O módulo *Serial Score Controller (SSC)* é constituído por dois blocos principais:

i) *Serial Receiver (SSR)*

ii) *Score Dispatcher (SD)*

No presente relatório apresenta-se a solução dos dois blocos, mencionados, interface com o mostrador de pontuação (*Score Display*), onde serão visíveis as pontuações e outras informações registadas relativas aos jogos efetuados.

Enquanto o bloco *SSR é* responsável pela recolha da informação, controladamente, em formato *serial*, bit a bit e, sempre que ocorram dados válidos, transmite/entrega cada trama de 7 bits (onde, no bit zero (0) se encontra o valor *RS* e nos seis (6) bits restantes de dados), no formato *paralell*, ao *SD*.

O Bloco *SD* é responsável pela comunicação, ao mostrador *LCD*.

A implementação destes dois blocos permite a independência quanto à escrita no bloco *SSR*, que ao despachar a informação para o bloco *SD*, pode voltar a receber nova série de dados, enquanto o *SD* resolve a apresentação da informação no *LCD*.

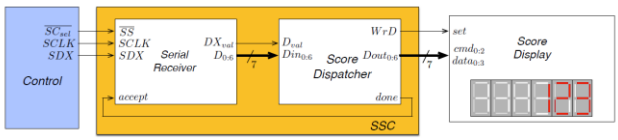

Figure 1

Diagrama de blocos do módulo Serial Score Controller

1 Serial Receiver

O Bloco *SSR*, conforme Figure 1 é implementado com recurso a:

i) um bloco de controlo, designado por *Serial Control*, que efetua o controlo e validação dos dados recebidos no bloco e a sua transmissão para o bloco;

ii) *SSR*, conforme o diagrama de blocos representado na ??

iii) um bloco conversor série paralelo, designado por *Shift Register*, está implementado com recurso a um *Shift Register* que tem por encargo a transformação dos bits rececionados, conforme requerido, e sua transmissão ao bloco *D*;

iv) um contador, designado por *Counter*, de 3 bits recebidos, cujo objetivo é o controlo e estabilidade dos dados no conversor série paralelo, ao receber o sétimo (7.º) bit, altera o valor da *dFlag* para um (1) e o sinal e o sinal *Wr* passa a 1 e coloca a trama de 7 bits em $D_{0..6}$ e aguarda pela validação da paridade, para que seja possível passar os dados para o bloco *SSR*. Quando este contador recebe o oitavo (8.º) bit, activa a *pFlag*, que passa a um (1), e ao ser analisado o sinal de *Rxerr*, caso esteja a um (1) ocorreu erro e é efetuado "reset", à informação contida neste bloco *LCDD*, que aguarda pela chegada de dados, mas caso seja zero (0), a informação ($D_{0..8}$ *e* DX_{val}) passa para o bloco *LCDSR* e aguarda pela conclusão do processo através da ativação do sinal *accept*. Fica assim disponível para processar mais dados;

v) um bloco de validação de paridade, designado *Parity Check*, que é um contador de 1 bit, que quando recebe o oitavo (8.º) bit, avalia se ocorreu erro de transmissão, com base no bit de paridade, no sinal data. Caso seja um (1) ocorreu um erro na transmissão, por alteração de um valor ímpar de bits, entre a origem e a sua receção e o processo é interrompido. Caso seja zero (0) a transmissão é processada.

vi) O bloco Score Serial Receiver, conforme Figure 2, foi construído tendo por modelo o bloco Serial Receiver de LCD, com alteração nos contadores, que são a 3 bits. Por ser solução semelhante, cuja implementação de código VHDL está em anexo A.

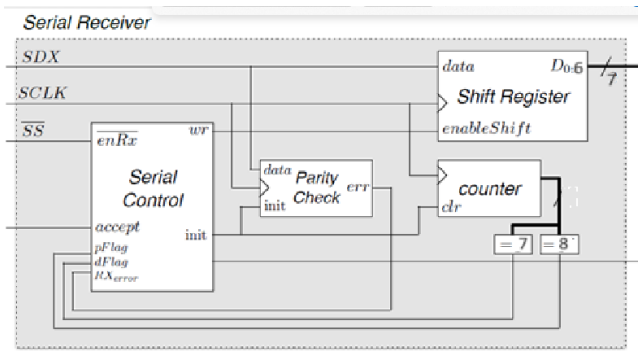

Figure 2

Diagrama de blocos do Bloco Score Serial Receiver

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figure 3 e foi resultado do exercício de avaliação dos estados dos sinais de entrada que garantem o normal e independente funcionamento do módulo principal Score *Serial Receiver*, de alguma forma já explanados na explicação dos propósitos dos restantes 3 módulos adjacentes a este.

A descrição hardware do bloco *Serial Control* em VHDL encontra-se no Anexo A.

