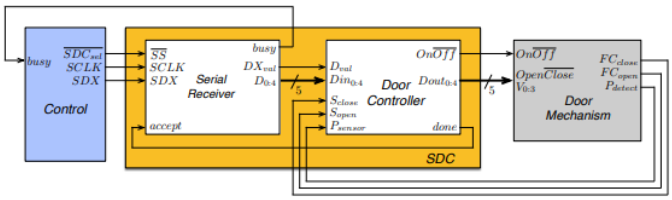
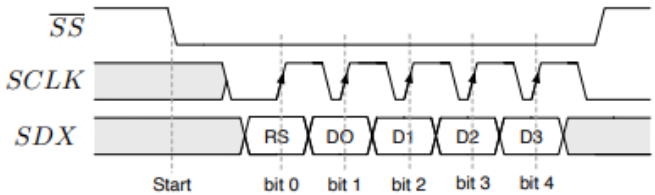
O módulo Serial Door Controller (SDC) é o responsável pela interface com o mecanismo da porta, é constituído por recetor em série e um módulo de controlo, representado na Figura 1a). O SDC recebe em série uma mensagem constituída por cinco bits de informação. A comunicação com o SDC realiza-se segundo o protocolo ilustrado na Figura 1b), tendo como primeiro bit de informação, o bit Open!Close (OC) que indica se o comando é para abrir ou fechar a porta. Os restantes bits contêm a informação da velocidade de abertura ou fecho. O SDC indica que está disponível para a receção de uma nova trama após ter processado a trama anterior, colocando o busy no nível lógico "0".



a) Diagrama de blocos



b) Protocolo de comunicação

Figura 1 – *Serial Door Controller*

1 Serial Receiver

O bloco Serial Receiver do SDC 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.

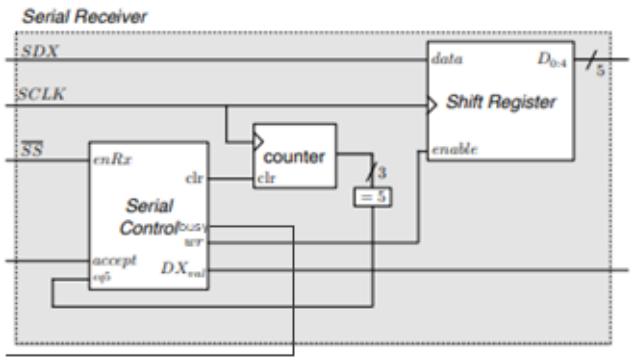


Figura 2 – Diagrama de blocos Serial Receiver

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 dados para paralelo até o contador chegar aos 5 clocks. No seguinte estado, '10', o sistema informa o Door Controller que pode ler os bits recebidos, e aguarda o sinal accept que indica que o mesmo já enviou a informação para o Door Mechanism. O último estado, '11', serve apenas para garantir que o sinal accept volta ao nível lógico '0' antes de receber a próxima trama. O sinal busy é enviado no estado '10' e '11', que informa o software que a trama ainda não foi processada.

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

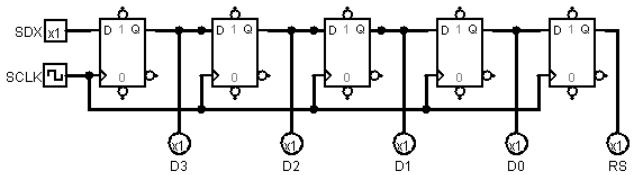


Figura 3 - Diagrama de blocos do bloco *Shift Register*.

