O módulo Serial Score Controller (SSC) implementa a interface com o mostrador de pontuação (Score Display), realizando a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao mostrador de pontuação

1 Módulo SSC

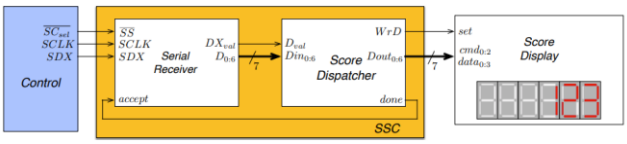

Figura 1 – Diagrama de blocos do módulo Serial Score Controller e do Score Display

O módulo SSC recebe em série uma mensagem composta por sete (7) bits de informação e um (1) bit de paridade par, segundo o protocolo de comunicação. Os três primeiros bits de informação, indicam o comando a realizar no mostrador de pontuação. Os restantes quatro bits identificam o campo de dados. Tal como acontece com o SLCDC, o canal de receção série pode ser libertado após a receção da trama recebida pelo Score Display, não sendo necessário esperar pela sua execução do comando correspondente. Assim, o bloco Score Dispatcher pode ativar, prontamente, o sinal done para informar o bloco Serial Receiver que a trama já foi processada.

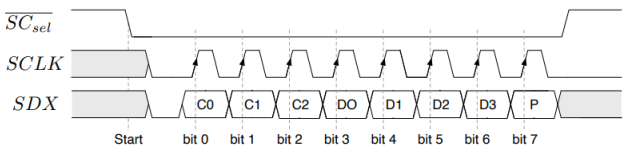

Figura 2 – Protocolo de comunicação do módulo Serial Score Controller

Cmd 2 1 0	*data* 3 2 1 0	*Function*
0 0 0	$d_3\ d_2\ d_1\ d_0$	*update digit 0*
0 0 1	$d_3\ d_2\ d_1\ d_0$	*update digit 1*
...	...	...
1 0 1	$d_3\ d_2\ d_1\ d_0$	*update digit 5*
1 1 0	$-\ -\ -\ -$	*update display*
1 1 1	$-\ -\ -\ 0$	*display on*
1 1 1	$-\ -\ -\ 1$	*display off*

Figura 3 – Comandos do módulo Score Display

2 Score Display

O Score Display aceita três linhas de comando de entrada (cmd 2.. 0) e quatro linhas de dados de entrada (data 3.. 0). Os dados decodificados são conectados aos 7 pinos de saída que acionam os LEDs do display de 7 segmentos.

3 Interface com o *Control*

Implementou-se a classe ScoreDisplay em *software*, recorrendo a linguagem Kotlin, o bloco Serial Receiver e o bloco Score Dispatcher em hardware, recorrendo a linguagem VHDL.

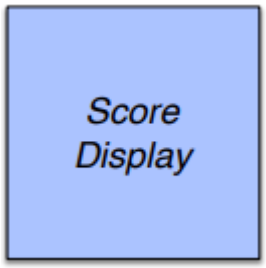

Figura 4- Classe Score Display

3.1 Classe Score Display

- Init, que inicia a classe
- SetScore, um comando para atualizar o valor value no mostrador da pontuação
- Off, envia um comando para ativar/desativar a visualização do mostrador de pontuação

4 Serial Receiver

O bloco Serial Receiver do módulo SLCDC é constituído por quatro blocos principais: i) um bloco de controlo; ii) um bloco conversor série paralelo; iii) um contador de bits recebidos; e iv) um bloco de validação de paridade, designados por Serial Control, Shift Register, Counter e Parity Check respetivamente.

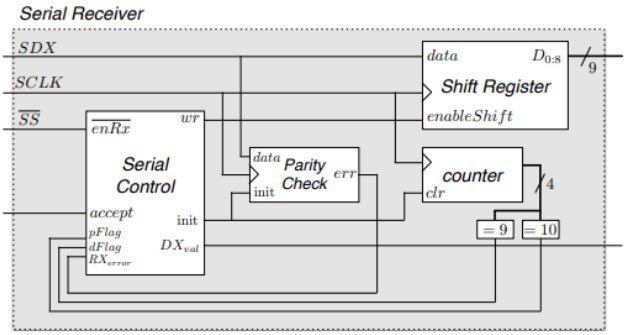

Figura 5 – Diagrama de blocos do bloco Serial Receiver

- O Serial Control é uma máquina de estados com 5 estados, Initializing, Starting, Waiting, Acepted e ItsDone. Começa por ativar o init no primeiro estado, dando assim init ao Parity Check (reset) e clear ao Counter, quando o sinal enRX tiver a '0', a máquina transita para o próximo estado Starting ativando o sinal wr, dando assim enableShift ao Shift Register.

