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.

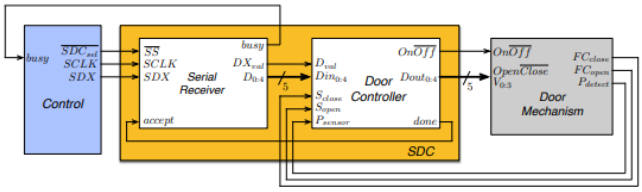

Figura 1 – Diagrama de blocos do *Serial Door Controller*

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 $OpenClose$ (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".

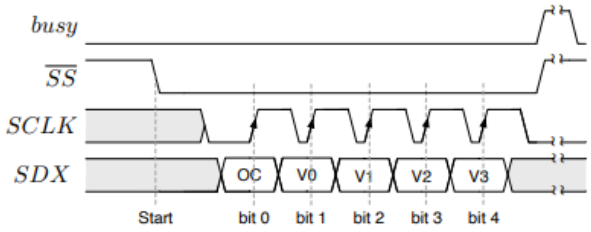

Figura 2 – Protocolo de comunicação do *Serial Door Controller*

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

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*. O *Serial Receiver* foi implementado com base no diagrama de blocos apresentado na Figura 3.

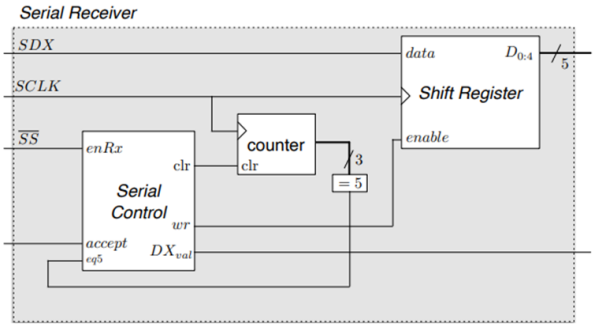

Figura 3 – Diagrama de blocos do Serial Receiver

O bloco *Shift Register* foi implementado de acordo com o diagrama de blocos representado na Figura 4.

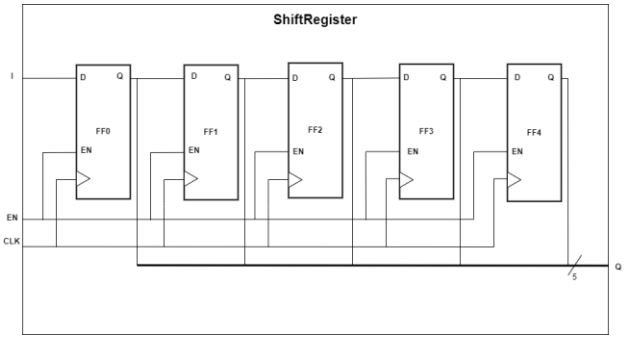

Figura 4 – Diagrama de blocos do Shift Register

Foram utilizados 5 *Flip-Flops* ligados em série. A entrada do primeiro *Flip-Flop* é a entrada *I* do circuito, e os restantes *Flip-Flop* têm como a entrada a saída do *Flip-Flop* anterior. As saídas dos cinco *Flip-Flops* representam o sinal de dados de cinco bits.

O bloco *Serial Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 5.
A descrição hardware do bloco *Serial Receiver* em *VHDL* encontra-se no Anexo A.

