, O módulo de interface com o LCD (Serial LCD Interface, SLCDC) implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao LCD, conforme representado na Figura 1.

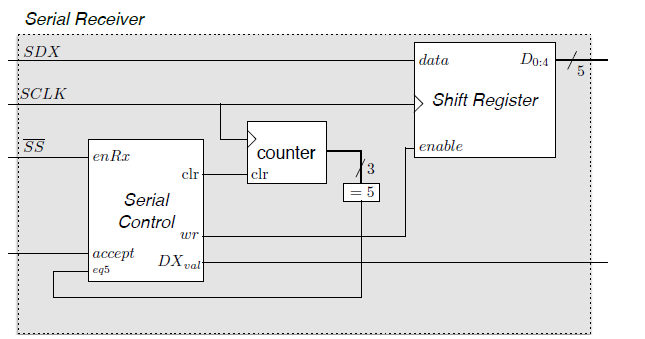
Uma imagem com diagrama

Descrição gerada automaticamente

Figura 1 – Diagrama de blocos do Serial LCD Controller

# SLCDC

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. O Serial Receiver deverá ser implementado com base no diagrama de blocos apresentado na Figura 2.

Figura 2 – Diagrama de blocos do Serial Receiver

Uma imagem com texto, Teclas, fogão, Eletrodoméstico de cozinha

Descrição gerada automaticamente

Figura 3 – Diagrama de temporal do SLCDC

# Interface com o *Control*

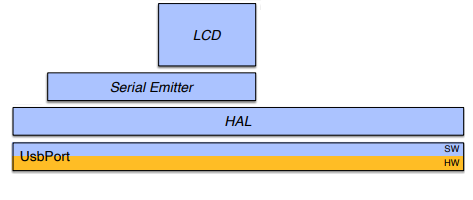
Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 4. 

Figura 4 – Diagrama lógico do módulo *Control* de interface com o módulo *SLCDC.*

*LCD* e *Serial Emitter* desenvolvidos são descritos nas secções 2.1. e 2.2, e o código fonte desenvolvido nos Anexos P e Q, respetivamente.

## *LCD*

Init que tem a seguinte sequência de comandos:

Uma imagem com diagrama

Descrição gerada automaticamente

### fun writeNibbleSerial()

c faz a transformação á data para incluir o valor do rs e chama a função send do serial emitter.

### fun writeNibble()

apenas chama writeNibbleSerial.

fun writeByte()

divide a data de 8 bits em 2 nibbles, 4 bits da parte alta e 4 bits da parte baixa e chama o writeNible com cada uma das partes.

### fun writeCMD()

chama writeByte indicando o rs a falso ooque quer dizer que é um comando de LCD a executar.

### fun writeData()

chama writeByte indicando que está a escrever data no display.

Cursor faz transformação de 2 ints para 1 int com a máscara correspondente à posição desejada.

## *Serial Emitter*

Tem duas funções, o init e send.

No init, é chamado o HAL.init, e é chamado o setBits com a máscara SS, é setBits e não clrBits pois a variável SS é active-low.

No send em 1º lugar ativa o SS com clrBits, depois tem um ciclo no qual faz clrBits do SCLOCK divide a data que recebe bit a bit com clrBits ou setBits dependendo do valor e irá escrever isso em série ao invés de o escrever em paralelo.

Após isto coloca SCLOCK a 1 e faz 5 iterações. Depois do ciclo o clock é colocado a 0 e negamos o SS e fazemos um sleep de 1ms para que a placa consiga receber a informação antes de passar para a próxima trama.

# *Conclusões*

Software envia tramas mais rápido do que LCD consegue receber então colocamos um “sleep” de 1ms para que a placa consiga receber a informação antes de passar para a próxima trama.

