1 Serial Score Controller

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.

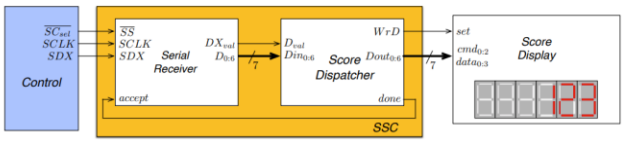

Figura 1 – Diagrama de blocos do módulo *SSC*

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 13. 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.

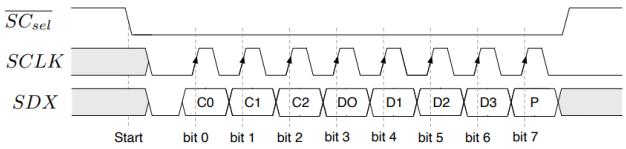

Figura 2 – Protocolo de comunicação com o módulo Serial LCD Controller

O emissor, realizado em software, quando pretende enviar uma trama para o módulo SSC promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente na linha $\overline{SCsel}$. Após a condição de início, o módulo SSC armazena os bits de dados da trama nas transições ascendentes do sinal *SCLK*.

1.1 Serial Receiver

O bloco *Serial Receiver* do módulo SSC 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. O bloco *Serial Receiver* deverá ser implementado com base no diagrama de blocos apresentado na Figura 3.

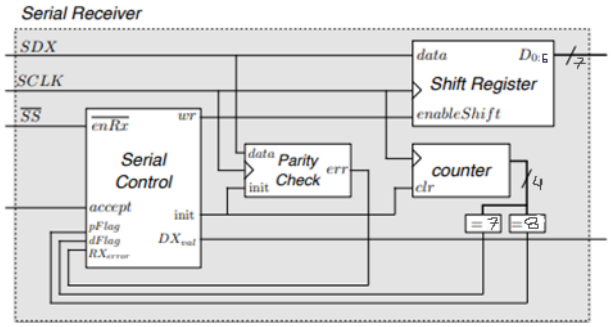

Figura 3 - Diagrama de blocos do bloco *Serial Receiver*

O bloco *Serial Receiver* foi implementado de acordo com o diagrama de blocos representado na Figura 3. Na implementação deste bloco foram usados: um contador de 4 bits (*counter*), que realiza a contagem do número de bits que já foram recebidos e quando receber um sinal de clear voltará a ter o valor lógico de 0; um controlador (*Serial_Control*), que será o responsável de verificar em que estado se encontra o *Serial Receiver* e fazer a passagem para outros estados; e finalmente um *shifter* bit a bit para a direita (*Shift Register*), que irá guardar os valores dos bits até recebermos 7 bits no total pela lógica da máquina de estados apresentada na Figura 4.