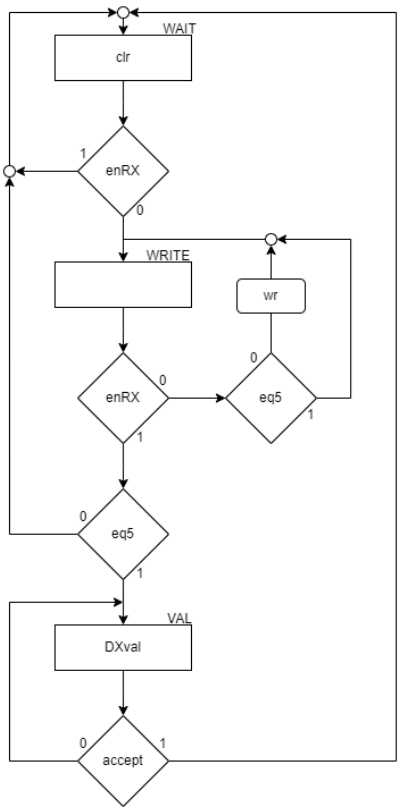

Figura 5 – Máquina de estados do bloco *Serial Control*

Inicialmente a máquina encontra-se no estado **WAIT**, em que se limpa o valor do contador e se espera para que o *LCD* seja selecionado. Quando é selecionado o serial receiver do *LCD* através de *enRX*, a máquina passa para o estado **WRITE** em que ativa o *Shift Register* através do sinal *wr* enquanto não são enviados os 5 bits de informação (Figura 2). Assim que são enviados os cinco bits de informação, a máquina desativa o sinal *wr* e espera para que o *LCD* deixe de ser selecionado através de *enRX* para sinalizar que tem dados a enviar ao *LCD Dispatcher*. Se forem enviados bits de informação insuficientes ou a mais, então a máquina de estados invalida a trama quando o *LCD* deixa de ser selecionado e volta para o estado **WAIT**. Caso seja enviada uma trama válida a máquina passa para o estado **VAL**. Neste estado, envia o sinal *DXval* para o *LCD Dispatcher*, para indicar que tem dados a enviar. Quando o sinal *accept* é enviado pelo *LCD Dispatcher* a indicar que a trama foi aceite, pode ser iniciado um novo ciclo de escrita e a máquina volta para o estado **WAIT**.

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 $On\overline{Off}$ e o sinal $Open\overline{Close}$ 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 $On\overline{Off}$ e colocar o sinal $Open\overline{Close}$ 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 de 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 $done$.

A descrição hardware do bloco *Door Controller* em *VHDL* encontra-se no Anexo B.

O bloco *Door Controller* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 6.

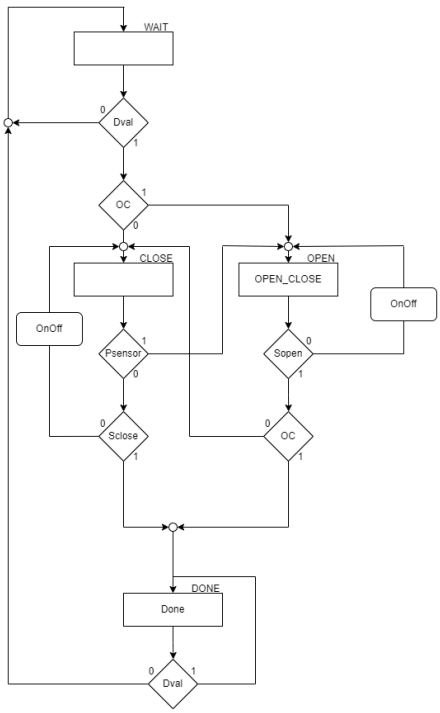

Figura 6 – Máquina de estados do bloco *Door Controller*

Inicialmente, a máquina de estados encontra-se no estado **CLOSE**, isto porque quando o *Access Control System* é inicializado, a porta tem de ser fechada caso esteja aberta, para não permitir acesso a pessoas não autorizadas. Depois de ser fechada a porta na inicialização do sistema, a máquina de estados volta para o estado de espera **WAIT**.

Neste estado, espera para receber um comando de abertura ou fecho através do *Serial Receiver*. Assim que o *Serial Receiver* sinaliza que tem uma trama para ser processada, o *Door Controller* passa a processá-la. Se o bit de OC da trama que indica se o comando é de fecho ou abertura estiver a 0, então passa para o estado **CLOSE**, se estiver a 1 passa para o estado **OPEN**.

No estado **CLOSE**, a máquina envia o sinal $On\overline{Off}$ para o mecanismo da porta enquanto a porta não estiver fechada com $Open\overline{Close}$ a 0. Se houver uma deteção de interrupção, então a máquina vai para o estado de **OPEN** para re-abrir a porta. Caso não exista interrupção, a máquina espera para que o mecanismo da porta termine o fecho da porta, e passa para o estado **DONE**.