2. Máquinas de estado
   1. LCDDispatcher

Uma imagem com diagrama, Esquema, esboço, Desenho técnico

Descrição gerada automaticamente

* 1. SerialControl

Uma imagem com diagrama, Esquema, Desenho técnico, file

Descrição gerada automaticamente

1. Atribuição de pinos do módulo *SLCDC*

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

# inputs

set\_location\_assignment PIN\_C10 -to Reset

#set\_location\_assignment PIN\_C11 -to accept

# Leds

set\_location\_assignment PIN\_A8 -to Dout[0]

set\_location\_assignment PIN\_A9 -to Dout[1]

set\_location\_assignment PIN\_A10 -to Dout[2]

set\_location\_assignment PIN\_B10 -to Dout[3]

set\_location\_assignment PIN\_D13 -to Dout[4]

set\_location\_assignment PIN\_B11 -to Wrl

1. Descrição em VHDL do bloco SLCDC

library ieee;

use ieee.std\_logic\_1164.all;

entity SLCDC is

port(

SS, SCLK, CLK, SDX, Reset : in std\_logic;

Wrl : out std\_logic;

Dout : out std\_logic\_vector(4 downto 0));

end SLCDC;

architecture arc\_slcdc of SLCDC is

component SerialReceiver

port(

SS, SCLK, CLK, SDX, accept, Reset : in std\_logic;

DXval : out std\_logic;

D : out std\_logic\_vector(4 downto 0));

end component;

component LCDDispatcher

port(

Dval, CLK, Reset : in std\_logic;

Din : in std\_logic\_vector(4 downto 0);

Wrl, done : out std\_logic;

Dout : out std\_logic\_vector(4 downto 0));

end component;

signal state, v : std\_logic;

signal d : std\_logic\_vector(4 downto 0);

begin

sr : SerialReceiver port map(

CLK => CLK,

Reset => Reset,

SS => SS,

SCLK => SCLK,

SDX => SDX,

accept => state,

DXval => v,

D => d);

lcdd : LCDDispatcher port map(

Reset => Reset,

CLK => CLK,

Dval => v,

Din => d,

Wrl => Wrl,

done => state,

Dout => Dout);

end arc\_slcdc;

1. Descrição em VHDL do sub-bloco Serial Receiver

library ieee;

use ieee.std\_logic\_1164.all;

entity SerialReceiver is

port(

SS, SCLK, CLK, SDX, accept, Reset : in std\_logic;

DXval, busy : out std\_logic;

D : out std\_logic\_vector(4 downto 0));

end SerialReceiver;

architecture arc\_sr of SerialReceiver is

component SerialControl

port(

Reset, enRx, accept, eq5, CLK : in std\_logic;

clr, wr, DXval, cenable, busy : out std\_logic);

end component;

component ShiftRegister

port(

data, clk, enable, reset : in std\_logic;

D : out std\_logic\_vector(4 downto 0));

end component;

component Counter

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

signal count : std\_logic\_vector(3 downto 0);

signal equal, clear, enwr, cenable : std\_logic;

begin

c : Counter port map(

Reset => clear,

PL => '0',

CE => cenable,

CLK => SCLK,

Data\_in => "0000",

Q => count);

equal <= '1' when (count(2) = '1' and count(1) = '0' and count(0) = '1') else '0';

sc : SerialControl port map(

busy => busy,

Reset => Reset,

CLK => CLK,

enRx => SS,

accept => accept,

eq5 => equal,

clr => clear,

cenable => cenable,

wr => enwr,

DXval => DXval);

sr : ShiftRegister port map(

reset => '0',

data => SDX,

clk => SCLK,

enable => enwr,

D => D);

end arc\_sr;

1. Descrição em VHDL do sub-bloco Shift Register

library ieee;

use ieee.std\_logic\_1164.all;

entity ShiftRegister is

port(

data, clk, enable, reset : in std\_logic;

D : out std\_logic\_vector(4 downto 0));

end ShiftRegister;

architecture arc\_sr of ShiftRegister is

component 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 f1, f2, f3, f4: std\_logic;

begin

ffd4: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => data,

EN => enable,

Q => f4);

ffd3: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f4,

EN => enable,

Q => f3);

ffd2: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f3,

EN => enable,

Q => f2);

ffd1: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f2,

EN => enable,

Q => f1);

ffd0: FFD port map(

SET => '0',

RESET => reset,

CLK => clk,

D => f1,

EN => enable,

Q => D(0));

D(1) <= f1;

D(2) <= f2;

D(3) <= f3;

D(4) <= f4;

end arc\_sr;

1. Descrição em VHDL do sub-bloco Serial Control

library ieee;

use ieee.std\_logic\_1164.all;

entity SerialControl is

port(

Reset, enRx, accept, eq5, CLK : in std\_logic;

clr, wr, DXval, cenable, busy : out std\_logic);

end SerialControl;

architecture arc\_sc of SerialControl is

type STATE\_TYPE is (NOT\_BUSY, COUNT, VALIDATE, STILL\_BUSY);

signal CurrentState, NextState: STATE\_TYPE;

begin

CurrentState <= NOT\_BUSY when Reset = '1' else NextState when rising\_edge(CLK);

