O módulo de interface com o mecanismo da porta (Serial Door Controller, SDC) implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao mecanismo da porta, conforme representado na Figura 1.

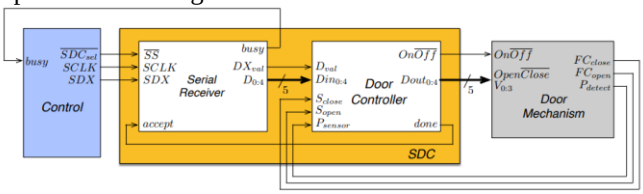

Figura 1- modulo do Serial Door Controller

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 2, tendo como primeiro bit de informação, o bit OC (OpenClose) 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".

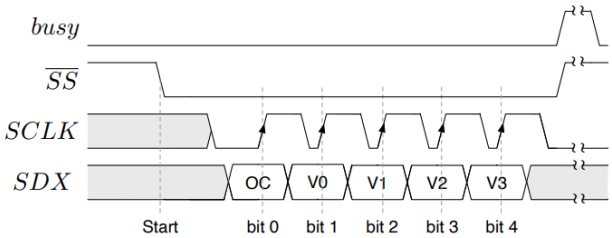

Figura 2 - Protocolo de comunicação do Serial Door Controller

1 Serial Receiver

O bloco *Serial Receiver* tem como função converter o input de em série, num output de 5 bits (em paralelo), de forma que o próprio DoorController (Hardware) consiga receber os dados transmitidos pelo controlo (Software em *kotlin*). Este bloco cumpre a sua funcionalidade da forma que passo a explicar.

O Sinal *SDX* é o bit de transmissão dos dados, o *SCLK* é o *clock* controlado pelo *Software*, o not SS é o sinal de *enable* que diz se o controlo está a transferir dados.

Os Blocos que podemos ver no diagrama da Figura 2 são: um Shift *Register*, que tem a função de ir guardando os bits que vão chegando do sinal SDX; um *counter* de 3 bits, que tem a função de guardar em que que bit da trama recebida é que o circuito vai; um bloco "=5" ou *equals five*, que tem como função fazer output de '1', caso o contador chegue ao valor binário 5 e, finalmente, um Controlo (Serial *Control*), que tem como função, como o próprio nome indica, controlar estes componentes dividindo as "tarefas" em estados.

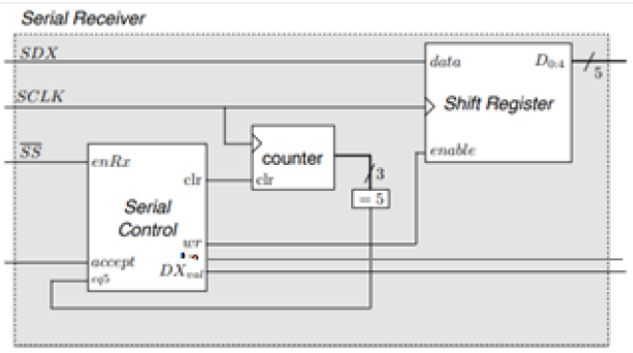

Figura 3 – Módulo do Serial Receiver

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na figura 4.

O Serial *Control* define três estados: o estado *Receiving*, *Waiting* e o *Accepted*. Este começa no estado *Receiving*, em que not SS está a '1' (está desativado, pois é um sinal "*active low*") e, portanto, está à espera de que alguma trama seja enviada. Assim que o sinal not SS é posto a '0' (fica activo), o controlo passa para o estado *Waiting*, em que o controlo liga o *enable* do Shift *Register* e desliga o *clr* do *counter*, permitindo que este conte. Neste estado o Serial *Receiver* está a receber os bits da trama. Quando o contador chega ao valor binário de 5, o módulo *equals five* gera um '1' no output, fazendo o controlo mudar de estado, visto que significa que os 5 bits da trama já foram recebidos. O novo estado é o estado *Accepted*, em que o *enable* do Shift *register* não é ligado, mas o sinal *DXval* é posto a '1'. Como o output do Shift *register* está sempre conectado à saída "K" do bloco em análise, a forma que temos de avisar o próximo componente de que a saída é valida é explicitando-o noutra saída, que é o caso do *DXval*, que tem como função avisar o LCD *Dispatcher* que os 5 bits que está a receber são válidos. Assim que o próximo componente tiver lido o output "K", este envia um sinal ao Serial *Receiver* a dizer que já o fez. Este bit é o sinal *accepted* que, estando o controlo no estado *Accepted*, quando o bit homónimo é posto a '1', o controlo percebe que os dados foram lidos, e passa, de novo, para o primeiro estado, *Receiving*, onde repetirá todo este processo. O Serial Receiver emite o sinal *busy* em todos os estados exceto o *Receiving* que representa o *Serial Receiver* ao meio da receção de uma trama.

