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, conforme representado na figura 1.

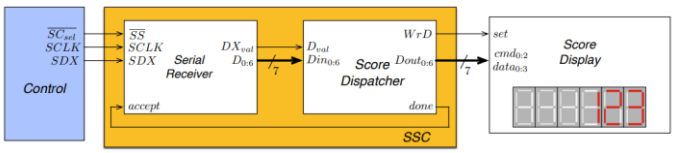

Figura 1 – Diagrama de blocos do módulo *Serial Score Controller*

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 ilustrado na Figura 2. Os três primeiros bits de informação, indicam o comando a realizar no mostrador de pontuação, segundo a Tabela 1. 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.

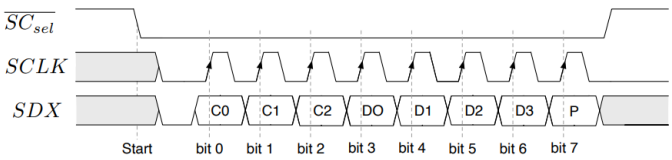

Figura 2- Protocolo de comunicação do módulo Serial Score Controller

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*

Tabela 1 – Comandos do módulo Score Display

1 Serial Receiver

O bloco *Serial Receiver* do módulo *SSC* deve ser implementado adotando, com as devidas adaptações, uma arquitetura similar à do bloco *Serial Receiver* do módulo *SLCDC*, neste caso adaptando a estrutura para a receção de sete (7) *bits* de informação em vez de nove (9) *bits*.

Para isso sabíamos que teríamos de fazer alterações no *shift register* bem como nos comparadores do que sai do *counter*. O *shift register* passaria a guardar a trama composta por 7 *bits* e os comparadores iriam comparar o output do *counter* com o valor 7 e o valor 8. Fora isto tudo o resto do hardware deste componente irá permanecer igual ao *Serial Receiver do LCD*.

2 Score Dispatcher

Após a receção de uma trama válida (proveniente do bloco *Serial Receiver*), o bloco *Score Dispatcher*, deverá proceder à atuação do comando recebido sobre o mostrador de pontuação.
Na figura 1, podemos observar o *ASM-chart* do componente em questão.

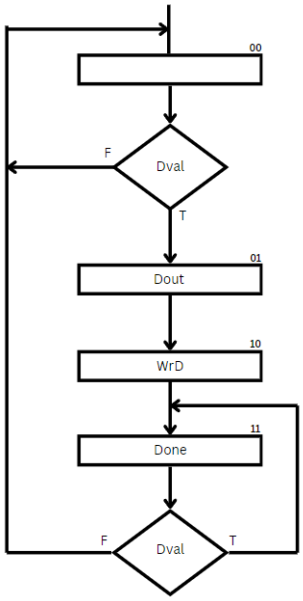

Figura 3 – Máquina de estados do bloco *Score Dispatcher*

Esta máquina segue o mesmo comportamento que o *LCD Dispatcher*.

Primeiramente ele terá de detetar *Dval*, depois ele ira ler ou registar *Din*, colocar *Dout* na saída, fazer *clock* do sinal *WrD*, ativar done, esperar que *Dval* esteja a 0 e tirar *done ao verificar Dval*.

3 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 4.

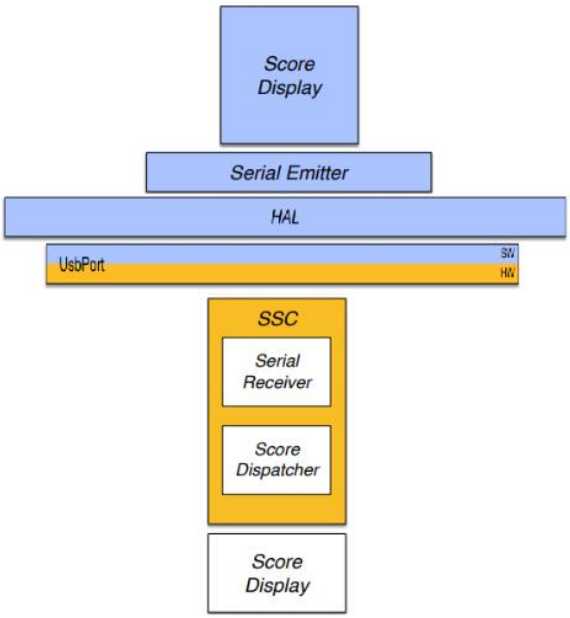

Figura 4 – Diagrama lógico do módulo *Control* de interface com o módulo *Serial Score Controller*

As classes *Score* Display, Serial Emitter desenvolvidas são descritas nas secções 3.1 e 3.2, e o código fonte desenvolvido nos Anexos I e J, respetivamente.