Quando existe uma interrupção e se passa para o estado OPEN, é enviado para o mecanismo da porta o sinal $On\overline{Off}$ enquanto a porta não está aberta com $Open\overline{Close}$ a 1. Quando a porta é aberta, a máquina verifica se o bit de OC era 0 ou 1 inicialmente, se era 0, então o comando inicial era de fecho, e a máquina volta para o estado **CLOSE**. Se era 1 então o comando inicial eraa de abertura e a máquina transiciona para o estado **DONE**. Caso OC seja 0 e volte para **CLOSE**, se deixar de haver interrupção termina o processo de fechar a porta. Se continuar a haver interrupção, então a máquina transiciona entre os estados **OPEN** e **CLOSE** com $On\overline{Off}$ com o valor lógico '0' até deixar de haver interrupção.

Quando é completado um comando de fecho ou abertura, a máquina passa para o estado **DONE**, onde sinaliza ao *Serial Receiver* que terminou o comando. Depois de o *Serial Receiver* baixar o sinal $Dval$, a máquina volta para o estado **WAIT**, onde espera por um novo comando.

3 Interface com o Control

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 8.

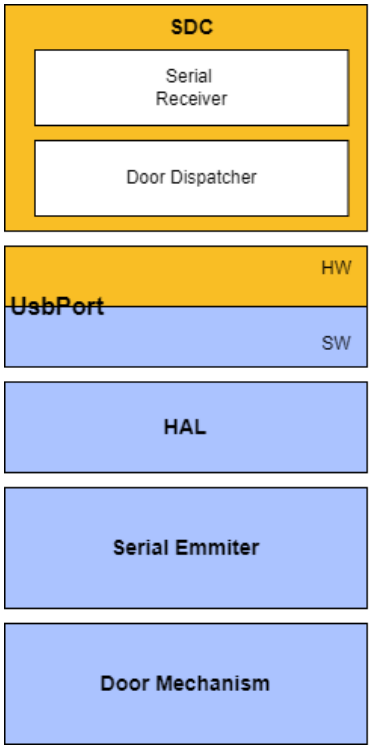

Figura 8 – Diagrama lógico do módulo *Control* de interface com o módulo *SLCDC*

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

3.1 Door Mechanism

A classe *Door Mechanism* é responsável por controlar o mecanismo da porta do sistema, através de envio em modo serial de comandos para o controlador em hardware.

Foram adicionadas as constantes `OPENCMD`, `CLOSECMD`.

As constantes `OPENCMD` e `CLOSECMD` representam o comando de abertura e fecho da porta, respetivamente.

A função `init()` fecha a porta. Neste sistema, o acesso deve ser permitido apenas a pessoas autorizadas, o que significa que o sistema deve ter a porta fechada inicialmente.

A função `open(velocity: Int)` envia um comando de abertura da porta através do envio para o *Serial Emitter* do *Door Control* uma trama de cinco bits com o bit *OC* a 1. Para enviar o comando, é feito um or entre a constante `OPENCMD` e o parâmetro `velocity`, com o destino do *Serial Emitter* da porta.

A função `close(velocity: Int)` envia um comando de fecho da porta através do envio para o *Serial Emitter* do *Door Control* uma trama com o bit *OC* a 0. Para enviar o comando, é feito um or entre a constante `CLOSECMD` e o parâmetro `velocity`, com o destino do *Serial Emitter* do *Door Control*.

A função `finished(): Boolean` Indica se o mecanismo da porta já terminou ou não a operação de fechar/abrir ao verificar se o *Serial Emitter* do *Door Control* está *busy*. Se estiver *busy*, retorna false, se não estiver busy, retorna true.

3.2 Serial Emitter

O *Serial Emitter* é responsável por envio de dados para o *LCD* ou o controlador da porta em modo serial.

Foram adicionadas as constantes `DELAY`, `SDXMASK`, `SCLKMASK` e `BUSYMASK`.

As constantes `MASK` representam as posições dos respetivos sinais na *USB Port*, e a constante `DELAY` representa o tempo de atraso entre pulsos de `SCLK`.
Também foi utilizada a função global adicional `waitTimeNano`, que espera um tempo definido em nanossegundos.

