,D0..4] & enableShift # [D0..5] & !enableShift;

/* Parity Check */

err.SP = 'b'0; /* Not used. */ err.AR = init;
err.CKMUX = SCLK;

err.D = err $ data; /* Parity. */

/* Serial Control ASM */

[SCQ0..1].SP = 'b'0; /* Not used. */
[SCQ0..1].AR = 'b'0; /* Not used. */
[SCQ0..1].CK = MCLK;

enRX = SS; accept = done; pFlag = Seven; dFlag = Six; RXerror = !err;
/* Gray encoding. */
$DEFINE INIT                                    'b'00
$DEFINE WRITE                                   'b'01
$DEFINE WAIT_ENRX_AND_PFLAG_AND_NOT_RXERROR     'b'11
$DEFINE DATA_VALID                              'b'10

SEQUENCE [SCQ1..0] {
    PRESENT INIT
        OUT init;
        IF !accept & enRX NEXT WRITE;
        DEFAULT NEXT INIT;
    PRESENT WRITE
        OUT wr;
        IF !enRX NEXT INIT;
        IF enRX & dFlag NEXT WAIT_ENRX_AND_PFLAG_AND_NOT_RXERROR;
        DEFAULT NEXT WRITE;
    PRESENT WAIT_ENRX_AND_PFLAG_AND_NOT_RXERROR
        IF !enRX NEXT INIT;
        IF enRX & pFlag & RXerror NEXT INIT;
        IF enRX & pFlag & !RXerror NEXT DATA_VALID;
        DEFAULT NEXT WAIT_ENRX_AND_PFLAG_AND_NOT_RXERROR;
    PRESENT DATA_VALID
        OUT DXval;
        IF accept NEXT INIT;
        DEFAULT NEXT DATA_VALID;
}
```

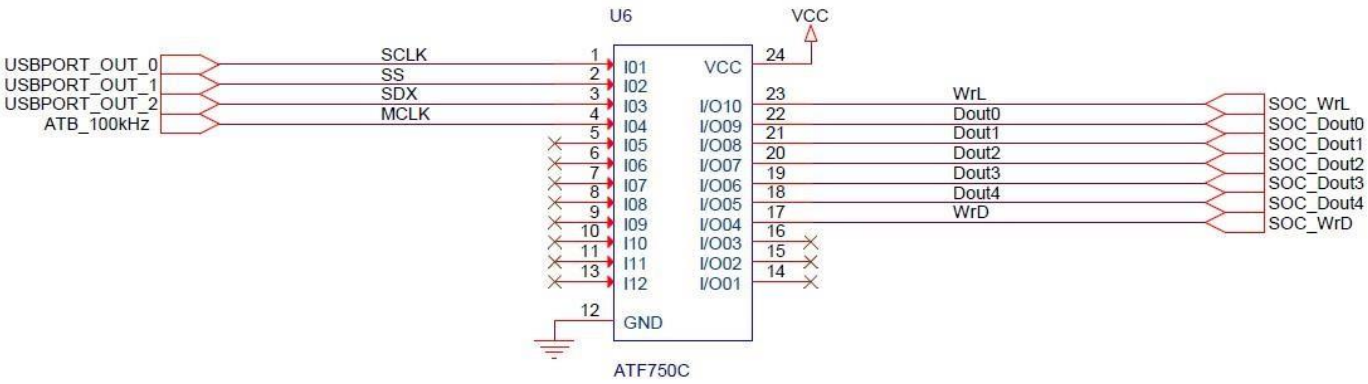
```
/* Dispatcher ASM */

[DQ0..1].SP = 'b'0; /* Not used. */
[DQ0..1].AR = 'b'0; /* Not used. */
[DQ0..1].CK = MCLK;

Dval = DXval;
LnD = D5;
/* Gray encoding. */
$DEFINE WAIT_DVAL               'b'00
$DEFINE WRITE_LCD               'b'01
$DEFINE WRITE_ROULETTE_DISPLAY  'b'10
$DEFINE DONE                    'b'11

SEQUENCE [DQ1..0] {
    PRESENT WAIT_DVAL
        IF Dval & LnD NEXT WRITE_LCD;
        IF Dval & !LnD NEXT WRITE_ROULETTE_DISPLAY;
        DEFAULT NEXT WAIT_DVAL;
    PRESENT WRITE_LCD
        OUT WrL;
        NEXT DONE;
    PRESENT WRITE_ROULETTE_DISPLAY
        OUT WrD;
        NEXT DONE;
    PRESENT DONE
        OUT done;
        IF !Dval NEXT WAIT_DVAL;
        DEFAULT NEXT DONE;
}
```

A.2. Esquema elétrico do módulo *SOC*



A.3. Código Java da classe *SerialEmitter*

```
public class SerialEmitter {
public enum Destination {RDisplay, LCD}
private static final int SS_MASK_OUT_PORT = 0x01;//SOC Select.
private static final int SCLK_MASK_OUT_PORT = 0x02;//SOC Clock.
private static final int SDX_MASK_OUT_PORT = 0x04; //Send.
private static final int DATA_SIZE_BITS = 5;

public static void main(String[] args) {
HAL.init();          init();      }
    public static void init() {
HAL.clrBits(SS_MASK_OUT_PORT);

HAL.clrBits(SCLK_MASK_OUT_PORT);
    }
    public static void send(Destination addr, int data)
    {          HAL.setBits(SS_MASK_OUT_PORT);
    int bit = Destination.LCD == addr ? 1 : 0;
    int parity = bit ^ 1; // 1 for odd parity, 0 for
    even.
    sendBit(bit);

    for (int i = 0; i < DATA_SIZE_BITS;
    i++){              bit = data & 1;
    data >>= 1;
    parity ^= bit;
    sendBit(bit);
       }
   sendBit(parity);
   HAL.clrBits(SS_MASK_OUT_PORT);
   }

private static void sendBit(int bit) {
if (1 == bit) {
      HAL.setBits(SDX_MASK_OUT_PORT);
}
else{
      HAL.clrBits(SDX_MASK_OUT_PORT);
}
      HAL.setBits(SCLK_MASK_OUT_PORT);
      HAL.clrBits(SCLK_MASK_OUT_PORT);
    }
}
```