3.1 Classe Serial Emitter

Este Módulo tem como objetivo enviar tramas para os diferentes módulos Serial Receiver.
Init() – Iniciar o usbport para escrever.
send(addr: Destination, data: Int, size : Int) - enviar uma trama para o SerialReceiver identificado o destino em addr,os bits de dados em 'data' e em size o número de bits a enviar.

3.2 Classe *Score Display*

Este Módulo tem como objetivo controlar o placar de pontuação.

init() – Inicia a class com os valores iniciais.

sendScoreBit(value:Int) – Envia para o ScoreDisplay o value.

setScore(value: Int) - Envia comando para atualizar o valor do mostrador de pontuação.

off(value: Boolean) - Envia comando para desativar/ativar a visualização do mostrador de pontuação.

4 Conclusões

A implementação do módulo Serial Score Controller (SSC) integra a interface com o placar de pontuação, permitindo a receção de dados em série e a sua subsequente entrega ao display. O SSC recebe uma mensagem composta por sete bits de informação e um bit de paridade par. Os três primeiros bits determinam o comando, enquanto os quatro restantes correspondem ao campo de dados. O bloco Score Dispatcher é responsável por executar os comandos no placar após a receção de uma trama válida, sinalizando ao Serial Receiver que o processo foi concluído.

O bloco Serial Receiver, adaptado do módulo SLCDC, ajusta o seu registo de deslocamento e os comparadores para manipular tramas de sete bits. O Score Dispatcher segue uma lógica similar ao LCD Dispatcher, detetando sinais de validação, registando e enviando dados conforme necessário.

A interface com o módulo Control foi implementada em Java (Kotlin), consistindo nas classes Serial Emitter e Score Display. A classe Serial Emitter inicializa a comunicação via porta USB e envia tramas para os Serial Receivers. A classe Score Display gerencia o placar de pontuação, permitindo enviar bits específicos, atualizar o placar e controlar a visualização.

A arquitetura e a implementação do módulo SSC são coerentes e bem estruturadas, facilitando a comunicação e controle eficientes entre os componentes de hardware e software envolvidos no sistema de pontuação.

A. Descrição VHDL do bloco Serial Score Controller

```
library ieee;
use ieee.std_logic_1164.all;

entity SerialScoreController is
port(
SDX, SCLK, SS, Reset, clk: in std_logic;
Dout: out std_logic_vector(6 downto 0);
WrD: out std_logic
);
end SerialScoreController;

architecture behavioral of SerialScoreController is

component SerialReceiver is
port(
SDX, SCLK, SS, accept, clk, Reset: in std_logic;
D: out std_logic_vector(6 downto 0);
DXval: out std_logic
);
end component;

component Score_Dispatcher is
port(
Din: in std_logic_vector(6 downto 0);
Dval, clk, Reset: in std_logic;
WrD, done: out std_logic;
Dout: out std_logic_vector(6 downto 0)
);
end component;



signal term0: std_logic_vector(6 downto 0);
signal term1, term2: std_logic;


begin

SerialReceiver_U0: SerialReceiver port map(
SDX => SDX,
SCLK => SCLK,
SS => SS,
accept => term2,
Reset => Reset,
CLK => CLK,
D => term0,
DXval => term1
);


Score_Dispatcher_U0: Score_Dispatcher port map(
Din => term0,
Dval => term1,
clk => clk,
Reset => Reset,
WrD => WrD,
```

```
done => term2,
Dout => Dout
);


end behavioral;
```

B. Descrição VHDL do bloco Serial Receiver

```
library ieee;
use ieee.std_logic_1164.all;

entity SerialReceiver is
port(
SDX, SCLK, SS, accept, Reset, clk: in std_logic;
D: out std_logic_vector(6 downto 0);
DXval: out std_logic
);
end SerialReceiver;

architecture behavioral of SerialReceiver is

component ParityCheck is
port(
data, init, clk: in std_logic;
err: out std_logic
);
end component;

component Counter is
port(
Clear, clk: in std_logic;
Q: out std_logic_vector(3 downto 0)
);
end component;

component ShiftRegister is
port(
data : in std_logic;
clk : in std_logic;
enableShift : in std_logic;
reset : in std_logic;
D : out std_logic_vector(6 downto 0)
);
end component;

component SerialControl is
port(
Reset, clk, enRX, accept, dFlag, pFlag, RXerror: in std_logic;
wr, init, DXval: out std_logic
);
end component;

component Compare8 is
port(
Data: in std_logic_vector(3 downto 0);
R: out std_logic
);
end component;
```

```
component Compare7 is
port(
Data: in std_logic_vector(3 downto 0);
R: out std_logic
);
end Component;


signal term0, term1, term2, term3, term4 : std_logic;
signal term5 : std_logic_vector(3 downto 0);

begin

SerialControl_U0: SerialControl port map(
Reset => Reset,
clk => clk,
enRX => SS,
accept => accept,
dFlag => term0,
pFlag => term1,
RXerror => term2,
wr => term3,
init => term4,
DXval => DXval
);

ParityCheck_U0: ParityCheck port map(
Data => SDX,
Init => term4,
clk => SCLK,
err => term2
);

Counter_U0: Counter port map(
Clear => term4,
clk => SCLK,
Q => term5
);

Compare8_U0: Compare8 port map(
Data => term5,
R => term1
);


Compare7_U0: Compare7 port map(
Data => term5,
R => term0
);

ShiftRegister_U0: ShiftRegister port map(
Data => SDX,
clk => SCLK,
enableShift => term3,
Reset => Reset,
D => D
);
```

```
end behavioral;
```