Foi adicionada a função `clkPulse()`, que realiza um pulso de `SCLK` com tempos de espera `DELAY`(100 nanosegundos) ao chamar `HAL.setBits()`, e `HAL.clrBits()`.

A função `init()` inicia a classe colocando o sinal `SCLK` a 0, e desselecionando os *Serial Receivers* da porta e do *LCD*.

A função `send(addr: Destination, data: Int)` envia os dados serialmente. Inicialmente seleciona um dos *Serial Controllers* através de `HAL.writeBits()`, e espera um tempo `DELAY`(100 nanosegundos) para o hardware preparar a escrita. Após o tempo `DELAY` são escritos os bits da trama de dados um a um através de `HAL.writeBits()` de acordo com a Figura 2, sendo que para cada um deles é chamada a função `clkPulse()`.

A função `isbusy()` verifica se o canal série está ocupado, utilizando a função `HAL.isBit()` para verificar o mesmo.

4 Conclusões

Neste módulo foi implementado o módulo *Serial Door Controller* através dos blocos *Serial Receiver* e *Door Controller*, que implementa a receção em série da informação enviada pelo módulo de controlo, entregando-a posteriormente ao mecanismo da porta.

Em termos de recursos da placa DE-10 Lite, são utilizados 22/49,760 elementos lógicos (0,04%), 12 registos, e 15/360 pinos (4%).

A. Descrição VHDL do bloco *Serial Receiver*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SerialReceiver is
PORT( busy: out std_logic;
            SDX : in STD_LOGIC;
            SCLK : in STD_LOGIC;
            SS : in std_logic;
            accept : IN STD_LOGIC;
            clk: IN std_logic;
            D : out std_logic_vector(4 downto 0);
            DXval : out STD_LOGIC;
            RST: in std_logic
            );
end SerialReceiver;

ARCHITECTURE logic OF SerialReceiver is



component SerialControl is
      port (
            RST : in std_logic;
            clk : in std_logic;
            enRX : in std_logic;
            accept : in std_logic;
            eq5 : in std_logic;
            clr : out std_logic;
            wr : out std_logic;
            DXval : out std_logic
      );
end component;



component counterUp3 is
PORT(
            RST : in STD_LOGIC;
            CE : in std_logic;
            CLK : IN STD_LOGIC;
            Q : out std_logic_vector(2 downto 0)
            );
end component;

component Shiftregister is
PORT( I : in std_logic;
            EN : in STD_LOGIC;
            RST : in std_logic;
            CLK : IN STD_LOGIC;
            DATA : out std_logic_vector(4 downto 0)
            );
end component;

signal sclr, swr, seq5, sval: std_logic;
signal scounter : std_logic_vector ( 2 downto 0);
```

```
Begin

Control : SerialControl port map(
      RST => RST,
      clk => clk,
      enRX => SS,
      accept => accept,
      eq5 => seq5,
      clr => sclr,
      wr => swr,
      DXval => sval
);

Counter: counterUp3 port map(
      RST => sclr,
      CE => '1',
      CLK => SCLK,
      Q => scounter
);


SRegister : Shiftregister port map(
      I => SDX,
      EN => swr,
      RST => RST,
      CLK => SCLK,
      DATA => D
);