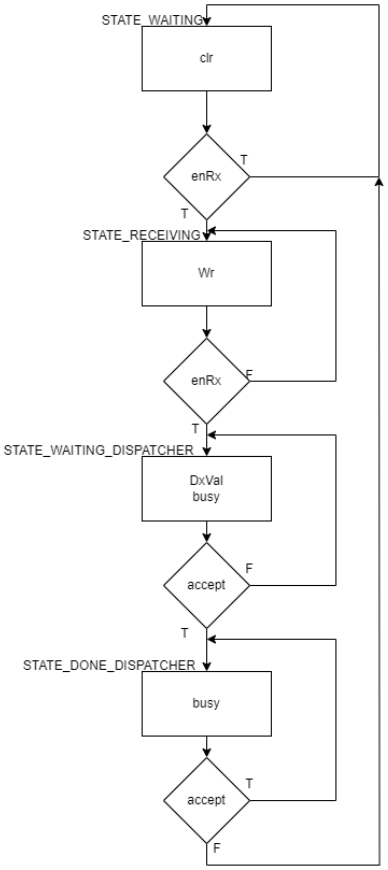


Figura 4 – Máquina de estados do bloco *Serial Control*

2 Door Controller

O bloco Door Controller, após este ter recebido uma trama válida recebida pelo Serial Receiver, procede à atuação do comando recebido no mecanismo da porta. Se o comando recebido for de abertura, o Door Controller coloca o sinal On!Off e o sinal Open!Close no valor lógico '1', até o sensor de porta aberta (FCopen) ficar ativo. No entanto, se o comando for de fecho, o Door Controller ativa o sinal On!Off e colocar o sinal Open!Close no valor lógico '0', até o sensor de porta fechada (FCclose) ficar ativo. Se durante o fecho for detetada uma pessoa na zona da porta, através do sensor de presença (Pdetect), o sistema interrompe o fecho reabrindo a porta. Após a interrupção do fecho da porta, o bloco Door Controller permite forma automática, ou seja, sem necessidade de envio de uma nova trama, o encerramento da porta e o finalizar do comando de fecho. Após concluir qualquer um dos comandos, o Door Controller sinaliza o Serial Receiver que está pronto para processar uma nova trama através da ativação do sinal done.

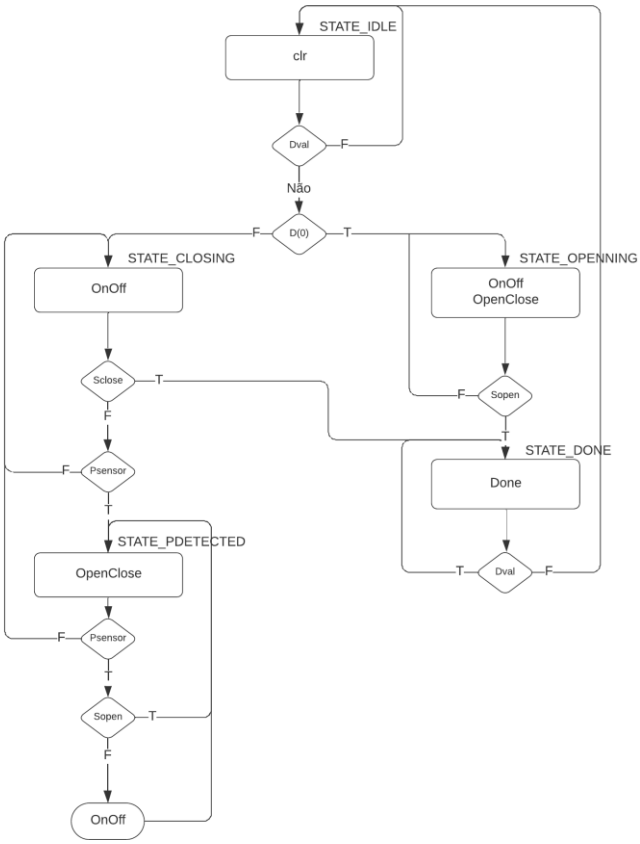


Figura 5 – Máquina de estados do bloco Door Controller

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.

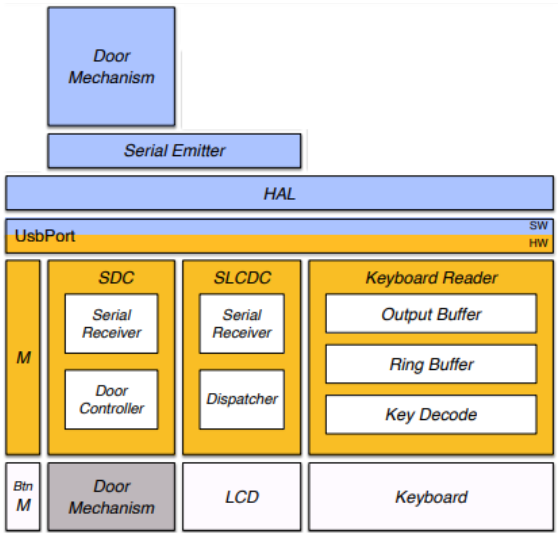


Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *SLCDC.*

3.1 Door Controller

O bloco Door Controller contém as funções de abertura e fecho da porta, alterando o bit de menor peso para selecionar o mecanismo, e os restantes bits para a sua velocidade, e também tem a funcionalidade de interromper o envio de outras tramas enquanto o hardware processa a anterior.