Para garantir a transmissão de dados em série o o módulo *Serial Control utiliza o protocolo que se plasma na figura.*

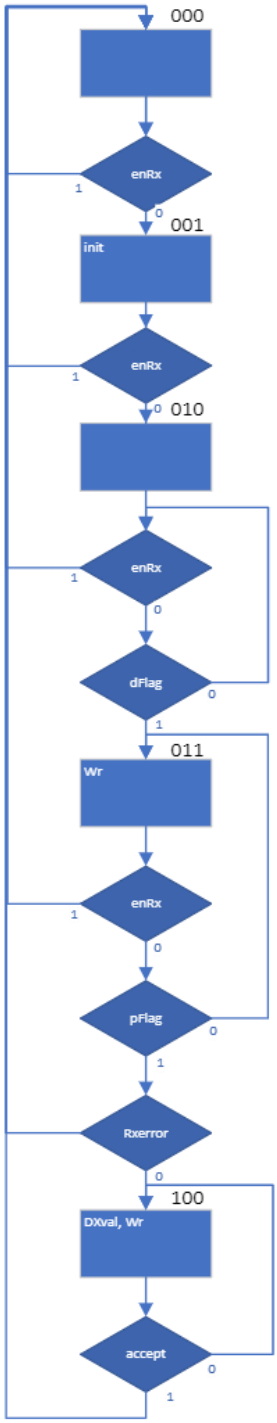

Figure 3

Máquina de estados do bloco Serial Control

2 *Score Dispatcher*

O bloco *SD* que garante a passagem de informação para o *LCD,* e desta forma garante que o bloco SSR possa estar disponível para receber informação. O processo enquanto o sinal de Dval (=0) continua num ciclo sem fim, até que ocorre a ativação do sinal Dval (=1) e assim vai transmitir os sinais de E e de 7 bits, onde o primeiro é de RS (que indica se a trama é de controlo ou de dados) e 6 bits de informação. eguarda que é implementado pela ASM *ASM-chart* na Figure 3 e cuja descrição de hardware em VHDL está no anexo A

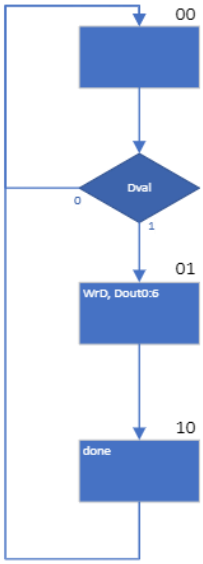

Figure 4

Máquina de estados do bloco Score Dispatcher

3 *Interface com o Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Kotlin e seguindo a arquitetura lógica apresentada no modelo da figura 14 do enunciado.

Embora utilize a classe LCD, como está incluída no relatório do módulo Serial LCD Controller, não será duplicada aqui, apenas será apresentada a classe *ScoreDisplay* desenvolvida são descritas em secção 3.1 e o código fonte desenvolvido é apresentado no anexo B.

3.1 Classe *SCORE DISPLAY*

Sendo a interface entre hardware e software utiliza a classe *HAL*, que virtualiza o acesso ao sistema *UsbPort*, e escreve no mostrador de pontuação, é constituída pela função main, e pelas seguintes outras funções:
Esta classe tem as seguintes funções:

- *i)* fun init(): Inicia a classe, estabelecendo os valores iniciais;
- *ii)* fun setScore(value: Int): Envia comando para atualizar o valor do mostrador de pontuação;
- *iii)* fun off(value: Boolean): Envia comando para desativar/ativar a visualização do mostrador de pontuação.

4 Conclusões

Nesta fase do projeto é possível garantir apenas a visualização de informação no simulador, o que indica que a componente do software não apresenta erros graves, mas ocorre alguma incorreção no Hardware (VHDL), nomeadamente na adaptação que decorre do desenvolvimento do bloco Score Dispacher, hoje concluído, que estamos a analisar.

A. Descrição VHDL do bloco Serial Score Controller

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

entity SSC is
port(
    clk: in std_logic;
    rst: in std_logic;
    SC_sel: in std_logic;
    SCLK: in std_logic;
    SDX: in std_logic;
    WrD: out std_logic;
    Dout: out std_logic_vector(6 downto 0);
    accept: in std_logic -- apenas para teste, remover quando o score dispatcher estiver
implementado
);
end SSC;


architecture structural of SSC is
component SerialReceiver is
port(
    rst: in std_logic;
    clk: in std_logic;
    SDX: 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;