busy <= sval;
DXval <= sval;
seq5 <= scounter(0) and not scounter(1) and scounter(2) and not sclk;
end logic;
```

B. Descrição VHDL do bloco *Door Controller*

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

entity DoorController is
	port (
		clk    : in std_logic;
		RST    : in std_logic;
		Din    : in std_logic_vector(4 downto 0);
		Dval   : in std_logic;
		Sclose: in std_logic;
		Sopen  : in std_logic;
		Psensor: in std_logic;
		OnOff  : out std_logic;
		Done   : out std_logic;
		Dout   : out std_logic_vector(4 downto 0)
	);
end DoorController;

architecture logic of DoorController is
type STATE_TYPE is (STATE_CLOSE, STATE_OPEN, STATE_WAIT, STATE_DONE);

signal currentstate, nextstate: STATE_TYPE;
signal OC, OPEN_CLOSE :std_logic;
begin
	OC <= Din(0);
	currentstate <= STATE_CLOSE when RST = '1' else nextstate when rising_edge(clk);

	GenerateNextState:
	process(currentstate, Sopen, Sclose, Psensor, Dval, oc)
		Begin

		case currentstate is

			when STATE_WAIT    =>  if(Dval = '1' and OC ='1') then
						       nextstate <= STATE_OPEN;
						       elsif(Dval = '1' and OC ='0') then
						       nextstate <= STATE_CLOSE;
						       else
						       nextstate <= STATE_WAIT;
						       end if;

			WHEN STATE_OPEN  =>    if(Sopen ='1' and OC = '1') then
						       nextstate <= STATE_DONE;
						       elsif ( Sopen ='1' and OC = '0') then
						       nextstate <= STATE_CLOSE;
						       else
						       nextstate <= STATE_OPEN;
						       end if;

			WHEN STATE_CLOSE => if(Psensor ='1') then
						       nextstate <= STATE_OPEN;
						       elsif (Sclose ='1') then
						       nextstate <= STATE_DONE
						       else
						       nextstate <= STATE_CLOSE;
						       end if;

			WHEN STATE_DONE =>         if(Dval = '1') then
						       nextstate <= STATE_DONE;
						       else
						       nextstate <= STATE_WAIT;
						       end if;

		end case;
	end process;
```

```
	OnOff <= '1' when ((currentstate = STATE_OPEN and Sopen = '0') or (currentstate =
STATE_CLOSE and Sclose = '0' and Psensor = '0')) else '0';

	OPEN_CLOSE <= '1' when currentstate = STATE_OPEN else '0';

	Dout <= Din(4) & Din(3) & Din(2) & Din(1) & OPEN_CLOSE;

	Done <= '1' when currentstate = STATE_DONE else '0';


end logic;
```

C. Atribuição de pinos do bloco *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_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK2_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_ADC_10
set_location_assignment PIN_P11 -to clk
#set_location_assignment PIN_N14 -to CLOCK2_50
#set_location_assignment PIN_N5 -to CLOCK_ADC_10

#============================================================
# SW
#============================================================
set_location_assignment PIN_C10 -to SCLK
set_location_assignment PIN_C11 -to SDX
set_location_assignment PIN_D12 -to FCopen
set_location_assignment PIN_C12 -to Pdetect
set_location_assignment PIN_A12 -to FCclose
#set_location_assignment PIN_B12 -to SW[5]
#set_location_assignment PIN_A13 -to SW[6]
#set_location_assignment PIN_A14 -to SW[7]
#set_location_assignment PIN_B14 -to SW[8]
set_location_assignment PIN_F15 -to RST

#============================================================
# LED
#============================================================
set_location_assignment PIN_A8 -to OnOff
set_location_assignment PIN_A9 -to busy
#set_location_assignment PIN_A10 -to LEDR[2]
#set_location_assignment PIN_B10 -to LEDR[3]
#set_location_assignment PIN_D13 -to LEDR[4]
set_location_assignment PIN_C13 -to Dout[0]
set_location_assignment PIN_E14 -to Dout[1]
set_location_assignment PIN_D14 -to Dout[2]
set_location_assignment PIN_A11 -to Dout[3]
set_location_assignment PIN_B11 -to Dout[4]
```

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

D. Código Kotlin *Door Mechanism*

```
import SerialEmitter.Destination

fun main() {

    HAL.init()
    SerialEmitter.init()
    LCD.init()

    DoorMechanism.open(1)
    TUI.writeLine("OPENING", TUI.ALIGN.Center, TUI.LINES.First, true)

    while (!DoorMechanism.finished());

    LCD.clear()

    LCD.write("CLOSING...")
    DoorMechanism.close(1)

    while (!DoorMechanism.finished());
}


/**
 * 22/5/2023
 *
 *
 * Responsible for controlling the state of the door mechanism.
 * @author Bernardo Pereira
 * @author AntÃ³nio Paulino
 * @see SerialEmitter
 */

object DoorMechanism {

    /**
     * Represents the open command for the door mechanism.
     */
    private const val OPENCMD_FRAME = 0b00001