C. Descrição VHDL do bloco Serial Control

```
library ieee;
use ieee.std_logic_1164.all;

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

architecture behavioral of SerialControl is

type STATE_TYPE is (STATE_INIT, STATE_WRITING, STATE_ERRORCHECK, STATE_WAITINGACCEPT);

signal CurrentState, NextState : STATE_TYPE;

begin

-- Flip-Flop's
CurrentState <= STATE_INIT when Reset = '1' else NextState when rising_edge(clk);

--Generate Next State
GenerateNextState:
process(CurrentState, enRX, accept, dFlag, pFlag, RXerror)

        begin

                case CurrentState is

                        when STATE_INIT => if(enRX = '0' and accept = '0') then
                                                NextState <= STATE_WRITING;
                                                else NextState <= STATE_INIT;
                                        end if;

                        when STATE_WRITING => if(dFlag = '1') then
                                                        NextState <= STATE_ERRORCHECK;
                                                                else if(enRX = '0') then
                                                        NextState <= STATE_WRITING;
                                                                else NextState <= STATE_INIT;
                                                        end if;
                                                end if;

                        when STATE_ERRORCHECK =>if(enRX = '1') then
                                                        if(pFlag = '1') then
                                                                if(RXerror = '0') then
                                                                NextState <= STATE_WAITINGACCEPT;
                                                                else NextState <= STATE_INIT;
                                                                end if;
                                                        else NextState <= STATE_INIT;
                                                        end if;
                                                else NextState <= STATE_ERRORCHECK;
                                                end if;

                        when STATE_WAITINGACCEPT => if(accept = '1') then
                                                        NextState <= STATE_INIT;
```

```
		else NextState <= STATE_WAITINGACCEPT;

		end if;
			end case;

		end process;

wr <= '1' when (CurrentState = STATE_WRITING) else '0';
init <= '1' when (CurrentState = STATE_INIT) else '0';
DXval <= '1' when (CurrentState = STATE_WAITINGACCEPT) else '0';

end behavioral;
```

D. Descrição VHDL do bloco Parity Check

```
library ieee;
use ieee.std_logic_1164.all;

entity ParityCheck is
port(
data, init, clk: in std_logic;
err: out std_logic
);
end ParityCheck;

architecture behavioral of ParityCheck is

component ParityCheck_Counter is
port(
init, data, clk: in std_logic;
err: out std_logic
);
end component;

begin

ParityCheckCounter_U0: ParityCheck_Counter port map(
init => init,
data => data,
clk => clk,
err => err
);

end behavioral;
```

E. Descrição VHDL do bloco Shift Register

```
library ieee;
use ieee.std_logic_1164.all;

entity ShiftRegister is
port(
data : in std_logic;
clk : in std_logic;
enableShift : in std_logic;
Reset : in std_logic;
D : out std_logic_vector(6 downto 0)
```

```
);
end ShiftRegister;

architecture behavioral of ShiftRegister is

component ShiftRegister_FFD 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 term0, term1, term2, term3, term4, term5, term6: std_logic;

begin

FFD0: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => data,
EN => enableShift,
Q => term0
);

FFD1: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => term0,
EN => enableShift,
Q => term1
);

FFD2: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => term1,
EN => enableShift,
Q => term2
);

FFD3: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => term2,
EN => enableShift,
Q => term3
);

FFD4: ShiftRegister_FFD port map(
clk => clk,
```

```
Reset => Reset,
Set => '0',
D => term3,
EN => enableShift,
Q => term4
);

FFD5: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => term4,
EN => enableShift,
Q => term5
);

FFD6: ShiftRegister_FFD port map(
clk => clk,
Reset => Reset,
Set => '0',
D => term5,
EN => enableShift,
Q => term6
);

D(0) <= term6;
D(1) <= term5;
D(2) <= term4;
D(3) <= term3;
D(4) <= term2;
D(5) <= term1;
D(6) <= term0;


end behavioral;
```