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

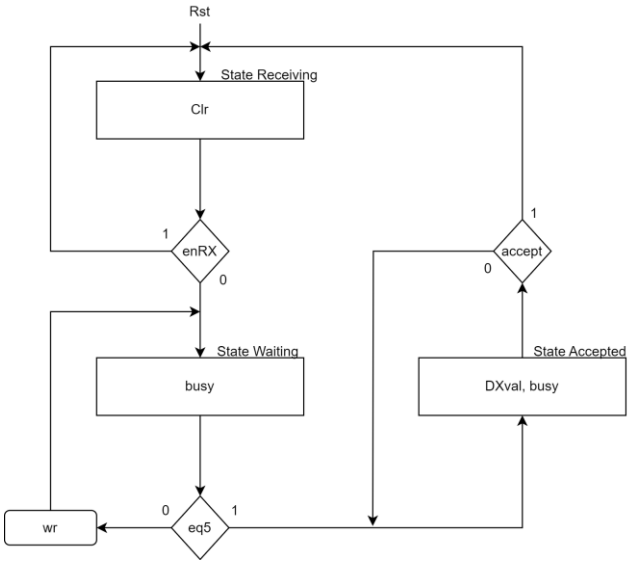

Figura 4 – Máquina de estados do bloco *Serial Control*

2 Door Controller

O Door Controller consiste em uma máquina de estado que recebe o Din do serial Receiver, e quando recebe o sinal Dval realiza a leitura do primeiro bit do Din para então realizar a abertura da porta ou o fecho, caso seja uma abertura ele se manterá nesse estado até receber o sinal Sopen que indica a conclusão, retornando para o estado waiting aonde aguarda novamente o sinal do Dval. Caso seja um pedido de fecho, o Door Controller começa a observar o sinal Psensor para caso algo passe pela porta ela passe a abrir novamente até que esse sinal vá a zero, e quando recebe o sinal Sclose que represente a conclusão do fecho, voltando novamente para o estado waiting.

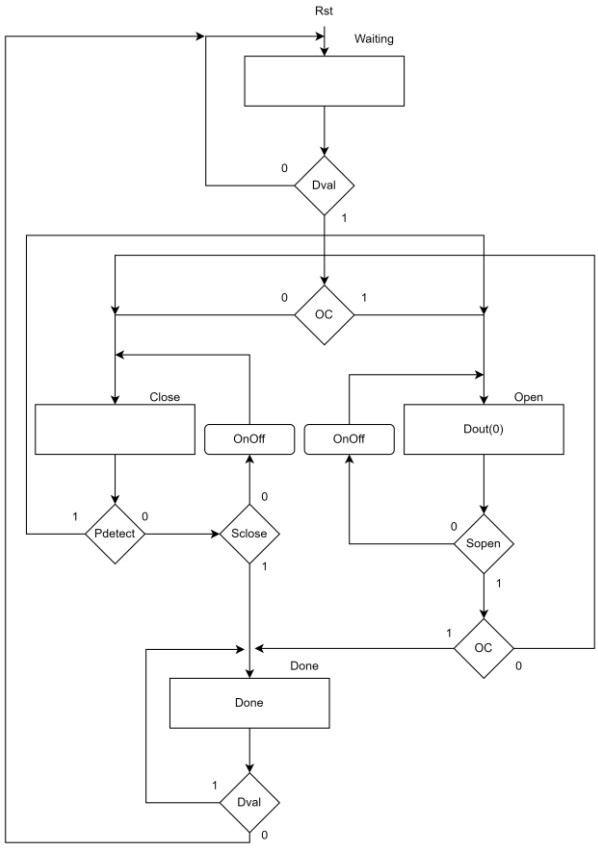

Figura 5 - ASM Chart do Door Controller

3 Conclusões

A implementação do módulo Serial Door Controller (SDC) permite a receção em série de informações enviadas pelo módulo de controlo e a seu posterior entrega ao mecanismo da porta. O SDC recebe uma mensagem de 5 bits, seguindo o protocolo de comunicação ilustrado na Figura 2, em que o primeiro bit (OC) indica se o comando é para abrir ou fechar a porta, e os demais bits contêm informações sobre a velocidade de abertura ou fechamento.

O bloco Serial Receiver tem a função de converter o sinal de entrada serial em um sinal de 5 bits em paralelo para que

o Door Controller (hardware) possa receber os dados transmitidos pelo controlo (software em Kotlin). O bloco Serial Control é responsável por controlar estes componentes, dividindo as tarefas em diferentes estados.
Em relação aos recursos utilizados, esta implementação envolve o uso de um shift register, um contador de 3 bits e circuitos de controlo. A latência no processamento dos comandos é determinada principalmente pela taxa de transmissão do sinal serial e pelo tempo de processamento dos componentes do Serial Receiver e Door Controller.