  Para transitar para o próximo estado Waiting, é feita novamente a verificação do sinal enRX, se este tiver o valor ´1´retornará para o estado inicial Initializing, se o sinal enRX tiver a '0', e o sinal dFlag tiver a '1' será feita a transição.

  Após isto, é feita novamente a verificação do sinal enRX, caso este esteja ativo, retornará para o estado incial, Initializing se o sinal estiver a '0', é feita a verificação do sinal pFlag, se este tiver a '1', a máquina transita para o estado Acepted.

  Neste estado, é feita uma verificação do sinal RXerror onde, caso este esteja ativo, retornará para o estado incial Initializing, se o sinal estiver a '0', passará para o próximo e último estado ItsDone.

  Neste último estado o sinal DXval é ativo e, é feita uma verificação do sinal Accept, caso este esteja a '1' retornará ao estado inicial.

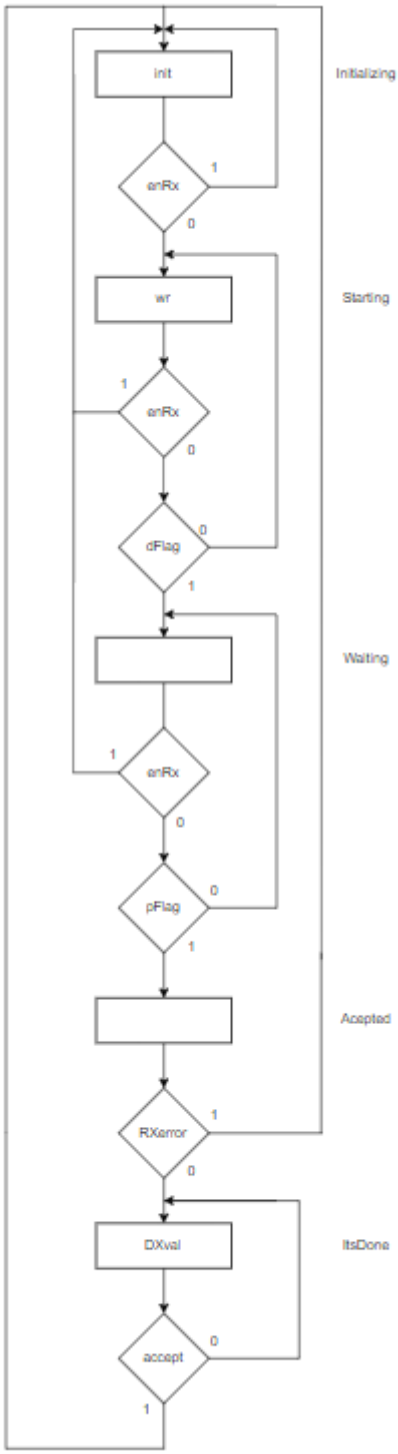

Figura 6 – Máquina de estados do Serial Control

5 Score Dispacher

O bloco Score Dispacher é responsável pela entrega das tramas válidas recebidas pelo bloco Serial Receiver ao Score Display, através da ativação do sinal W*rD*. Este bloco é apenas constituído pelo ScoreDisplayerControl.

O ScoreDisplayControl é uma máquina de estados com três estados, WaitingDval, ReceivingDval e DoneReceived. No primeiro estado WaitingDval, começa por verificar se o sinal Dval encontra-se a '1', se sim, significa que a trama foi recebida e a máquina transita para o próximo estado sendo este o ReceivingDval.

Neste estado, o sinal WrD é ativo e transita para o estado DoneReceived onde o sinal WrD deixa de estar ativo, deixando ativo o sinal done ativo, neste estado, o Dval é novamente verificado, porém neste caso se este tiver a '0' retornará para o estado inicial WaitingDval, significando que a trama já foi processada e a máquina poderá esperar por uma nova trama.

