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.

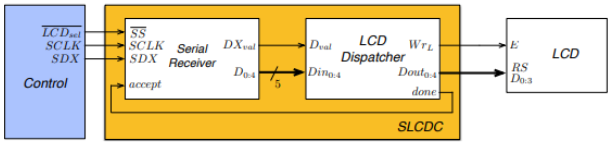


Figura 1 – Diagrama de blocos do Serial LCD Controller

1 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

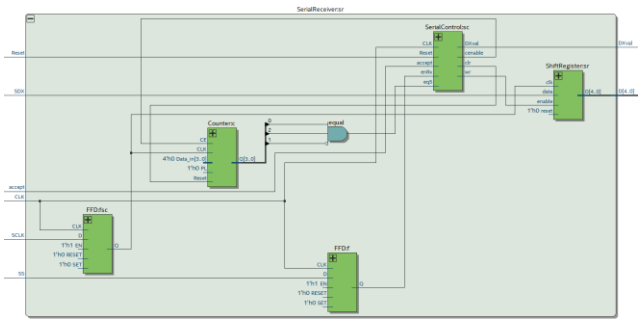


Figura 2 – Diagrama de blocos do Serial Receiver

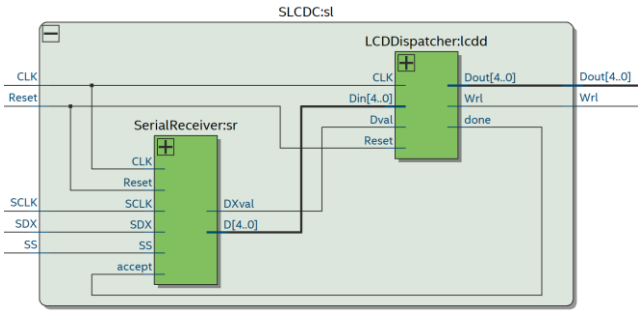


Figura 3 – Diagrama de blocos do SLCDC

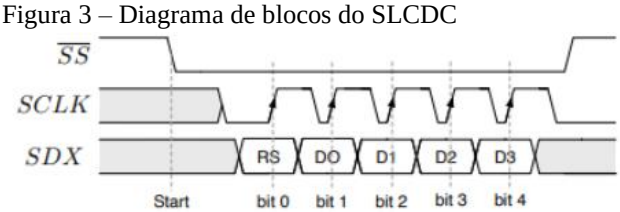


Figura 4 – Diagrama de temporal do SLCDC

2 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 5.

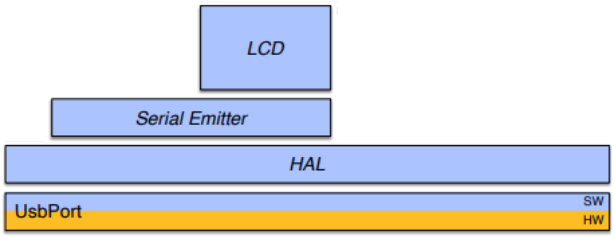
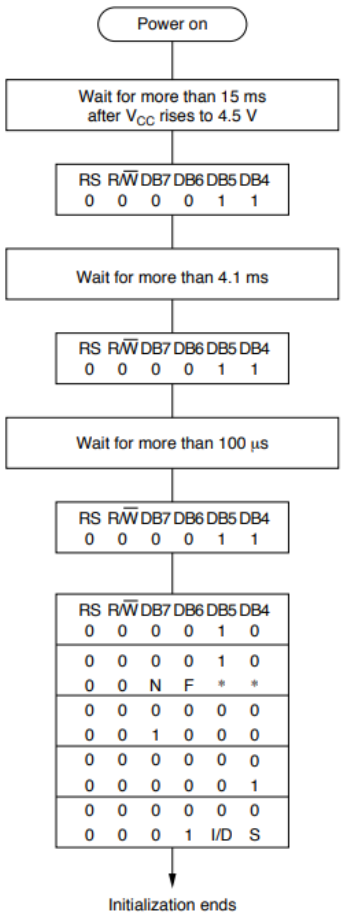


Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *SLCDC.*

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

2.1 LCD

Init que tem a seguinte sequencia de comandos:



A fução wrteNibbleSerial c faz a transformação á data para incluir o valor do rs e chama a função send do serial emitter.
wruteNibble apenas chama writeNibbleSerial.
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.
writeCMD chama writeByte indicando o rs a falso ooque quer dizer que é um comando de LCD a executar.
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.