Em resumo, a implementação do módulo Serial Door Controller proporciona uma comunicação confiável e eficiente entre o módulo de controlo e o mecanismo da porta. O protocolo de comunicação estabelecido permite a transmissão e interpretação corretas dos comandos de abertura e fechamento da porta. A arquitetura baseada em máquina de estados simplifica o controlo e processamento dos comandos, tornando a operação do sistema mais fluida e efetiva.

A. Descrição VHDL do bloco *Serial Receiver*

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

-- Entity

entity serialReceiver is
	port (
		SDX: in std_logic;
		MCLK: in std_logic;
		sCLK: in std_logic;
		SS: in std_logic;
		accept: in std_logic;
		D: out std_logic_vector(4 downto 0);
		DXval: out std_logic;
		Rst: in std_logic;
		busy: out std_logic
	);
end serialReceiver;

-- Architecture

architecture serialReceiver_arch of serialReceiver is

	-- components

	component serialControl
		port (
			clk: in std_logic;
			enRx: in std_logic;
			accept: in std_logic;
			clr: out std_logic;
			wr: out std_logic;
			rst: in std_logic;
			DXval: out std_logic;
			eq5: in std_logic;
			busy: out std_logic
		);
	end component;

	component ContadorUpDown3
		port (
	Clk : in std_logic; -- Clock
	Rst : in std_logic; -- Reset
	Q : out std_logic_vector(2 downto 0) -- output
		);
	end component;

	component shiftReg5
		port(
			Clk: in std_logic;
```

```
                I: in std_logic;
                En: in std_logic;
                rst: in std_logic;
                O: out std_logic_vector(4 downto 0));
        end component;


        -- signals
        signal eq5s2, clrs, wrs, sbusy: std_logic;
        signal eq5s1: std_logic_vector(2 downto 0);

        begin

        serialControl1: serialControl
                port map (
                        clk => MCLK,
                        enRx => SS,
                        accept => accept,
                        clr => clrs,
                        wr => wrs,
                        rst => Rst,
                        DXval => DXval,
                        eq5 => eq5s2,
                        busy => sbusy
                );

        counter1: ContadorUpDown3
                port map (
                        Clk => sClk,
                        Rst => Rst,
                        Q => eq5s1
                );

        shiftReg51: shiftReg5
                port map (
                        Clk => sClk,
                        I => SDX,
                        En => wrs,
                        rst => Rst,
                        O => D
                );

        eq5s2 <= eq5s1(0) and not eq5s1(1) and eq5s1(2);
        busy <= sbusy;


end serialReceiver_arch;
```

B. Descrição VHDL do bloco *Door Controller*

```
library IEEE;
use IEEE.std_logic_1164.all;

entity 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;
      OnOff: out std_logic;
                  clk: in std_logic;
                  rst: in std_logic;
      Dout: out std_logic_vector(4 downto 0);
      done: out std_logic
   );
end DoorController;

architecture behavioral of DoorController is
        type STATE_TYPE is (STATE_OPENING, STATE_WAITING, STATE_CLOSING, STATE_DONE);

        signal CurrentState, NextState: STATE_TYPE;

        begin
--              Flip-Flop's
                CurrentState <= STATE_WAITING when rst = '1' else Nextstate when rising_edge(clk);

--              Generate Next State
                GenerateNextState:
                        process (CurrentState, Din(4), Psensor, Sclose, Dval, Sopen)
                                begin
                                        case CurrentState is
                                                when STATE_WAITING => if (Dval = '1' and Din(4) = '1') then

                                                                        NextState <= STATE_OPENING;

                                                                elsif (Dval = '1' and Din(4) = '0') then

                                                                        NextState <= STATE_CLOSING;

                                                                else

                                                                        NextState <= STATE_WAITING;

                                                                end if;
                                                when STATE_OPENING => if (Sopen = '1' and Din(4) = '1') then

                                                                        NextState <= STATE_DONE;

                                                                elsif (Sopen = '1' and Din(4) = '0') then
```

```
                                        NextState <= STATE_CLOSING;
                                    else
                                        NextState <= STATE_OPENING;
                                    end if;
                                when STATE_CLOSING  => if (Psensor='1') then
                                        NextState <= STATE_OPENING;
                                    elsif (Sclose='1' and Psensor='0') then
                                        NextState <= STATE_DONE;
                                    else
                                        NextState <= STATE_CLOSING;
                                    end if;
                                when STATE_DONE => if (Dval = '0') then
                                        NextState <= STATE_WAITING;
                                    else
                                        NextState <= STATE_DONE;
                                    end if;
                            end case;
                    end process;