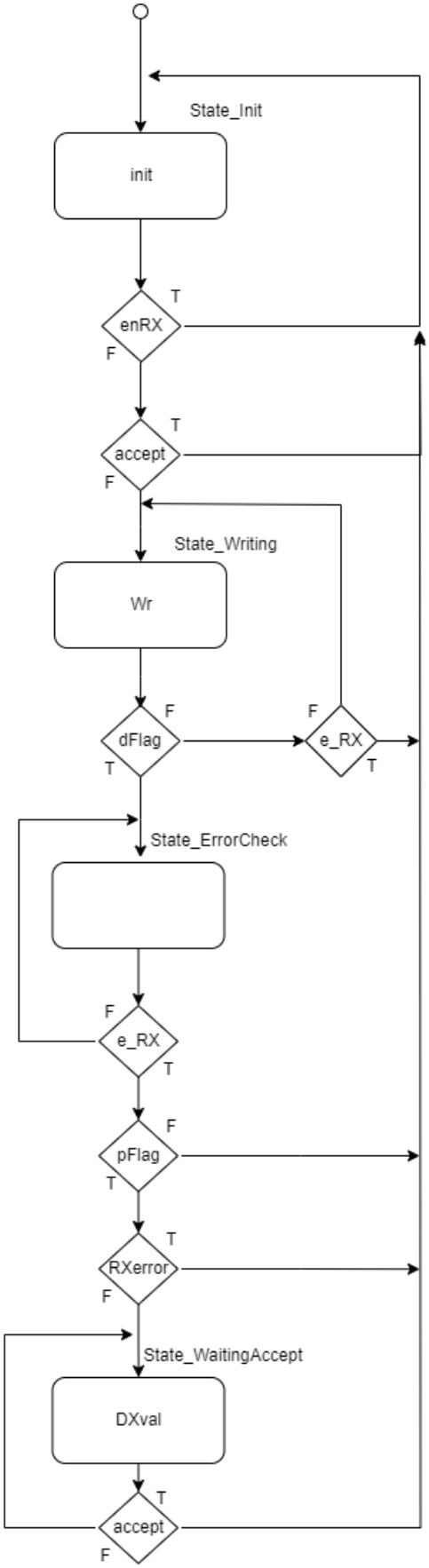

Figura 4 – Máquina de estados do bloco *Serial Control*

1.2 Score Dispatcher

O bloco *Score Dispatcher* é responsável pela entrega das tramas válidas recebidas pelo bloco *Serial Receiver* ao Score Display, através da ativação do sinal *WrL*. A receção de uma trama válida é sinalizada pela ativação do sinal D_{val}. O processamento das tramas recebidas pelo *Score Display* respeita os comandos definidos pelo fabricante, não sendo necessário esperar pela sua execução para libertar o canal de receção série. Assim, o bloco *Score Dispatcher* pode ativar, prontamente, o sinal *done* para notificar o bloco *Serial Receiver* que a trama já foi processada.

A máquina de estados inicialmente encontra-se no estado *STATE_WAITING* onde não ativa nenhum bit de saída até esta receber o valor lógico '1' no D_{val}. Caso não receba este valor, ficará presa no estado referido. O próximo estado será o *STATE_ENABLE* onde permite a escrita de dados e ativa o sinal *WrL*, após um ciclo de relógio passa para o estado *STATE_DONE*. Neste estado é ativado o sinal *Done* para indicar que já acabou de efetuar a escrita. Quando o sinal D_{val} voltar a ter o valor lógico de '0' volta para o estado *STATE_WAITING* caso contrário fica neste estado.

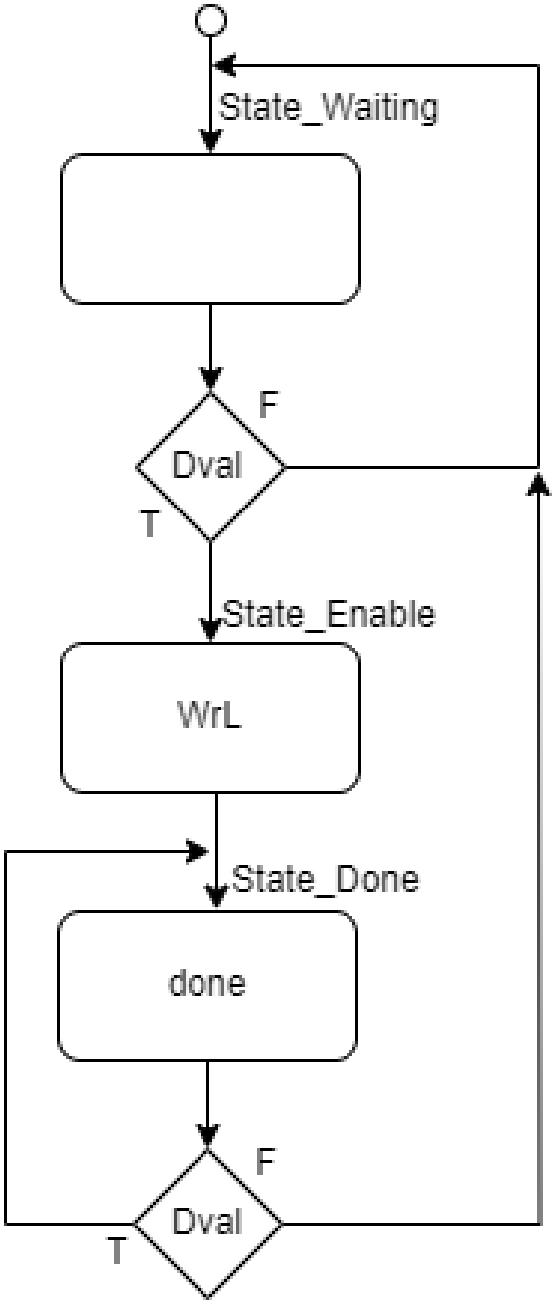

Figura 5 – Máquina de estados do Score_Dispatcher

2 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem Kotlin e seguindo a arquitetura lógica apresentada na Figura 6.