2.2 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.

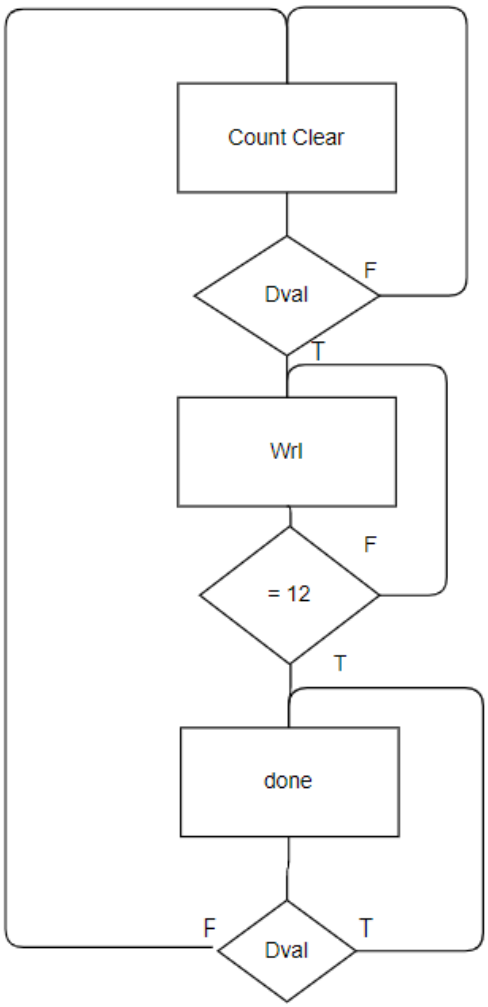
No send em 1º lugar ativa o SS com clrBits, depois tem um ciclo no qual faz clrBits do SCLOCK divide a data que recebe bit a bit com clrBits ou setBits dependendo do valor e irá escrever isso em série ao invés de o escrever em paralelo.
Após isto coloca SCLOCK a 1 e faz 5 iterações. Depois do ciclo o 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.

3 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.