GenerateNextState:

process(CurrentState, enRx, eq5, accept)

begin

case CurrentState is

when NOT\_BUSY =>

if (enRx = '0') then NextState <= COUNT;

else NextState <= NOT\_BUSY;

end if;

when COUNT =>

if (enRx = '1' ) then

if(eq5 = '1') then NextState <= VALIDATE;

else NextState <= NOT\_BUSY;

end if;

else NextState <= COUNT;

end if;

when VALIDATE =>

if (accept = '1') then NextState <= STILL\_BUSY;

else NextState <= VALIDATE;

end if;

when STILL\_BUSY =>

if(accept = '0') then NextState <= NOT\_BUSY;

else NextState <= STILL\_BUSY;

end if;

end case;

end process;

clr <= '1' when (CurrentState = NOT\_BUSY) else '0';

cenable <= '1' when (CurrentState = COUNT) else '0';

wr <= '1' when (CurrentState = COUNT) else '0';

DXval <= '1' when (CurrentState = VALIDATE) else '0';

busy <= '0' when (CurrentState = NOT\_BUSY) else '1';

end arc\_sc;

1. Descrição em VHDL do sub-bloco Dispatcher

library ieee;

use ieee.std\_logic\_1164.all;

entity Dispatcher is

port(

Dval, Reset, CLK, eq12 : in std\_logic;

Wrl, done, countclear : out std\_logic);

end Dispatcher;

architecture arc of Dispatcher is

type STATE\_TYPE is (ZEN, COUNT\_WRITE, VALIDATE);

signal CurrentState, NextState: STATE\_TYPE;

signal equal : std\_logic;

begin

CurrentState <= ZEN when Reset = '1' else NextState when rising\_edge(CLK);

equal <= eq12;

GenerateNextState:

process(CurrentState, Dval, equal)

begin

case CurrentState is

when ZEN =>

if (Dval = '1') then NextState <= COUNT\_WRITE;

else NextState <= ZEN;

end if;

when COUNT\_WRITE =>

if (equal = '1') then NextState <= VALIDATE;

else NextState <= COUNT\_WRITE;

end if;

when VALIDATE =>

if (Dval = '0' and equal = '1') then NextState <= ZEN;

else NextState <= VALIDATE;

end if;

end case;

end process;

Wrl <= '1' when (CurrentState = COUNT\_WRITE) else '0';

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

countclear <= '1' when (CurrentState = ZEN) else '0';

end arc;

1. Descrição em VHDL do sub-bloco Terminal Count

library ieee;

use ieee.std\_logic\_1164.all;

entity Terminal\_Count is

port(

Q : in std\_logic\_vector(3 downto 0);

TC: out std\_logic);

end Terminal\_Count;

architecture arc\_tc of Terminal\_Count is

begin

TC <= Q(0) and Q(1) and Q(2) and Q(3);

end arc\_tc;

1. Descrição em VHDL do sub-bloco Registry

library ieee;

use ieee.std\_logic\_1164.all;

entity Registry is

port(

D:in std\_logic\_vector(3 downto 0);

CLK, E, Reset: in std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end Registry;

architecture arc\_reg of Registry is

component 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;

begin

ff0: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(0),

EN => E,

Q => Q(0));

ff1: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(1),

EN => E,

Q => Q(1));

ff2: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(2),

EN => E,

Q => Q(2));

ff3: FFD port map(

SET => '0',

RESET => Reset,

CLK => CLK,

D => D(3),

EN => E,

Q => Q(3));

end arc\_reg;

1. Descrição em VHDL do sub-bloco FFD

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY 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 FFD;

ARCHITECTURE LogicFunction OF FFD IS

BEGIN

Q <= '0' when RESET = '1' else '1' when SET = '1' else D WHEN rising\_edge(clk) and EN = '1';

END LogicFunction;

1. Descrição em VHDL do sub-bloco LCDDispatcher

library ieee;

use ieee.std\_logic\_1164.all;

entity LCDDispatcher is

port(

Dval, Reset, CLK : in std\_logic;

Din : in std\_logic\_vector(4 downto 0);

Wrl, done : out std\_logic;