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.

4 Conclusões

O módulo *Serial Door Controller* tem como objetivo enviar informação em série para o Door Mechanism através de emissor em *software* e um recetor em *hardware*. Este módulo é essencial para o correto funcionamento da porta, permitindo controlar a abertura e o fecho da mesma.

A. Descrição VHDL do SDC

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

entity SDC is
        port(
        SDX: in std_logic;
   SCLK: in std_logic;
   SS: in std_logic;
   rst : in std_logic;
        clk : in std_logic;
        Sclose, Sopen, Psensor: in std_logic;

        busy : out std_logic;
        OnOff: out std_logic;
        Dout: out std_logic_vector(4 downto 0)
        );
end SDC;

architecture structural of SDC is
component SerialReceiver_v2 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, busy: out std_logic;
    D: out std_logic_vector(4 downto 0)
    );
end component SerialReceiver_v2;

component DoorController is
port(
                Dval :in std_logic;
                Din : in std_logic_vector(4 downto 0);
                Sclose : in std_logic;
                Sopen : in std_logic;
                Psensor : in std_logic;
                clk : in std_logic;
                rst: in std_logic;

                OnOff : out std_logic;
                Dout : out std_logic_vector(4 downto 0);
                done : out std_logic
                );
end component DoorController;

signal DXval_S, done_s: std_logic;
signal D_S: std_logic_vector(4 downto 0);
begin
```

```
SR1: SerialReceiver_v2 port map(SDX => SDX, SS => SS, SCLK => SCLK, busy => busy,

        accept => done_s, DXval => DXval_S, D => D_S, rst => rst, clk => clk);
DC1: DoorController port map(Dval => DXval_S, Din => D_S, Sclose => Sclose, Sopen
=> Sopen,

        Psensor => Psensor, clk => clk, OnOff => OnOff, Dout => Dout, done =>
done_s, rst => rst);
end structural;
```

B. Código *Kotlin – Door Mechanism*

```
import isel.leic.utils.Time

object DoorMechanism { // Controla o estado do mecanismo de abertura da porta.

    // Inicia a classe, estabelecendo os valores iniciais.
    fun init(){

    }
    // Envia comando para abrir a porta, com o parâmetro de velocidade
    fun open(velocity: Int){
        var data = velocity.toString(2)
        data +=1 //adiciona o bit 1
        SerialEmitter.send(SerialEmitter.Destination.DOOR,data.toInt())
    }
    // Envia comando para fechar a porta, com o parâmetro de velocidade
    fun close(velocity: Int){
        var data = velocity.toString(2)
        data +=0 //adiciona o bit 0
        SerialEmitter.send(SerialEmitter.Destination.DOOR,data.toInt())
    }

    // Verifica se o comando anterior está concluído
    fun finished() : Boolean{
        while(true) {
            if(!SerialEmitter.isBusy())return true
        }
    }
    fun doorMechanism(){

        LCD.cursor(0,0)
        FileAccess.currentUser?.let { TUI.centerMessageDisplay(it.name) }
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Opening...")
        open(2)
        finished()
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Opened")
        Time.sleep(3000)
        close(2)
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Closing...")
        finished()
        LCD.cursor(1,0)
        TUI.centerMessageDisplay("Closed")
        Time.sleep(1000)
    }
}
```

C. 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 = 0x08
    private const val SCLK = 0x10
    private const val LCDSELECT = 0x01
    private const val DOORSELECT = 0x02
    private const val BUSY = 0x40

    // Inicia a classe
    fun init() {
        HAL.setBits(LCDSELECT)
        HAL.setBits(DOORSELECT)
        HAL.clrBits(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) {
            Time.sleep(1)
            val lastBit=sdx%2
            if(lastBit==1) HAL.setBits(SDX)
            else HAL.clrBits(SDX)
            sdx/=2
            HAL.setBits(SCLK)
            Time.sleep(1)
            HAL.clrBits(SCLK)
        }
        HAL.clrBits(SCLK)
        HAL.setBits(ss)
    }

    // Retorna true se o canal série estiver ocupado
    fun isBusy(): Boolean {
        return HAL.isBit(BUSY)
    }
}
```