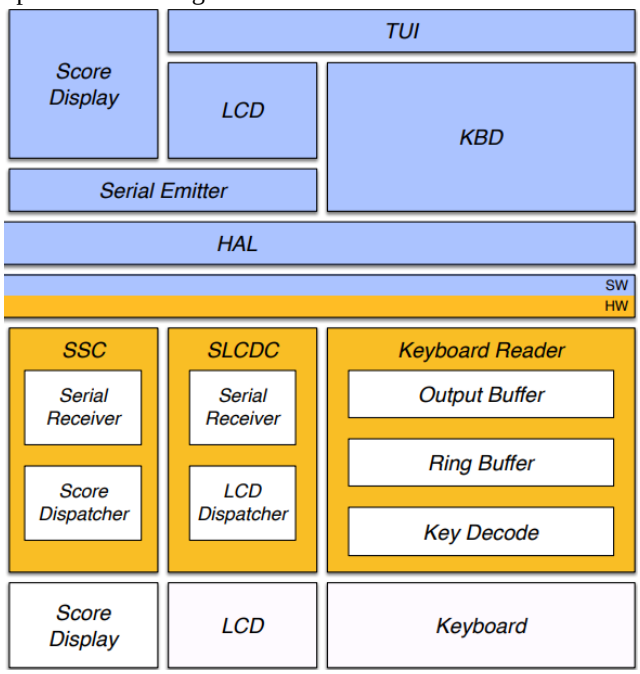

Figura 6 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*, o *SLCDC* e o *SSC*

As classes *Score Display* e *Serial Emitter* desenvolvidas são descritas nas secções 2.1 e 2.2, e o código fonte desenvolvido nos anexos "Score Display.kt" e "Serial Emitter.kt", respetivamente.

2.1 Score Display

Este módulo tem como objetivo efetuar o controlo do mostrador de pontuação.

A função init() inicializa este módulo; a função off(value: Boolean) é responsável por ativar ou desativar a visualização do mostrador de pontuação segundo um *boolean*; a função setScore(value: Int) é responsável pela atualização do valor do mostrador de pontuação; a função startAnimation e stopAnimation servem para atualizar o booleano que guarda a respetiva ação quanto à animação do *Score Display*; a função updateAnimation atualiza a "trama" da animação consoante o tempo; a função displayAlternatingShapes muda a forma a demonstrar no *Score Display*.

2.2 Serial Emitter

Este módulo tem como objetivo enviar tramas para os diferentes módulos *Serial Receiver*. Este módulo é utilizado para chegar aos mesmos objetivos que se chegaram no módulo *Score Display* quanto ao modo de envio de dados em série.

A função init() inicializa este módulo, inicializando o HAL, colocando o bit correspondente ao *SS* a '1' lógico e apaga os bits correspondentes ao *SCLK* e ao *SDX*.

A função send(addr: Destination, data: Int) envia uma trama para o *Serial Receiver* identificado o destino em *addr* e os bits de dados em '*data'* de forma que o *Serial Receiver* possa receber os mesmos.

3 Conclusões

Para a realização deste módulo foram utilizados o *Intellij* para o *Software* e o *Quartus Prime Lite* para o *Hardware*. Como o sistema está a ser feito por camadas o desenvolvimento foi mais rápido pois deu-nos uma maior independência. Foi necessário a implementação de atrasos em algumas funções, relacionadas com a manipulação de bits por exemplo a função SerialEmitter.send(), o mesmo foi necessário na função de inicialização que entre todos os comandos tem de ter um *delay* pois o hardware é mais lento do que o Software.