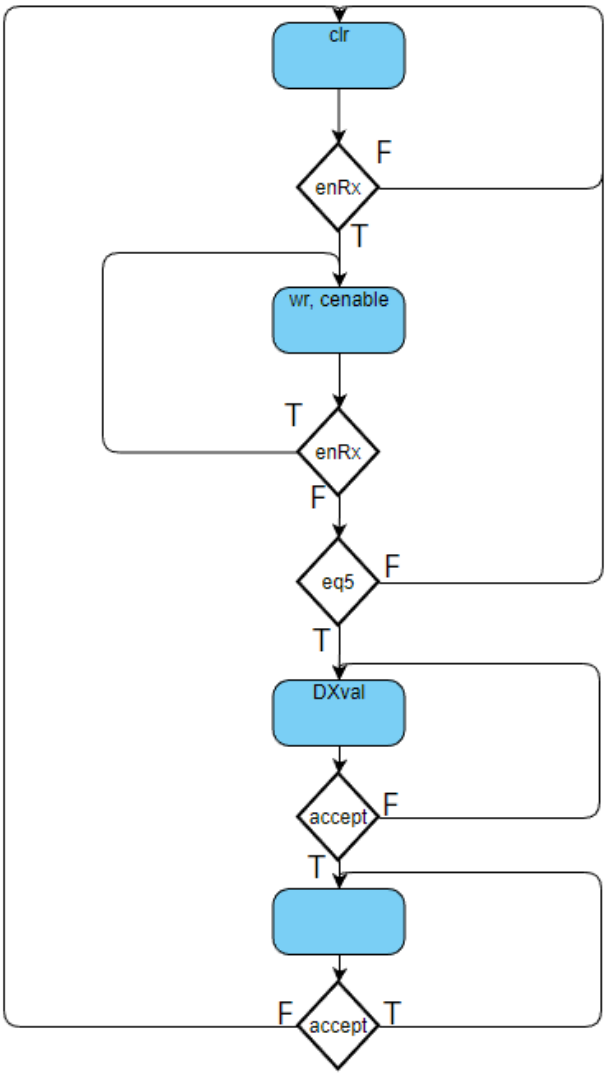
A. Descrição VHDL do bloco *SLCDC*

a. LCDDispatcher



SerialReceiver(i, ii)

i.SerialControl



ii.ShiftRegister

Guarda valores(bits) 1 a 1 até que consiga fazer uma palavra de 5 bits.

B. 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
```

C. Código *Kotlin – LCD*

```
object LCD {
    private const val RS = 0x04
    private const val Enable = 0x02
    private const val DATA = 0x78
    private const val HIGH = 0xF0
    private const val LOW = 0x0F

    // Escreve um nibble de comando/dados no LCD em paralelo
    private fun writeNibbleParallel(rs: Boolean, data: Int) {
        HAL.writeBits(RS, (if (rs) 1 else 0).shl(2))
        HAL.writeBits(DATA, data.shl(3))
        Thread.sleep(0, 40)
        HAL.setBits(Enable)
        Thread.sleep(0, 230)
```

```
        HAL.clrBits(Enable)
        Thread.sleep(0, 270)
    }

    // Escreve um nibble de comando/dados no LCD em série
    private fun writeNibbleSerial(rs: Boolean, data: Int) {
        val r = if (rs) 1 else 0
        SerialEmitter.send(SerialEmitter.Destination.LCD, data.shl(1) + r)
    }

    // Escreve um nibble de comando/dados no LCD
    private fun writeNibble(rs: Boolean, data: Int) {
//        writeNibbleParallel(rs, data)
        writeNibbleSerial(rs, data)
    }

    // Escreve um byte de comando/dados no LCD
    private fun writeByte(rs: Boolean, data: Int) {
        writeNibble(rs, data.and(HIGH).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(40)
        writeNibble(false, 0x03)
        Thread.sleep(5)
        writeNibble(false, 0x03)
        Thread.sleep(1)
        writeNibble(false, 0x03)
        Thread.sleep(1)
        writeNibble(false, 0x02)
        Thread.sleep(1)
        writeCMD(0x28)
        Thread.sleep(1)
        writeCMD(0x08)
        Thread.sleep(1)
```

```
        writeCMD(0x01)
        Thread.sleep(1)
        writeCMD(0x06)
        Thread.sleep(1)
        writeCMD(0x0E)
    }

    // 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(1)
    }
}
fun main(){
    LCD.init()
    while (true){
        println("clear")
        LCD.clear()
        println("write")
        LCD.write("ajuda")
        println("cursor")
        LCD.cursor(1,0)
        Thread.sleep(2000)
    }
}
```

D. Código *Kotlin – SeriaEmitter*

```
object SerialEmitter {
    enum class Destination { LCD, DOOR }
```

```
    private const val SS = 2
    private const val SDX = 4
    private const val SCLOCK = 8

    fun init() {
        HAL.init()
        HAL.setBits(SS)
    }

    fun send(addr: Destination, data: Int) {
        println("send : $data")
        HAL.clrBits(SS)
        for (i in 0 until 5) {
            HAL.clrBits(SCLOCK)
            val b = data.and(1.shl(i))
            if (b == 0) {
                HAL.clrBits(SDX)
                print(0)
            } else {
                HAL.setBits(SDX)
                print(1)
            }
            HAL.setBits(SCLOCK)
        }
        HAL.clrBits(SCLOCK)
        HAL.setBits(SS)
        Thread.sleep(1)
        println()
    }

    fun isBusy(): Boolean {
        return false
    }
}

fun main() {
    SerialEmitter.init()
    for (i in 31 downTo 0) {
        SerialEmitter.send(SerialEmitter.Destination.LCD, i)
    }
}
```