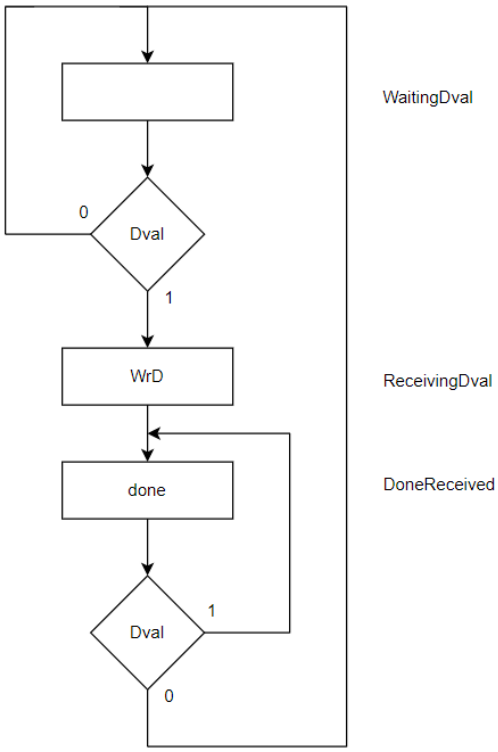

Figura 7- Máquina de estados Score Dispacher Control

6 Conclusão

Com a implementação deste módulo foi possível a interação com o ScoreDisplay a partir de dois blocos: Serial Receiver e Score Dispacher Control. Estes dois blocos têm a função de ler e transmitir a mensagem enviada pelo bloco de controlo até ao Score.

A. Descrição VHDL do bloco Serial Score Controller

Código VHDL do SSC:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity SSC is
port(
SS : in std_logic;
Reset : in std_logic;
SCLK : in std_logic;
Clk : in std_logic;
SDX : in std_logic;
WrD : out std_logic;
DoutSSC: out std_logic_vector(6 downto 0)
);
end SSC;
architecture strucutural of SSC is
component SerialReceiverSSC is
port(
Reset : in std_logic;
SDX : in std_logic;
Clk : in std_logic;
SCLK : in std_logic;
SS : in std_logic;
accept : in std_logic;
D : out std_logic_vector(6 downto 0);
DXval : out std_logic
);
end component;
component ScoreDispacher is
port(
```

```
    Dval : in std_logic;
    Din : in std_logic_vector(6 downto 0);
    WrD : out std_logic;
    Dout : out std_logic_vector(6 downto 0);
    done : out std_logic
    );
    end component;

    signal DXVal_exit : std_logic;
    signal D_exit : std_logic_vector(6 downto 0);
    signal done_exit : std_logic;

    begin

    SD: ScoreDispacher port map (Dval => DXVal_exit, Din => D_exit , done => done_exit, WrD => WrD, Dout =>
DoutSSC);

    SR : SerialReceiverSSC port map (Reset => Reset, SDX => SDX, Clk => Clk, SCLK => SCLK, SS => SS, accept =>
done_exit, D => D_exit, DXval => DXval_exit);

    end strucutural;
```

Código VHDL do ScoreDispatcher:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity ScoreDispacher is
        port(
                Dval : in std_logic;
                reset : in std_logic;
                clk : in std_logic;
                Din : in std_logic_vector(6 downto 0);
                WrD : out std_logic;
                Dout : out std_logic_vector(6 downto 0);
                done : out std_logic
);
end ScoreDispacher;
architecture structural of ScoreDispacher is
```

```
        component ScoreDispacherControl is
                port(
                        clk : in std_logic;
                        reset : in std_logic;
                        Dval : in std_logic;
                        WrD : out std_logic;
                        done : out std_logic
);
end component;
begin

SD: ScoreDispacherControl port map (Dval => Dval, done => done, WrD => WrD, reset => reset, clk => clk);

end structural;
```

Código VHDL do ScoreDispatcherControl:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity ScoreDispacherControl is
        port(
                clk : in std_logic;
                reset : in std_logic;
                Dval : in std_logic;
                WrD : out std_logic;
                done : out std_logic
);
end ScoreDispacherControl;
architecture behavior of ScoreDispacherControl is

type STATE_TYPE is (WaitingDval, ReceivingDval, DoneReceived);

signal currentState, NextState: STATE_TYPE;

begin
```

```
currentState <= WaitingDval when reset = '1' else NextState when rising_edge(clk);