Descrição VHDL do bloco *Serial Receiver*

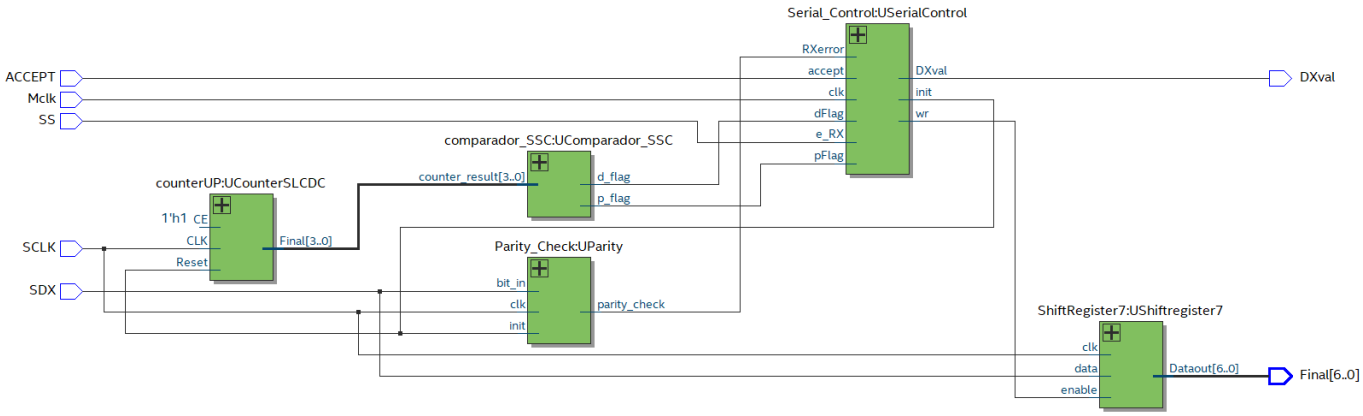

Figura 7: Descrição do bloco Serial Receiver do SSC

```
-- Entidade Serial_Receiver_SSC que define a interface do receptor serial
entity Serial_Receiver_SSC is
    port(
        SDX      : in std_logic; -- Entrada do dado serial
        SCLK     : in std_logic; -- Entrada do clock principal
        Mclk     : in std_logic; -- Entrada do clock secundário
        SS       : in std_logic; -- Entrada do bit enable
        ACCEPT   : in std_logic; -- Sinal do bit de aceitação
        DXval    : out std_logic; -- Saída do bit de validação
        Final    : out std_logic_vector(6 downto 0) -- Saída dos dados finais recebidos
    );
end entity Serial_Receiver_SSC;

architecture Serial_Receiver_arch of Serial_Receiver_SSC is

    -- Componente contador que conta de forma crescente
    component counterUP
        port(
            CE     : in std_logic; -- Entrada do bit de enable
            Reset  : in std_logic; -- Entrada do reset
            CLK    : in std_logic; -- Entradado clock
            Final  : out std_logic_vector(3 downto 0) -- Saída do contador
        );
    end component;
```

```
-- Componente de controle serial
component Serial_Control
   port(
      clk, e_RX, accept, dFlag, pFlag, RXerror : in std_logic;
      wr, init, DXval                          : out std_logic
   );
end component;

-- Componente shift regiser de deslocamento de 7 bits
component ShiftRegister7
   port (
      data    : in std_logic; -- Entrada do dado
      clk     : in std_logic; -- Entrada do clock
      enable  : in std_logic; -- Entrada do bit enable
      Dataout : out std_logic_vector(6 downto 0) -- Saída dos dados do registrador
   );
end component;

-- Componente de verificação de paridade
component Parity_Check
   port (
      clk         : in std_logic; -- Entrada do clock
      init        : in std_logic; -- Entrada do bit de inicialização
      bit_in      : in std_logic; -- Entrada do bit de paridade
      parity_check : out std_logic -- Saída de verificação de paridade
   );
end component;

-- Componente comparador
component comparador_SSC
   port (
      counter_result : in std_logic_vector(3 downto 0); -- Entrada do resultado do contador
      d_flag         : out std_logic;                   -- Saída da Flag de dado
      p_flag         : out std_logic                    -- Saída da Flag de paridade
   );
end component;

-- Declaração dos sinais internos
```

```
    signal dFLAG_s, pFLAG_s, RXerror_s, wr_s, init_s : std_logic;
    signal CounterResult : std_logic_vector(3 downto 0);

begin

    -- Instância do componente Serial_Control
    USerialControl : Serial_Control
        port map(
            e_RX   => SS,
            accept => ACCEPT,
            clk    => MCLK,
            pFlag  => pFLAG_s,
            dFlag  => dFLAG_s,
            RXerror=> RXerror_s,
            wr     => wr_s,
            init   => init_s,
            DXval  => DXval
        );

    -- Instância do componente Parity_Check
    UParity : Parity_Check
        port map(
            clk          => SCLK,
            init         => init_s,
            bit_in       => SDX,
            parity_check  => RXerror_s
        );

    -- Instância do componente counterUP
    UCounterSLCDC: counterUP
        port map(
            CE     => '1',
            Reset  => init_s,
            CLK    => SCLK,
            Final  => CounterResult
        );

    -- Instância do componente ShiftRegister7
```

```
    UShiftregister7 : ShiftRegister7
        port map(
            data    => SDX,
            clk     => SCLK,
            enable  => wr_s,
            Dataout => Final
        );

    -- Instância do componente comparador_SSC
    UComparador_SSC : comparador_SSC
        port map(
            counter_result => CounterResult,
            d_flag         => dFLAG_s,
            p_flag         => pFLAG_s
        );

end Serial_Receiver_arch;
```