--                  Generate Outputs
                    OnOff<= '1' when ((CurrentState = STATE_OPENING and Sopen = '0') or (CurrentState = STATE_CLOSING and Psensor = '0' and Sclose = '0')) else '0';
                    done <= '1' when (CurrentState = STATE_DONE) else '0';
                    Dout(3 downto 0) <= Din(3 downto 0);
                    Dout(4) <= '1' when ( CurrentState = STATE_OPENING) else '0';
end architecture;
```

C. Atribuição de pinos do módulo SDC

```
#============================================================
# Altera DE10-Lite board settings
#============================================================


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 MCLK

#============================================================
# SW
#============================================================

set_location_assignment PIN_B14 -to Pswitch
set_location_assignment PIN_F15 -to rst

#============================================================
# LED
#============================================================
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]

#============================================================
# HEX0
#============================================================

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

#============================================================
# HEX1
#============================================================

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

```
set_location_assignment PIN_A16 -to HEX1[7]

#============================================================
# HEX2
#============================================================

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

#============================================================
# HEX3
#============================================================

set_location_assignment PIN_F21 -to HEX3[0]
set_location_assignment PIN_E22 -to HEX3[1]
set_location_assignment PIN_E21 -to HEX3[2]
set_location_assignment PIN_C19 -to HEX3[3]
set_location_assignment PIN_C20 -to HEX3[4]
set_location_assignment PIN_D19 -to HEX3[5]
set_location_assignment PIN_E17 -to HEX3[6]
set_location_assignment PIN_D22 -to HEX3[7]

#============================================================
# HEX4
#============================================================

set_location_assignment PIN_F18 -to HEX4[0]
set_location_assignment PIN_E20 -to HEX4[1]
set_location_assignment PIN_E19 -to HEX4[2]
set_location_assignment PIN_J18 -to HEX4[3]
set_location_assignment PIN_H19 -to HEX4[4]
set_location_assignment PIN_F19 -to HEX4[5]
set_location_assignment PIN_F20 -to HEX4[6]
set_location_assignment PIN_F17 -to HEX4[7]

#============================================================
# HEX5
#============================================================

set_location_assignment PIN_J20 -to HEX5[0]
set_location_assignment PIN_K20 -to HEX5[1]
set_location_assignment PIN_L18 -to HEX5[2]
set_location_assignment PIN_N18 -to HEX5[3]
set_location_assignment PIN_M20 -to HEX5[4]
set_location_assignment PIN_N19 -to HEX5[5]
set_location_assignment PIN_N20 -to HEX5[6]
set_location_assignment PIN_L19 -to HEX5[7]
```

D. Código Kotlin do Serial Emitter

```
const val SDC_ENABLE = 0x02
const val LCD_ENABLE = 0x01
const val SERIAL_CLK = 0x10
const val SIZE = 0x05
const val SERIAL_NIBBLE = 0x08
const val BUSY = 0x40
const val SERIAL_NIBBLE_MASK = 0x01
const val SERIAL_NIBBLE_SHIFT = 0x03
enum class Destination { LCD, DOOR }

object SerialEmitter {
    fun init(){
        HAL.setBits(SDC_ENABLE or LCD_ENABLE)
        HAL.clrBits(SERIAL_CLK)
    }
    fun send(addr: Destination, data: Int){
        when(addr){
            Destination.LCD -> HAL.clrBits(LCD_ENABLE)
            Destination.DOOR -> HAL.clrBits(SDC_ENABLE)
        }
        repeat(SIZE){
            val msg = ((data shr(it)) and SERIAL_NIBBLE_MASK) shl
(SERIAL_NIBBLE_SHIFT)
            HAL.writeBits(SERIAL_NIBBLE, msg)
            HAL.setBits(SERIAL_CLK)
            HAL.clrBits(SERIAL_CLK)
        }
        when(addr) {
            Destination.LCD -> HAL.setBits(LCD_ENABLE)
            Destination.DOOR -> HAL.setBits(SDC_ENABLE)
        }
    }
    fun isBusy(): Boolean = HAL.readBits(BUSY) != 0
}
```

E. Código Kotlin do Door Mechanism

```
const val OPEN = 0x1

object DoorMechanism {
    fun init(){}
    fun open(velocity: Int){
        val msg = velocity shl (1) or OPEN
        SerialEmitter.send(Destination.DOOR, msg)
    }
    fun close(velocity: Int){
        val msg = velocity shl (1)
        SerialEmitter.send(Destination.DOOR, msg)
    }
    fun finished(): Boolean = !SerialEmitter.isBusy()
}
```