GenerateNextState:
process(currentState, Dval)
begin
case currentState is
when WaitingDval => if (Dval = '1') then
NextState <= ReceivingDval;
else
NextState <= WaitingDval;
end if;
when ReceivingDval => NextState <= DoneReceived;

when DoneReceived =>                    if (Dval = '1') then
NextState <= DoneReceived;
else
NextState <= WaitingDval;
end if;
end case;
end process;

WrD <= '1' when (CurrentState = ReceivingDval)
else '0';

done <= '1' when (CurrentState = DoneReceived)
else '0';

end behavior;
```

Código VHDL do SerialReceiverSSC:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity SerialReceiverSSC is
port(
Reset : in std_logic;
SDX : in std_logic;
```

```
Clk : in std_logic;
SCLK : in std_logic;
SS : in std_logic;
accept : in std_logic;
D : out std_logic_vector(6 downto 0);
DXval : out std_logic
);
end SerialReceiverSSC;
architecture structural of SerialReceiverSSC is
component ShiftRegisterSerialReceiverSSC is
port(
data: in std_logic;
SCLK: in std_logic;
E: in std_logic;
D: out std_logic_vector(6 downto 0)
);
end component;
component Counter is
port(
PL:in std_logic;
CE:in std_logic;
CLK:in std_logic;
Data_in: in std_logic_vector(3 downto 0);
RESET: in std_logic;
TC: out std_logic;
Q:out std_logic_vector(3 downto 0)
);
end component;
component ParityCheck is
port(
data : in std_logic;
Sclk : in std_logic;
init : in std_logic;
err : out std_logic
);
end component;
component SerialControl is
port(
```

```
enRx : in std_logic;
accept : in std_logic;
pFlag : in std_logic;
dFlag : in std_logic;
RXerror : in std_logic;
wr : out std_logic;
init : out std_logic;
Reset : in std_logic;
Clk : in std_logic;
DXval : out std_logic
);
end component;
component Equal7 IS
port(
Q: in STD_LOGIC_VECTOR(3 downto 0);
TC: out STD_LOGIC
);
end component;
component Equal6 IS
port(
Q: in STD_LOGIC_VECTOR(3 downto 0);
TC: out STD_LOGIC
);
end component;

signal ParitCheck_err_exit : std_logic;
signal SerialControl_init_exit : std_logic;
signal SerialControl_wr_exit : std_logic;
signal CounterSerialReceiver_exit : std_logic_vector(3 downto 0);
signal dflag_6: std_logic;
signal pflag_7: std_logic;


begin

Shift: ShiftRegisterSerialReceiverSSC port map (data=>SDX ,SCLK=>SCLK ,E => SerialControl_wr_exit  ,D=>D);
```

Count: Counter port map (PL => '0', CE => '1', CLK => SCLK, Data_in => "0000", RESET => SerialControl_init_exit, Q => CounterSerialReceiver_exit);

Parity: ParityCheck port map (data => SDX, Sclk => SCLK, init => SerialControl_init_exit, err => ParitCheck_err_exit);

Controle: SerialControl port map (enRX => SS, accept => accept, pFlag => pflag_7 , dFlag => dflag_6, RXerror => ParitCheck_err_exit, wr => SerialControl_wr_exit, init => SerialControl_init_exit, DXval => DXval, Clk => Clk, Reset => Reset);

EQ1: Equal6 port map(Q => CounterSerialReceiver_exit,TC => dflag_6);

EQ2: Equal7 port map(Q => CounterSerialReceiver_exit ,TC => pflag_7);

end structural;

Código VHDL do ShiftRegisterSerialReceiverSSC:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity ShiftRegisterSerialReceiverSSC is
port(
data: in std_logic;
SCLK: in std_logic;
E: in std_logic;
D: out std_logic_vector(6 downto 0)
);
end ShiftRegisterSerialReceiverSSC;
architecture structural of ShiftRegisterSerialReceiverSSC is
component FFD IS
PORT(
CLK : in std_logic;
RESET : in STD_LOGIC;
SET : in std_logic;
D : IN STD_LOGIC;
EN : IN STD_LOGIC;
Q : out std_logic
);
end component;
```