begin

SReceiver: SerialReceiver port map(
    rst => rst,
    clk => clk,
    SDX => SDX,
    SCLK => SCLK,
    SS => SC_sel,
    accept => accept,
    DXval => WrD,
    D => Dout
);
```

B. Descrição VHDL do bloco Serial Score Receiver

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

entity SerialReceiver is
port(
    rst: in std_logic;
    clk: in std_logic;
    SDX: 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 SerialReceiver;

architecture structural of SerialReceiver is
component SerialControl is
port(
    rst: in std_logic;
    clk: in std_logic;
    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;
    DXval: out std_logic
);
end component;

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

component Cont is
port(
    R: in std_logic;
    CE: in std_logic;
    Clk: in std_logic;
    Q: out std_logic_vector(3 downto 0)
    );
end component;

component ShiftRegister is
port(
    CLK: in std_logic;
    RST: in std_logic;
    data: in std_logic;
```

```
    enable: in std_logic;
    D: out std_logic_vector(6 downto 0)
    );
end component;


signal Serr: std_logic;
signal Swr: std_logic;
signal Sinit: std_logic;
signal Scount: std_logic_vector(3 downto 0);
signal Sdflag: std_logic;
signal Spflag: std_logic;

begin

SC: SerialControl port map(
    rst => rst,
    clk => clk,
    enRX => SS,
    accept => accept,
    pflag => Spflag,
    dflag => Sdflag,
    RXerror => Serr,
    wr => Swr,
    init => Sinit,
    DXval => DXval
);

Pcheck: ParityCheck port map(
    data => SDX,
    clk => SCLK,
    init => Sinit,
    err => Serr
);

ShRegister: ShiftRegister port map(
    data => SDX,
    CLK => SCLK,
    RST => rst,
    enable => Swr,
    D => D
);

Counter: Cont port map(
    R => Sinit,
    Clk => SCLK,
    CE => '1',
    Q => Scount
);

Spflag <= Scount(1) and Scount(3);
Sdflag <= Scount(0) and Scount(3);

end structural;
```

C. Descrição VHDL do bloco Serial Score Control

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

entity SerialControl is
port(
    rst: in std_logic;
    clk: in std_logic;
    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;
    DXval: out std_logic
);
end SerialControl;

architecture behavioral of SerialControl is

type STATE_TYPE is (STATE_000, STATE_001, STATE_010, STATE_011, STATE_100);
signal CurrentState, NextState : STATE_TYPE;

begin

--Flip-Flop
CurrentState <= STATE_000 when rst ='1' else NextState when rising_edge(clk);

GenerateNextState:
process(CurrentState, enRX, dFlag, pFlag, RXerror, accept)
begin

case CurrentState is
    when STATE_000 => if (enRX = '1') then
                                                NextState <= STATE_000;
                                            else
                                                NextState <= STATE_001;
                                            end if;

    when STATE_001 => if (enRX = '1') then
                                                NextState <= STATE_000;
                                            else
                                                NextState <= STATE_010;
                                                end if;

    when STATE_010 => if (enRX = '1') then
                                                NextState <= STATE_000;
                                            elsif (dFlag = '0') then
                                                NextState <= STATE_010;
                                            else
                                                NextState <= STATE_011;
                                                end if;
    when STATE_011 => if (enRX = '1') then
                                                NextState <= STATE_000;
```

```
                                        elsif (pFlag = '0') then
                                               NextState <= STATE_011;
                                        elsif (RXerror = '1') then
                                               NextState <= STATE_000;
                                        else
                                               NextState <= STATE_100;
                                        end if;

    when STATE_100 => if (accept = '0') then
                                               NextState <= STATE_100;
                                        else
                                               NextState <= STATE_000;
                                        end if;

    end case;

end process;



init <= '1' when (CurrentState = STATE_001) else '0';
wr <= '1' when (CurrentState = STATE_011 or CurrentState = STATE_100) else '0';
DXval <= '1' when (CurrentState = STATE_100) else '0';

end behavioral;Descrição VHDL do bloco Serial Control
```

D. Descrição VHDL do bloco Score Dispatcher

(em branco)

A. Código Kotlin – Score Display

```
object ScoreDisplay { // Controla o mostrador de pontuação.
    val SCsel = 0x80
    val SCLK = 0x10
    val SCORE = SerialEmitter.Destination.SCORE
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init(){
        /*while (true){
            off(false)
            Thread.sleep(500)
            off(true)
            Thread.sleep(500)
        }*/

    }
    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int){
        var v = value
        var cmd = 0
        while (v != 0 && cmd < 6){
            var num = v % 10
            num = (cmd.shl(4)).or(num)
            SerialEmitter.send(SCORE, num, 7)
            cmd++
            v /= 10
        }
        while (cmd < 6){
            val num = (cmd.shl(4).or(0xF))
            SerialEmitter.send(SCORE, num, 7)
            cmd++
        }
        SerialEmitter.send(SCORE, 0x60, 7)

    }
    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean){
        if (value){
            SerialEmitter.send(SCORE, 0x71, 7)
        }
        else{
            SerialEmitter.send(SCORE, 0x70, 7)
        }
    }
}

fun main(){
    HAL.init()
    SerialEmitter.init()
    ScoreDisplay.init()
    ScoreDisplay.setScore(123)
}
```