Descrição VHDL do bloco Score Dispatcher

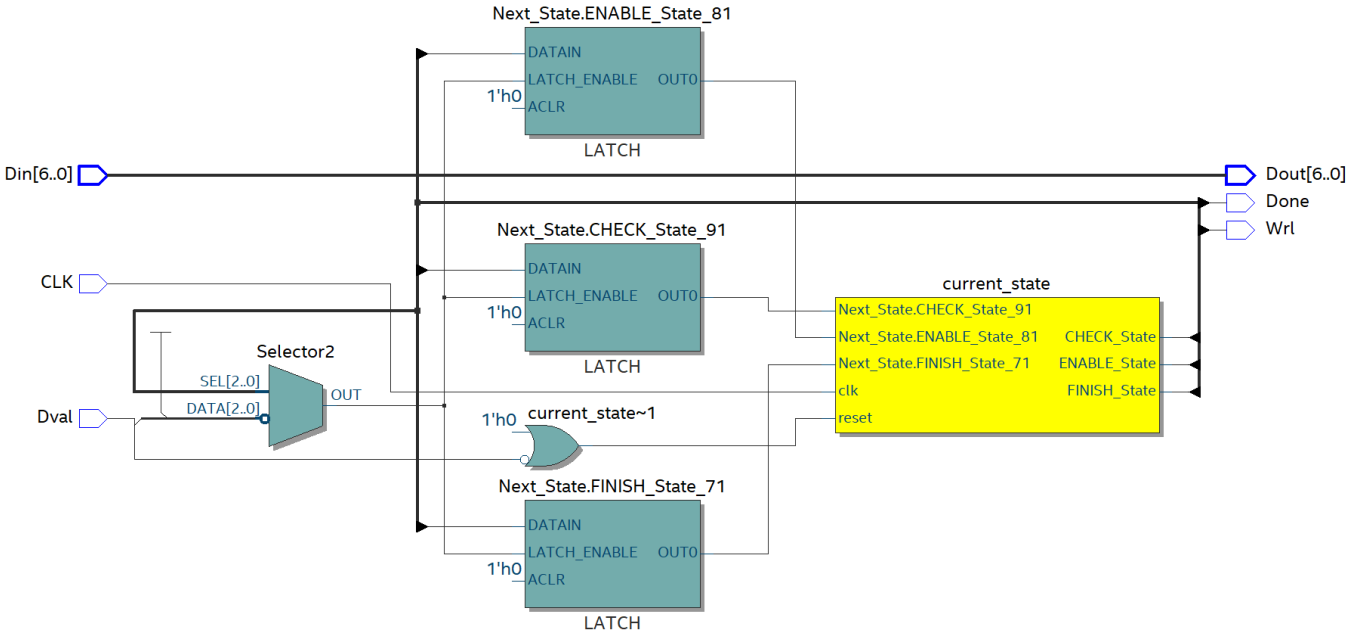

Figura 8: Diagrama de blocos do Score_Dispatcher

```
-- Entidade Score_Dispatcher que define a interface de pontuação no hardware
entity Score_Dispatcher is
    port(
        Din: in std_logic_vector(6 downto 0);  -- Entrada dos dados
        Dval: in std_logic; -- Entrada do bit de validação de dados
        clk: in std_logic; -- Entrada do clock
        WrL: out std_logic; -- Saída do bit de escrita
        done: out std_logic; -- Saída do bit de conclusão
        Dout: out std_logic_vector(6 downto 0) -- Saída de dados
    );
end Score_Dispatcher;

architecture behavioral of Score_Dispatcher is

    -- Definição dos estados possíveis
    type STATE_TYPE is (STATE_WAITING, STATE_ENABLE, STATE_DONE);
    signal CurrentState, NextState : STATE_TYPE;

begin

    -- Transição de estado com o clock
    process(clk)
    begin
        if rising_edge(clk) then
            CurrentState <= NextState;
        end if;
    end process;

    -- Processo de controle de estado
    process (CurrentState, Dval, Din)
    begin
        case CurrentState is
            when STATE_WAITING =>
                if Dval = '1' then
                    NextState <= STATE_ENABLE;
                else
```

```
                NextState <= STATE_WAITING;
            end if;

        when STATE_ENABLE =>
            NextState <= STATE_DONE;

        when STATE_DONE =>
            if Dval = '0' then
                NextState <= STATE_WAITING;
            else
                NextState <= STATE_DONE;
            end if;
    end case;
end process;

-- Atribuições de saída
Dout <= Din;  -- Atribui a entrada de dados para a saída de dados
WrL <= '0' when (CurrentState = STATE_WAITING or CurrentState = STATE_DONE) else '1';  -- Define o sinal de escrita/ativação
done <= '1' when (CurrentState = STATE_DONE) else '0';  -- Define o sinal de conclusão

end behavioral;  -- Fim da arquitetura behavioral
```