```
signal saidaFFD1: std_logic;
signal saidaFFD2: std_logic;
signal saidaFFD3: std_logic;
signal saidaFFD4: std_logic;
signal saidaFFD5: std_logic;
signal saidaFFD6: std_logic;
signal saidaFFD7: std_logic;

begin

FFD1: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>data,     EN=>E,Q=>saidaFFD1);
FFD2: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD1,EN=>E,Q=>saidaFFD2);
FFD3: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD2,EN=>E,Q=>saidaFFD3);
FFD4: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD3,EN=>E,Q=>saidaFFD4);
FFD5: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD4,EN=>E,Q=>saidaFFD5);
FFD6: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD5,EN=>E,Q=>saidaFFD6);
FFD7: FFD port map(CLK=>SCLK,RESET=> '0' ,SET=> '0',D=>saidaFFD6,EN=>E,Q=>saidaFFD7);

D(6)<= saidaFFD1;
D(5)<= saidaFFD2;
D(4)<= saidaFFD3;
D(3)<= saidaFFD4;
D(2)<= saidaFFD5;
D(1)<= saidaFFD6;
D(0)<= saidaFFD7;

end structural;
```

Código VHDL do ScoreDisplay:

```
----------------------------------------------------------------------------
----------------------------------------------------------------------------
-- Project     : Score Display (Space Invaders Game)
-- Affiliations: DEETC, ISEL - IPL
-- Funding     : -
----------------------------------------------------------------------------
-- File        : scoreDisplay.vhd
-- Author(s)   : Pedro Miguens Matutino
-- Date        : 2024/02/16
----------------------------------------------------------------------------
```

```
-- Copyright (c) 2024 Pedro Miguens Matutino
----------------------------------------------------------------------------
-- Description :
-- .
----------------------------------------------------------------------------
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity scoreDisplay is
port(	set	: in std_logic;
		cmd	: in std_logic_vector(2 downto 0);
		data	: in std_logic_vector(3 downto 0);
		HEX0	: out std_logic_vector(7 downto 0);
		HEX1	: out std_logic_vector(7 downto 0);
		HEX2	: out std_logic_vector(7 downto 0);
		HEX3	: out std_logic_vector(7 downto 0);
		HEX4	: out std_logic_vector(7 downto 0);
		HEX5	: out std_logic_vector(7 downto 0)
		);
end scoreDisplay;

architecture structural of scoreDisplay is

component reg_4bit is
port(	clk : in std_logic;
		reset : in std_logic;
		set : in std_logic;
		d : in std_logic_vector(3 downto 0);
		en : in std_logic;
		q : out std_logic_vector(3 downto 0)
		);
end component;

component FFDScoreDisplay is
port(	clk : in std_logic;
		reset : in std_logic;
		set : in std_logic;
```

```
                d : in std_logic;
                en : in std_logic;
                q : out std_logic
                );
end component;

component dec2hex is
port(   d : in std_logic_vector(3 downto 0);
                clear : in std_logic;
                dout: out std_logic_vector(7 downto 0)
                );
end component;

component dec_3_8 is
port( addr      :       in std_logic_vector(2 downto 0);
                en              :       in std_logic;
                dout    :       out std_logic_vector(7 downto 0)
                );
end component;

type register_array is array (0 to 5) of std_logic_vector(3 downto 0);
signal reg_values, out_values : register_array;
type seg_array is array (0 to 5) of std_logic_vector(7 downto 0);
signal display_values : seg_array;
signal en_digit : std_logic_vector(7 downto 0);
signal clear : std_logic;
begin

decoder : dec_3_8 port map(addr => cmd, en => '1', dout => en_digit);

circuit_gen : for ii in 0 to 5 generate
        in_reg          : reg_4bit      port map(clk => set, reset => '0' , set => '0', d => data, en => en_digit(ii), q =>
reg_values(ii));
        out_reg         : reg_4bit      port map(clk => set, reset => '0' , set => '0', d => reg_values(ii), en => en_digit(6),
q => out_values(ii));
        hex_digit       : dec2hex       port map(d => out_values(ii),clear => clear, dOut => display_values(ii));
end generate circuit_gen;
```

