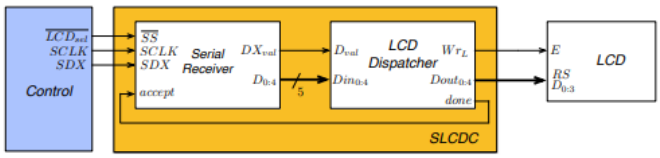
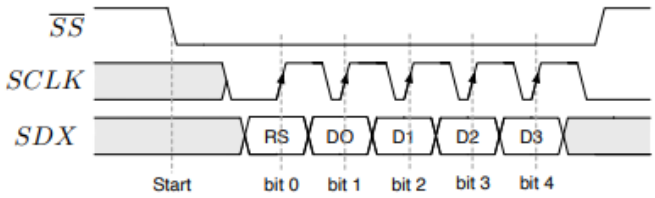
O módulo *Serial LCD Interface* é constituído por dois blocos principais: *i*) o recetor em série da informação (*Serial Receiver*); e *ii*) o despachador de informação para o LCD (designado por *LCD Dispatcher*. Neste caso o módulo *Control*, implementado em *software*, é a entidade fornecedora.



a) Diagrama de blocos



b) Protocolo de comunicação

Figura 1 – *Serial LCD Controller*

1 Serial Receiver

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

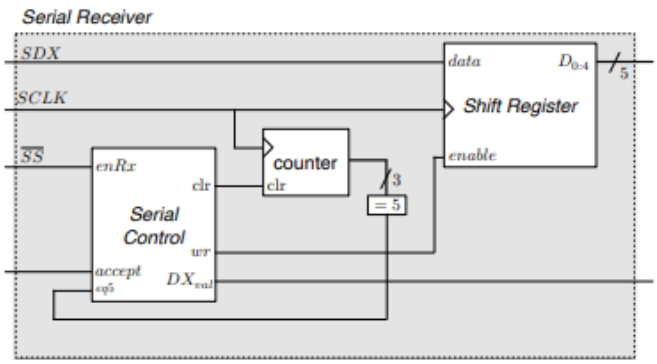


Figura 2 – Diagrama de blocos *Key Decode*

O bloco *Shift Register* foi implementado de acordo com o diagrama de blocos representado na Figura 3. Este bloco permite deslocar os bits de SDX em série para paralelo.

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4. No estado '00' o sistema aguarda o sinal enRx que indica quando vai receber dados, enquanto isso mantém o contador a zeros. Quando passa para o seguinte estado, '01', o sistema dá enable no Shift Register para o mesmo começar a converter os bits de SDX para paralelo até o contador chegar aos 5 clocks. No seguinte estado, '10', o sistema informa o *LCD Dispatcher* que pode ler os bits recebidos, e aguarda o sinal accept que indica que o mesmo já enviou a informação para o *LCD*. O último estado, '11', serve apenas para garantir que o sinal accept volta ao nível lógico '0' antes de receber a próxima trama.