Descrição VHDL do bloco *SSC*

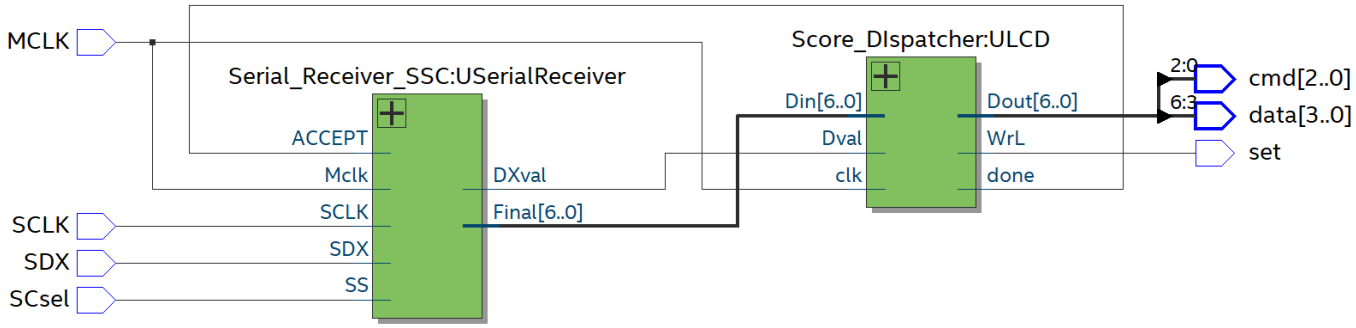

Figura 9: Diagrama de blocos do SSC

```
-- Entidade SSC que define a interface do controlador serial
entity SSC is
    port (
        SDX : in std_logic; -- Entrada do dado em série
        SCLK : in std_logic; -- Entrada do clock principal
        MCLK : in std_logic; -- Entrada do clock secundário
        SCsel : in std_logic; -- Entrada do bit de seleção
        set : out std_logic; -- Saída do bit de ajuste
        cmd : out std_logic_vector(2 downto 0); -- Saída dos comandos
        data : out std_logic_vector(3 downto 0) -- Saída dos dados
    );
end SSC;

architecture SSC_arch of SSC is

    -- Declaração do componente Serial_Receiver_SSC
    component Serial_Receiver_SSC
    port(
        SDX: in std_logic; -- Entrada do dado em série
        SCLK: in std_logic; -- Entrada do clock principal
        Mclk : in std_logic; -- Entrada do clock secundário
        SS: in std_logic; -- Entrada do bit enable
        ACCEPT: in std_logic; -- Entrada do bit de aceitação
        DXval: out std_logic; -- Saída do bit de validação
        Final: out std_logic_vector(6 downto 0) -- Saída do dado final recebido
    );
    end component;
```

```
    -- Declaração do componente Score_Dispatcher
    component Score_Dispatcher
    port(
        Din: in std_logic_vector(6 downto 0); -- Entrada dos dados
        Dval: in std_logic; -- Entrada do bit de validação
        CLK : in std_logic; -- Entrada do clock principal
        WrL: out std_logic; -- Saída do bit que permite a escrita
        done: out std_logic; -- Saída do bit de conclusão
        Dout: out std_logic_vector(6 downto 0) -- Saída dos dados processados
    );
    end component;
    -- Declaração dos sinais internos
    signal Done_s : std_logic;
    signal DXval_s : std_logic;
    signal Final_s : std_logic_vector(6 downto 0);
begin
    -- Instanciação do componente Serial_Receiver_SSC
    USerialReceiver : Serial_Receiver_SSC port map (
        SDX => SDX,
        SCLK => SCLK,
        Mclk => MCLK,
        SS => SCsel,
        ACCEPT => Done_s,
        DXval => DXval_s,
        Final => Final_s
    );
    -- Instanciação do componente Score_Dispatcher
    ULCD : Score_Dispatcher port map (
        Dval => DXval_s,
        CLK => MCLK,
        Din => Final_s,
        Wrl => set,
        Done => Done_s,
        Dout(6 downto 3) => data,
        Dout(2 downto 0) => cmd
    );

end SSC_arch;
```