    /**
     * Represents the close command for the door mechanism.
     */
    private const val CLOSECMD_FRAME = 0b00000


    /**
     * Initializes the door mechanism by closing the door.
     */
    fun init() {
        close(AccessControlSystem.DOOR_CLOSE_SPEED)
    }


    /**
     * Opens the door with the specified velocity
```

```
     * @param velocity The velocity at which the door is opened
     */
    fun open(velocity: Int) = SerialEmitter.send(Destination.DOOR, OPENCMD_FRAME
or velocity.shl(1))


    /**
     * Closes the door with the specified velocity
     * @param velocity The velocity at which the door is closed
     */
    fun close(velocity: Int) = SerialEmitter.send(Destination.DOOR,
CLOSECMD_FRAME or velocity.shl(1))


    /**
     * Checks if the door mechanism has finished the close/open operation
     * @return false if the door mechanism is busy, true otherwise
     */
    fun finished(): Boolean = !SerialEmitter.isBusy()


}
```

E. Código Kotlin - *Serial Emmiter*

```
fun main() {
    HAL.init()
    SerialEmitter.send(SerialEmitter.Destination.LCD, 0x15)
}


/**
 * 22/5/2023
 *
 * Serial emitter for sending data to the door controller or the LCD of the
[AccessControlSystem]
 * @property Destination Serial emitter destinations.
 * @author Bernardo Pereira
 * @author AntÃ³nio Paulino
 * @see LCD
 * @see DoorMechanism
 * @see HAL
 */
object SerialEmitter {

    enum class Destination(val mask: Int) { LCD(0b00000010), DOOR(0b00000100) }

    /**
     * The wait time in nanoseconds between sending bit signals to the hardware
     */
    private const val DELAY = 1000

    /**
     * The bit mask of the SDX output to the USB Port
     */
    private const val SDXMASK = 0b00000001

    /**
     * The bit mask of the SCLK output to the USB Port
     *
     */
    private const val SCLKMASK = 0b10000000

    /**
     * The bit mask of the BUSY input on the USB Port. Represents if the serial
receiver is busy.
     *
     */
    private const val BUSYMASK = 0b00100000

    /**
     * The size in bits of frames sent through the serial emitter.
     */

    private const val FRAME_SIZE = 5

    /**
     * Initializes the Door Mechanism Control and LCD Serial Receivers by
deselecting them. Sets [isbusy] to false.
     */
```

```
    fun init() {

        HAL.setBits(Destination.LCD.mask)
        HAL.setBits(Destination.DOOR.mask)
        HAL.clrBits(SCLKMASK)

        waitTimeNano(DELAY)
    }

    /**
     * Sends an SCLK pulse to the hardware through the USB Port output port
     */
    private fun clkPulse() {
        waitTimeNano(DELAY)

        HAL.setBits(SCLKMASK)
        waitTimeNano(DELAY)

        HAL.clrBits(SCLKMASK)
        waitTimeNano(DELAY)
    }

    /**
     * Sends data to the destination Serial Receiver according to the serial
communication protocol for the [AccessControlSystem]
     * @param addr The destination
     * @param data The data to send
     */
    fun send(addr: Destination, data: Int) {



        HAL.clrBits(addr.mask) //Select the destination Serial Receiver
        waitTimeNano(DELAY)

        for (i in 0 until FRAME_SIZE) { //Write frame to LCD
            val sdx = (1.shl(i) and data).shr(i)
            HAL.writeBits(SDXMASK, sdx)
            clkPulse()
        }

        HAL.setBits(addr.mask) //Deselect the destination Serial Receiver

        if(addr == Destination.DOOR) {
            while (!DoorMechanism.finished()) {
                waitTimeMilli(DELAY / 10)
            }
        }

        waitTimeNano(DELAY)
    }

    /**
     * Checks if the Door Serial Receiver is busy
     * @return true if the Door Serial Receiver is busy, false otherwise
     */
    fun isBusy(): Boolean = HAL.isBit(BUSYMASK)
```

```
}
```