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

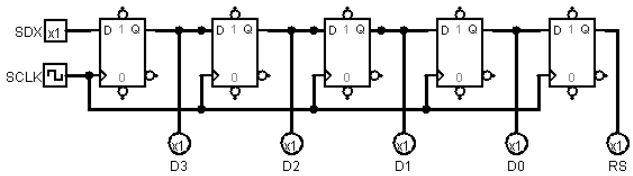


Figura 3 - Diagrama de blocos do bloco *Shift Register.*

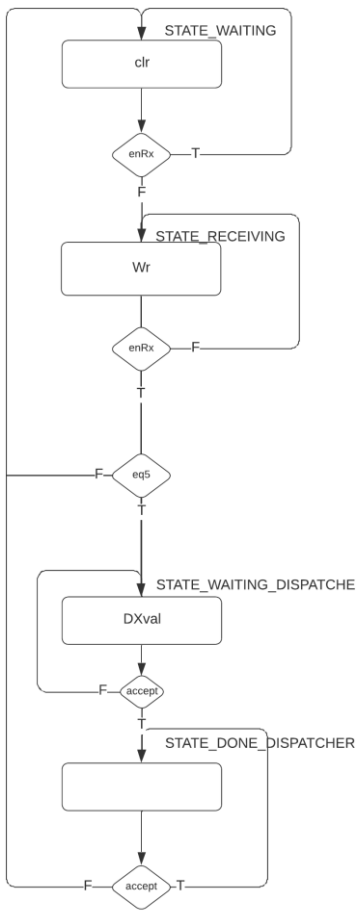


Figura 4 – Máquina de estados do bloco *Serial Control*

2 LCD Dispatcher

O bloco *LCD Dispatcher* entrega a trama recebida pelo *Serial Receiver* ao LCD através da ativação do sinal WrL, após este ter recebido uma trama válida, indicado pela ativação do sinal DXval. O LCD processa as tramas recebidas de acordo com os comandos definidos pelo fabricante, não sendo necessário esperar pela sua execução para libertar o canal de receção série. Assim, o *LCD Dispatcher* pode sinalizar ao *Serial Receiver* que a trama foi processada, ativando o sinal done. Este bloco foi implementado pela máquina de estados representada em *ASM-chart* na Figura 5.

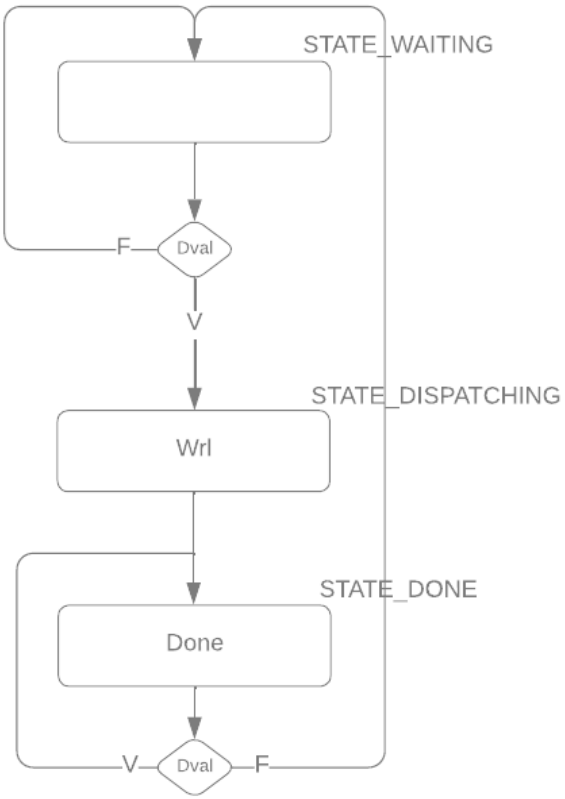


Figura 5 – Máquina de estados do bloco *LCD Dispatcher*

3 Interface com o *Control*

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

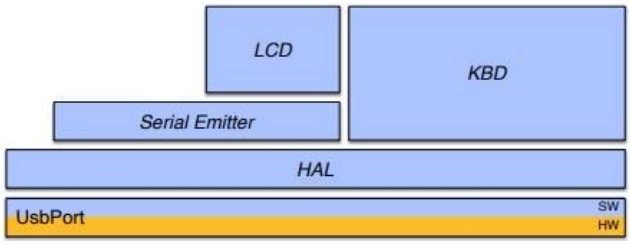


Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *SLCDC.*

3.1 LCD

O bloco LCD escreve no LCD utilizando a interface de 4 bits. Ele pode enviar os bits tanto em paralelo como em série e é responsável pela inicialização do display, escrita de caracteres ou strings no mesmo, e envio de comandos.

3.2 Serial Emitter

O bloco *Serial Emitter* envia tramas para diferentes módulos *Serial Receiver*, o mesmo utiliza o protocolo de comunicação referido na Figura 1b.

LCD e Serial Emitter desenvolvidos são descritos nas secções 3.1 e 3.2, e o código fonte desenvolvido nos Anexos C e D, respetivamente.