Atribuição de pinos do módulo *SSC*

O módulo SSC tem interação tanto com o controlo como com o hardware, sendo assim a sua atribuição de pinos na placa "FPGA DE10-Lite" é somente dos sinais:

PIN_C14 -to HEX0[0]
PIN_E15 -to HEX0[1]
PIN_C15 -to HEX0[2]
PIN_C16 -to HEX0[3]
PIN_E16 -to HEX0[4]
PIN_D17 -to HEX0[5]
PIN_C17 -to HEX0[6]
PIN_D15 -to HEX0[7]

PIN_C18 -to HEX1[0]
PIN_D18 -to HEX1[1]
PIN_E18 -to HEX1[2]
PIN_B16 -to HEX1[3]
PIN_A17 -to HEX1[4]
PIN_A18 -to HEX1[5]
PIN_B17 -to HEX1[6]
PIN_A16 -to HEX1[7]

PIN_B20 -to HEX2[0]
PIN_A20 -to HEX2[1]
PIN_B19 -to HEX2[2]
PIN_A21 -to HEX2[3]
PIN_B21 -to HEX2[4]
PIN_C22 -to HEX2[5]
PIN_B22 -to HEX2[6]
PIN_A19 -to HEX2[7]

PIN_F21 -to HEX3[0]
PIN_E22 -to HEX3[1]
PIN_E21 -to HEX3[2]
PIN_C19 -to HEX3[3]
PIN_C20 -to HEX3[4]
PIN_D19 -to HEX3[5]

PIN_E17 -to HEX3[6]

PIN_D22 -to HEX3[7]

PIN_F18 -to HEX4[0]

PIN_E20 -to HEX4[1]

PIN_E19 -to HEX4[2]

PIN_J18 -to HEX4[3]

PIN_H19 -to HEX4[4]

PIN_F19 -to HEX4[5]

PIN_F20 -to HEX4[6]

PIN_F17 -to HEX4[7]

PIN_J20 -to HEX5[0]

PIN_K20 -to HEX5[1]

PIN_L18 -to HEX5[2]

PIN_N18 -to HEX5[3]

PIN_M20 -to HEX5[4]

PIN_N19 -to HEX5[5]

PIN_N20 -to HEX5[6]

PIN_L19 -to HEX5[7]