Dout : out std\_logic\_vector(4 downto 0));

end LCDDispatcher;

architecture arc\_lcdd of LCDDispatcher is

component Dispatcher

port(

Dval, Reset, CLK, eq12 : in std\_logic;

Wrl, done, countclear : out std\_logic);

end component;

component Counter

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

signal eq12, countclear, cenable, wr, d, clean: std\_logic;

signal count: std\_logic\_vector(3 downto 0);

begin

disp: Dispatcher port map(

Dval => Dval,

Reset => Reset,

CLK => CLK,

eq12 => eq12,

Wrl => wr,

done => d,

countclear => countclear);

cup: Counter port map(

Reset => Reset,

PL => clean,

CE => cenable,

CLK => CLK,

Data\_in => "0000",

Q => count);

clean <= eq12 or countclear;

Wrl <= wr;

eq12 <= count(3) and count(2) and not count(1) and not count(0);

done <= d;

Dout <= Din;

cenable <= wr or d;

end arc\_lcdd;

1. Descrição em VHDL do sub-bloco Counter

library ieee;

use ieee.std\_logic\_1164.all;

-- CC -> contador crescente!

entity Counter is

port(

PL, CE, CLK, Reset: in std\_logic;

Data\_in: in std\_logic\_vector(3 downto 0);

TC: out std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end Counter;

architecture arc\_cc of Counter is

component adder

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end component;

component MUX2x1

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end component;

component Registry

port(

D:in std\_logic\_vector(3 downto 0);

CLK, E, Reset: in std\_logic;

Q: out std\_logic\_vector(3 downto 0));

end component;

component Terminal\_Count

port(

Q : in std\_logic\_vector(3 downto 0);

TC: out std\_logic);

end component;

signal outadder, outmux, outreg: std\_logic\_vector(3 downto 0);

begin

ad: adder port map(

A => outreg,

B => "0000",

Ci => CE,

S => outadder);

mux: MUX2x1 port map(

A => Data\_in,

B => outadder,

S => PL,

Y => outmux);

reg: Registry port map(

Reset => Reset,

E => '1',

D => outmux,

CLK => CLK,

Q => outreg);

Q <= outreg;

utc: Terminal\_Count port map(

Q => outreg,

TC => TC);

end arc\_cc;

1. Descrição em VHDL do sub-bloco adder

library ieee;

use ieee.std\_logic\_1164.all;

entity adder is

Port(A, B :in std\_logic\_vector(3 downto 0);

Ci: in std\_logic;

Co: out std\_logic;

S: out std\_logic\_vector(3 downto 0));

end adder;

architecture arc\_adder of adder is

component full\_add

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end component;

signal c1, c2, c3: std\_logic;

begin

fa1: full\_add port map(

A => A(0),

B => B(0),

Cin => Ci,

S => S(0),

Cout => c1);

fa2: full\_add port map(

A => A(1),

B => B(1),

Cin => C1,

S => S(1),

Cout => c2);

fa3: full\_add port map(

A => A(2),

B => B(2),

Cin => C2,

S => S(2),

Cout => c3);

fa4: full\_add port map(

A => A(3),

B => B(3),

Cin => C3,

S => S(3),

Cout => Co);

end arc\_adder;

1. Descrição em VHDL do sub-bloco full add

library ieee;

use ieee.std\_logic\_1164.all;

entity full\_add is

Port(A, B, Cin: in std\_logic;

Cout, S: out std\_logic);

end full\_add;

architecture arc\_fa of full\_add is

begin

S <= A xor B xor Cin;

Cout <= (A and B) or (A and Cin) or (B and Cin);

end arc\_fa;

1. Descrição em VHDL do sub-bloco MUX 2x1

library ieee;

use ieee.std\_logic\_1164.all;

entity MUX2x1 is

port(A, B: in std\_logic\_vector(3 downto 0);

S: in std\_logic;

Y: out std\_logic\_vector(3 downto 0));

end MUX2x1;

architecture arc\_mux of MUX2x1 is

begin

Y(0)<=(S and A(0)) or (not S and B(0));

Y(1)<=(S and A(1)) or (not S and B(1));

Y(2)<=(S and A(2)) or (not S and B(2));

Y(3)<=(S and A(3)) or (not S and B(3));

end arc\_mux;