F. Descrição VHDL do bloco Counter

```
library ieee;
use ieee.std_logic_1164.all;

entity Counter is
port(
clear, clk: in std_logic;
Q: out std_logic_vector(3 downto 0)
);
end Counter;

architecture behavioral of Counter is

component Counter_Adder port(
A : in std_logic_vector(3 downto 0);
B : in std_logic_vector(3 downto 0);
CYi : in std_logic;
S : out std_logic_vector(3 downto 0)
);
end component;
```

```
component Counter_Reg port(
D : in std_logic_vector(3 downto 0);
CLK : in std_logic;
E : in std_logic;
Reset : in std_logic;
Q : out std_logic_vector(3 downto 0)
);
end component;

signal term0, term1 : std_logic_vector(3 downto 0);

begin

Counter_Adder_U0: Counter_Adder port map(
A => term0,
B(3) => '0',
B(2) => '0',
B(1) => '0',
B(0) => '1',
CYi => '0',
S => term1
);


Counter_Reg_U0: Counter_Reg port map(
D => term1,
clk => clk,
E => '1',
Reset => clear,
Q => term0
);

Q <= term0;

end behavioral;
```

G. Descrição VHDL do bloco Compare7

```
library ieee;
use ieee.std_logic_1164.all;

entity Compare7 is
port(
Data: in std_logic_vector(3 downto 0);
R: out std_logic
);
end Compare7;

architecture behavioral of Compare7 is

begin

R <= not Data(3) and Data(2) and Data(1) and Data(0);

end behavioral;
```

H. Descrição VHDL do bloco Compare8

```
library ieee;
use ieee.std_logic_1164.all;

entity Compare8 is
port(
Data: in std_logic_vector(3 downto 0);
R: out std_logic
);
end Compare8;

architecture behavioral of Compare8 is

begin

R <= Data(3) and not Data(2) and not Data(1) and not Data(0);

end behavioral;
```

I. Codigo Kotlin – Score Display

```
object ScoreDisplay { // Controla o mostrador de pontuação.
    val updatemask = 96
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() {
        off(false)
        for (command in 5 downTo 0)
            sendScoreBit(0b1111 + command.shl(4))
            sendScoreBit(updatemask)

    }
    private fun sendScoreBit(value:Int){
        SerialEmitter.send(SerialEmitter.Destination.SCORE, value, 8)
    }


    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int){
        var temp_value = value
        var divider = 100_000
        var encounteredNonZero = false

        for(command in 5 downTo 0){
            val sub = (temp_value/divider)*divider
            if (temp_value/divider != 0 || encounteredNonZero){
                encounteredNonZero = true
                sendScoreBit((temp_value/divider)+command.shl(4))}
            else
                sendScoreBit((0b1111)+command.shl(4))
            temp_value -= sub
            divider /= 10
        }
        sendScoreBit(updatemask)
    }

    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean){
```

```
            val maskon = 112
            val maskoff = 113
            if(value){
               sendScoreBit(maskoff)}
            else{
               sendScoreBit(maskon)
            }
         }
      }
      fun main(){
         SerialEmitter.init()
         ScoreDisplay.init()
         ScoreDisplay.setScore(123)
      }
```

J. Codigo Kotlin – Serial Emitter

```
object SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.
   enum class Destination {LCD, SCORE}

   // Inicia a classe
   fun init(){
      val mask_clr = 6
      HAL.setBits(mask_clr)
   }


// Envia uma trama para o SerialReceiver identificado o destino em addr,os bits de dados em 'data' e em size o número de bits a enviar.
   fun send(addr: Destination, data: Int, size : Int){
      val mask_data = 1
      var counter = 0
      val mask_LCD = 2
      val mask_SCORE = 4
      val mask_clock = 8
      var data_temp = data

      if(addr == Destination.LCD)
         HAL.clrBits(mask_LCD)
      else if(addr == Destination.SCORE)
         HAL.clrBits(mask_SCORE)

      for(i in 0 until size-1){
         HAL.clrBits(mask_clock)
         if (data_temp%2 ==0){
            HAL.writeBits(mask_data, 0)
            data_temp = data_temp.shr(1)
         }
         else {
            HAL.writeBits(mask_data, 1)
            data_temp = data_temp.shr(1)
            counter++
         }
         HAL.setBits(mask_clock)
      }

      HAL.clrBits(mask_clock)
      if (counter%2==0){
         HAL.writeBits(mask_data, 0)
```

```
        }else{
            HAL.writeBits(mask_data, 1)
        }
        HAL.setBits(mask_clock)
        HAL.setBits(mask_LCD)
        HAL.setBits(mask_SCORE)
    }
}
```