Código *Kotlin – Score Display*

```
object ScoreDisplay { // Controla o mostrador de pontuação.
    private const val OFF_MASK = 0xF //111 cmd 0001 data = 0001 111
    private const val ON_MASK = 0x7 // 111 cmd 0000 data = 0000 111
    private const val UPDATE_MASK = 0x6 // 110 cmd 0000 data = 0000 110
    private const val EMPTY_MASK = 0X78 // 1111 data 000 cmd depois na função somamos + cmd
    private var lastAnimationTime = System.currentTimeMillis() //Tempo da última animação
    private val animationInterval = 500L // 500 milliseconds
    private var animationCounter = 0 //Qual a última animação
    private val segments = arrayOf(0xA, 0xB, 0xC, 0xD, 0xE, 0x1) //lista das animações
    var isAnimating = true // animação do início é true inicialmente
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init(){
        off(true)//desliga
        off(false)//liga
        for (cmd in 5 downTo 0){
            SerialEmitter.send(SerialEmitter.Destination.SCORE,cmd,8)   //enviar o cmd
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8) //update para saber que
está ligado
    }

    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int){
        var value_temp=value //Para guardar o valor de entrada
        var divider = 100_000 //6 digitos de Score
        var leadingZero = true //0 à esquerda = true porque nunca começamos com número != 0 à esquerda
        for (cmd in 5 downTo 0) {
            var dividido = (value_temp / divider) * divider //Passar para um int e meter esse mesmo int
no resultado
            if (dividido == 0 && leadingZero && divider > 1) {
                SerialEmitter.send(SerialEmitter.Destination.SCORE, EMPTY_MASK + cmd, 8) //Se for 0 à
esquerda, apagar
            } else {
                SerialEmitter.send(SerialEmitter.Destination.SCORE, (value_temp / divider).shl(3) +
cmd, 8) //Escrever o valor
                leadingZero = false //Não é zero à esquerda
            }
            value_temp %= divider //Remover o número de maior grau
            divider /= 10 //Diminuir o grau do divisor
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8)//Para dar update ao score
display
    }

    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean){
        if(value){
            SerialEmitter.send(SerialEmitter.Destination.SCORE, OFF_MASK, 8)//ligar o off

        }
        else SerialEmitter.send(SerialEmitter.Destination.SCORE, ON_MASK, 8)//não ligar o off
    }

    //Começa a animar novamente
    fun startAnimation() {
        isAnimating = true
    }

    //Para de animar
    fun stopAnimation() {
        isAnimating = false
    }
```

```
    // Muda a animação em questão
    fun updateAnimation() {
        if (isAnimating && System.currentTimeMillis() - lastAnimationTime >= animationInterval) {
            lastAnimationTime = System.currentTimeMillis() //Update ao lastAnimationTime
            animationCounter = (animationCounter + 1) % segments.size //O counter de animação = ao
resto de divisão pelo tamanho dos segmentos
            displayAlternatingShapes()
        }
    }

    //AlternatingShapes, bastante explicativo, muda as shapes (da animação inicial)
    private fun displayAlternatingShapes() {
        for (i in 0..5) {
            val cmd = (segments[animationCounter].shl(3)) + i //Shift 3 vezes porque é o cmd primeiro
            SerialEmitter.send(SerialEmitter.Destination.SCORE, cmd, 8) //Para definir qual dos
displays mandamos, daí serem em sincronia
        }
        SerialEmitter.send(SerialEmitter.Destination.SCORE, UPDATE_MASK, 8)//dar update a todos
    }
}
```

Código *Kotlin – Serial Emitter*

```
object SerialEmitter {
    private const val LCD_MASK = 0x01 //É o bit 0 (0001)
    private const val SSC_MASK = 0x02 //É o bit 1 (0010)
    private const val SDX_MASK = 0x08 //É o bit 3 (1000)
    private const val CLK_REG_MASK = 0x10 //É o bit 4 (0001 0000)
    private const val CLEAR_MASK = 0x1F // É os bits todos antes do 32(5) ou seja (0001 1111)

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

    // Inicia a classe
    fun init() {
        HAL.clrBits(CLEAR_MASK)
        HAL.setBits(LCD_MASK)
        HAL.setBits(SSC_MASK)
    }
    //  if LCD size = 10
    //  if Score size = 8

    // 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.
    // Vai primeiro o cmd e depois a data
    fun send(addr: Destination, data: Int, size: Int) {
        var dataShifted = data
        var ParityBit = 0
        var counter = 0

        if (addr == Destination.LCD) {
            HAL.clrBits(LCD_MASK) //Entra negado no hardware
        } else if (addr == Destination.SCORE) {
            HAL.clrBits(SSC_MASK) //Entra negado no hardware
        }


        dataShifted = data.shl(3) //Mover a data 3 vezes pois enviamos pelo bit do SDX
        for (i in 0 until size - 1) {
            HAL.clrBits(CLK_REG_MASK) //Efetuar o clock
            HAL.writeBits(SDX_MASK, dataShifted) //Escrever o valor de dataShifted que está no
valor de SDX_MASK
            if (dataShifted.shr(3) % 2 == 1) {
                counter++
            }
            dataShifted = dataShifted.shr(1) //Mover a data um bit para escrevermos o bit seguinte
            HAL.setBits(CLK_REG_MASK) //Finalizar o clock
        }

        if (counter % 2 == 1) {
            ParityBit = 1 //Se o resto de divisão do counter
        }
        HAL.clrBits(CLK_REG_MASK) //Iniciar o clock
        HAL.writeBits(SDX_MASK, ParityBit.shl(3)) //Escrever o bit de paridade e o shift é para
estar em SDX
        HAL.setBits(CLK_REG_MASK) //Finalizar o clock
        HAL.setBits(LCD_MASK) //Colocar o bit do LCD
        HAL.setBits(SSC_MASK) //Colocar o bit do SSC
```

```
    }
}
```