4 Conclusões

O módulo *Serial LCD Interface* tem como objetivo enviar informação em série para o LCD através de emissor em *software* e um recetor em *hardware.* Este módulo é um dos principais responsáveis na interação homem-máquina do sistema, pois é ele o responsável pela interface com o utilizador.

A. Descrição VHDL do SLCDC

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;

entity SLCDC is
    port(
    SDX: in std_logic;
   SCLK: in std_logic;
   SS: in std_logic;
   rst : in std_logic;
    clk : in std_logic;

    Wrl: out std_logic;
    Dout: out std_logic_vector(4 downto 0)
    );
end SLCDC;
architecture structural of SLCDC is
component SerialReceiver is
    port
    (
    SDX: in std_logic;
    SCLK: in std_logic;
    SS: in std_logic;
    rst : in std_logic;
    accept: in std_logic;
     clk : in std_logic;

    DXval: out std_logic;
    D: out std_logic_vector(4 downto 0)
    );
    end component SerialReceiver;
component Dispatcher is
    port(
    Dval: in std_logic;
    Din: in std_logic_vector(4 downto 0);
    clk: in std_logic;
    reset: in std_logic;

    Wrl: out std_logic;
    Dout: out std_logic_vector(4 downto 0);
    done: out std_logic
    );
    end component Dispatcher;

signal DXval_signal, done_signal: std_logic;
signal D_signal : std_logic_vector(4 downto 0);
begin
SR1: SerialReceiver port map(SDX => SDX, SS =>SS, SCLK =>SCLK, clk=>clk, rst => rst, accept =>
done_signal,
                                    DXval => DXval_signal, D => D_signal
                                    );
Disp1: Dispatcher port map(Dval => DXval_signal, Din => D_signal, clk=>clk, reset => rst, Wrl =>Wrl,
                                  Dout => Dout, done => done_signal
                                  );
end structural;
```

B. Atribuição de pinos do módulo *SLCDC*

```
set_global_assignment -name FAMILY "MAX 10 FPGA"
set_global_assignment -name DEVICE 10M50DAF484C6GES
set_global_assignment -name TOP_LEVEL_ENTITY "DE10_Lite"
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name SDC_FILE DE10_Lite.sdc
set_global_assignment -name INTERNAL_FLASH_UPDATE_MODE "SINGLE IMAGE WITH ERAM"

#============================================================
# CLOCK
#============================================================
set_location_assignment PIN_P11 -to clk

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to rst
set_location_assignment PIN_C11 -to accept
#============================================================
# LED
#============================================================
set_location_assignment PIN_A8 -to D[0]
set_location_assignment PIN_A9 -to D[1]
set_location_assignment PIN_A10 -to D[2]
set_location_assignment PIN_B10 -to D[3]
set_location_assignment PIN_D13 -to D[4]

#============================================================
# End of pin and io_standard assignments
#============================================================
```

C. Código *Kotlin - LCD*

```
import isel.leic.utils.Time
object LCD { // Escreve no LCD usando a interface a 4 bits.
    enum class WriteType {PARALLEL, SERIAL}
    private var writeType=WriteType.PARALLEL
    private var rsInt = 0
    private const val LINE_VALUE = 0x40
    private const val COL_VALUE = 1
    private const val LCD_D = 0x0F
    private const val LCD_RS = 0x10
    private const val LCD_E = 0x20
    private const val LCD_HIGH = 0xF0
    private const val LCD_LOW = 0x0F
    //Instructions
    private const val RETURN_HOME = 0x02
    private const val DISPLAY_OFF = 0x00
    private const val DISPLAY_ON = 0x0F
    private const val CLEAR_DISPLAY = 0x01
    private const val ENTRY_MODE_SET = 0x06
    private const val SET_DDRAM_ADRESS = 0x80