1. Código *Kotlin* – *LCD*

object LCD {

private const val RS = 0x04

private const val Enable = 0x02

private const val DATA = 0x78

private const val LOW = 0x0F

private const val SLEEPTIME: Long = 1

private const val SLEEPTIME1 = SLEEPTIME \* 40

private const val SLEEPTIME2 = SLEEPTIME \* 5

private const val SLEEPWRITE = 40

private const val SLEEPSET = 230

private const val SLEEPCLR = 270

private const val ON = 1

private const val OFF = 0

private const val SET8BITS = 0x03

private const val SET4BITS = 0x02

private const val NUM\_LINES\_CHAR = 0x28

private const val DISPLAY\_OFF = 0x08

private const val DISPLAY\_CLEAR = 0x01

private const val ENTRY\_MODE = 0x06

private const val DISPLAY\_ON\_CONTROL = 0x0E

private const val SERIAL = true

// Escreve um nibble de comando/dados no LCD em paralelo

private fun writeNibbleParallel(rs: Boolean, data: Int) {

HAL.writeBits(RS, (if (rs) ON else OFF).shl(2))

HAL.writeBits(DATA, data.shl(3))

Thread.sleep(0, SLEEPWRITE)

HAL.setBits(Enable)

Thread.sleep(0, SLEEPSET)

HAL.clrBits(Enable)

Thread.sleep(0, SLEEPCLR)

}

// Escreve um nibble de comando/dados no LCD em série

private fun writeNibbleSerial(rs: Boolean, data: Int) {

val r = if (rs) ON else OFF

SerialEmitter.send(SerialEmitter.Destination.LCD, data.shl(1) + r)

}

// Escreve um nibble de comando/dados no LCD

private fun writeNibble(rs: Boolean, data: Int) {

if (SERIAL) writeNibbleSerial(rs, data) else writeNibbleParallel(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(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() {

SerialEmitter.init()

Thread.sleep(SLEEPTIME1)

writeNibble(false, SET8BITS)

Thread.sleep(SLEEPTIME2)

writeNibble(false, SET8BITS)

Thread.sleep(SLEEPTIME)

writeNibble(false, SET8BITS)

Thread.sleep(SLEEPTIME)

writeNibble(false, SET4BITS)

Thread.sleep(SLEEPTIME)

writeCMD(NUM\_LINES\_CHAR)

Thread.sleep(SLEEPTIME)

writeCMD(DISPLAY\_OFF)

Thread.sleep(SLEEPTIME)

writeCMD(DISPLAY\_CLEAR)

Thread.sleep(SLEEPTIME)

writeCMD(ENTRY\_MODE)

Thread.sleep(SLEEPTIME)

writeCMD(DISPLAY\_ON\_CONTROL)

}

// Escreve um caráter na posição corrente.

fun write(c: Char) {

writeDATA(c.code)

}

// Escreve uma “string” na posição corrente.

fun write(text: String) {

text.forEach { write(it) }

}

// Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)

fun cursor(line: Int, column: Int) {

writeByte(false, (line \* 4 + 8) \* 16 + column)

}

// Envia comando para limpar o ecrã e posicionar o cursor em (0,0)

fun clear() {

writeCMD(DISPLAY\_CLEAR)

}

}

1. Código Kotlin – SeriaEmitter

object SerialEmitter {

enum class Destination { LCD, DOOR }

private const val SLEEPTIME : Long = 1

private const val SSLCD = 0x02

private const val SDX = 0x04

private const val SCLOCK = 0x08

private const val SSDOOR = 0x10

private const val busy = 0x20

private const val MAX\_LENGTH = 5

private const val OFF = 0

// inicia o HAL e coloca os bits relativos ao SS tanto do lcd como da door a 1 (pois são active low)

fun init() {

HAL.init()

HAL.setBits(SSLCD)

HAL.setBits(SSDOOR)

}

// escreve 5 bits de informação, 1 a 1, no usbport

fun send(addr: Destination, data: Int) {

val mask = if (addr == Destination.DOOR) SSDOOR else SSLCD

HAL.clrBits(mask)

for (i in 0 until MAX\_LENGTH) {

HAL.clrBits(SCLOCK)

val b = data.and(1.shl(i))

if (b == OFF) {

HAL.clrBits(SDX)

} else {

HAL.setBits(SDX)

}

HAL.setBits(SCLOCK)

}

HAL.clrBits(SCLOCK)

HAL.setBits(mask)

Thread.sleep(SLEEPTIME)

}

// verifica a condição do sinal busy

fun isBusy() = HAL.isBit(busy)

}