```
clear_reg: FFDScoreDisplay port map ( clk => set, reset => '0' , set => '0', en => en_digit(7), d => data(0), Q => clear);

HEX0 <= display_values(0);
HEX1 <= display_values(1);
HEX2 <= display_values(2);
HEX3 <= display_values(3);
HEX4 <= display_values(4);
HEX5 <= display_values(5);

end structural;
```

Código VHDL do ParityCheck:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity ParityCheck is
	port(
		data : in std_logic;
		Sclk : in std_logic;
		init : in std_logic;
		err : out std_logic
);
end ParityCheck;
architecture structural of ParityCheck is

component Counter is
port(
	PL:in std_logic;
	CE:in std_logic;
	CLK:in std_logic;
	Data_in: in std_logic_vector(3 downto 0);
	RESET: in std_logic;
	TC: out std_logic;
	Q:out std_logic_vector(3 downto 0)
	);
end component;

signal SaidaB : std_logic_vector(3 downto 0);
```

```
begin

CounterUp : Counter port map (PL => '0', CE => data, CLK => Sclk, Data_in => "0000", Q => SaidaB, RESET => init);
err <= SaidaB(0);

end structural;
```

Código VHDL do SerialControl:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity SerialControl is
        port(
                enRx : in std_logic;
                RESET : in std_logic;
                accept : in std_logic;
                pFlag : in std_logic;
                dFlag : in std_logic;
                RXerror : in std_logic;
                clk : in std_logic;
    wr : out std_logic;
                init: out std_logic;
                DXval : out std_logic
);
end SerialControl;

architecture behavioral of SerialControl is

type STATE_TYPE is (Intializing, Starting, Waiting, Acepted, ItsDone);

signal currentState, NextState: STATE_TYPE;

begin

currentState <= Intializing when RESET = '1' else NextState when rising_edge(clk);

GenerateNextState:
process (currentState, enRx, accept, pFlag, RXerror, dFlag)
```

```
        begin
                case currentState is
                        when Intializing   => if (enRx = '0') then
                                                        NextState       <=
Starting;
                                                else
                                                        NextState       <=
Intializing;
                                                end if;

                        when Starting                   => if (enRx = '1') then
                                                        NextState       <=
Intializing;
                                                elsif (dFlag = '1') then
                                                        NextState       <=
Waiting ;
                                                else
                                                        NextState       <=
Starting;
                                                end if;

                        when Waiting                    => if (enRx = '1') then
                                                        NextState       <=
Intializing;
                                                elsif(pFlag = '1') then
                                                        NextState       <=
Acepted;
                                                else
                                                        NextState       <=
Waiting;
                                                end if;

                        when Acepted                            => if(RXerror = '0') then
                                                        NextState       <=
ItsDone;
                                                else
                                                        NextState       <=
Intializing;
```

```
                                                                    end if;

                    when ItsDone            => if(accept = '1') then
                                                        NextState   <=
Intializing;
                                                    else
                                                        NextState   <=
ItsDone;
                                                    end if;

        end case;
end process;

wr <= '1' when(currentState = Starting) else '0';
init <= '1' when(currentState = Intializing) else '0';
DXval <= '1' when (currentState = ItsDone) else '0';

end behavioral;
```

B. Código *Kotlin – Score Display*

```kotlin
object ScoreDisplay { // Controla o mostrador de pontuação.
    const val SS_SCORE = 0x02
    const val SCLK_SCORE = 0x10
    const val SDX_SCORE = 0x08

    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() {
        SerialEmitter.init()
    }




    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int) { //ainda n esta validado
        val tamanho = value.toString().length
        var number = tamanho - 1
        var count = 0
        while (count < tamanho) {
            val bin = value.toString()[count].digitToInt()
            val valor = bin.shl(3) + number
```

```
            SerialEmitter.send(SerialEmitter.Destination.SCORE, valor,7)
            number-=1
            count+=1
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE,0b0000110,7)
    }
    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean) {
        if (value) {
            SerialEmitter.send(addr = SerialEmitter.Destination.SCORE, 0b0001111 ,
7)
            //tem haver com a tabela do projeto Tabela1 - Modulo Score Display
        } else {
            SerialEmitter.send(addr = SerialEmitter.Destination.SCORE, 0b0000111 ,
7)
        }
    }
}
```