    // Escreve um nibble de comando/dados no LCD em paralelo
    private fun writeNibbleParallel(rs: Boolean, data: Int){
        if(!rs) HAL.clrBits(LCD_RS)
        else HAL.setBits(LCD_RS)
        Time.sleep(1)
        HAL.writeBits(LCD_D,data)
        HAL.setBits(LCD_E)
        Time.sleep(1)
        HAL.clrBits(LCD_E)
        Time.sleep(1)
    }
    // Escreve um nibble de comando/dados no LCD em série
    fun writeNibbleSerial(rs: Boolean, data: Int){
        rsInt = 0
        if(rs) rsInt=1
        var sdata = data.toString(2)
        sdata += if (rsInt == 0) 0
        else 1
        SerialEmitter.send(SerialEmitter.Destination.LCD,sdata.toInt())
    }
    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int){
        if(writeType==WriteType.PARALLEL) writeNibbleParallel(rs,data)
        else if(writeType==WriteType.SERIAL) writeNibbleSerial(rs,data)
    }
    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int){
        writeNibble(rs,data shr 4)
        writeNibble(rs,data and LCD_LOW)
    }
    // Escreve um comando no LCD
    private fun writeCMD(data: Int){
        writeByte(false,data)
    }
```

```
    // Escreve um dado no LCD
    private fun writeDATA(data: Int){
        writeByte(true,data)
    }
    // Envia a sequência de iniciação para comunicação a 4 bits.
    fun init(){
        Time.sleep(15)
        writeNibble(false,0x03)
        Time.sleep(5)
        writeNibble(false,0x03)
        Time.sleep(1/10)
        writeNibble(false,0x03)
        writeNibble(false,RETURN_HOME)
        writeCMD(0x02)
        writeCMD(0x08)
        writeCMD(DISPLAY_OFF)
        writeCMD(0x08)
        writeCMD(DISPLAY_OFF)
        writeCMD(CLEAR_DISPLAY)
        writeCMD(DISPLAY_OFF)
        writeCMD(ENTRY_MODE_SET)
    }
    // Escreve um caráter na posição corrente.
    fun write(c: Char){
        writeDATA(c.code)
        writeCMD(DISPLAY_ON)
    }

    // Escreve uma string na posição corrente.
    fun write(text: String){
        for(c in text)
            write(c)
    }
    // Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-
1)
    fun cursor(line: Int, column: Int){
        val cursorMove = SET_DDRAM_ADRESS + line*LINE_VALUE + column*COL_VALUE
        writeCMD(cursorMove)
    }
    // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
    fun clear(){
        writeCMD(CLEAR_DISPLAY)
    }
}

fun main(){
    LCD.init()
    LCD.write("ABD")
    LCD.cursor(1,10)
    LCD.write("D")
}
```

D. Código *Kotlin – Serial Emitter*

```
import isel.leic.utils.Time

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

    private const val SDX = 0x01
    private const val SCLK = 0x02
    private const val LCDSELECT = 0x04
    private const val DOORSELECT = 0x08

    // Inicia a classe
    fun init() {
        HAL.setBits(LCDSELECT)
        HAL.setBits(DOORSELECT)
        HAL.setBits(SCLK)
    }

    // Envia uma trama para o SerialReceiver identificado o destino em addr e os
bits de dados em 'data'.
    fun send(addr: Destination, data: Int) {
        var sdx=data
        var ss = 0
        if (addr == Destination.LCD) {
            ss = LCDSELECT
        } else if (addr == Destination.DOOR) {
            ss = DOORSELECT
        }
        HAL.clrBits(ss)
        repeat(5) {
            HAL.clrBits(SCLK)
            Time.sleep(1)
            val lastBit=sdx%10
            HAL.writeBits(SDX, lastBit)
            sdx /= 10
            HAL.setBits(SCLK)
            Time.sleep(1)
        }
        HAL.clrBits(SCLK)
        HAL.setBits(ss)
    }

    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean {
        TODO()
    }
}
fun main(){
    SerialEmitter.init()
    LCD.writeNibbleSerial(true,0x0C) //teste
}
```