O módulo *Keyboard Reader* é constituído por três blocos principais: *i*) o descodificador de teclado (*Key Decode*); *ii*) o bloco de armazenamento (designado por *Ring Buffer*); e *iii*) o bloco de entrega ao consumidor (designado por *Output Buffer*). Neste caso o módulo *Control*, implementado em *software*, é a entidade consumidora.

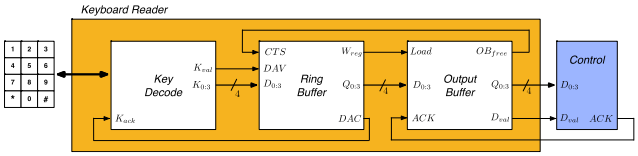
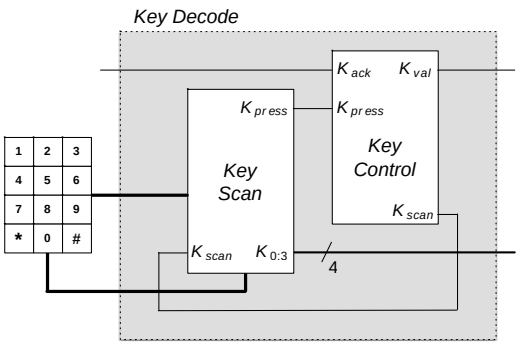


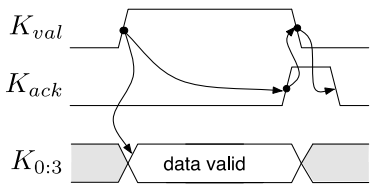
Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

1 Key Decode

O bloco *Key Decode* implementa um descodificador de um teclado matricial 4x3 por *hardware*, sendo constituído por três sub-blocos: *i)* um teclado matricial de 4x3; *ii)* o bloco *Key Scan*, responsável pelo varrimento do teclado; e *iii)* o bloco *Key Control*, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura 2a. O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Ring Buffer*), define que o sinal K_{val} é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal K_{ack} for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 2b.



a) Diagrama de blocos



b) Diagrama temporal

Figura 2 – Bloco *Key Decode*

O bloco *Key Scan* foi implementado de acordo com o diagrama de blocos representado na Figura 3. Escolhemos a versão I pois foi a que o professor recomendou e ao longo do desenvolvimento do trabalho consideramos ser uma versão que nos agradou.

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

A descrição hardware do bloco *Key Decode* em VHDL encontra-se no Anexo DA.

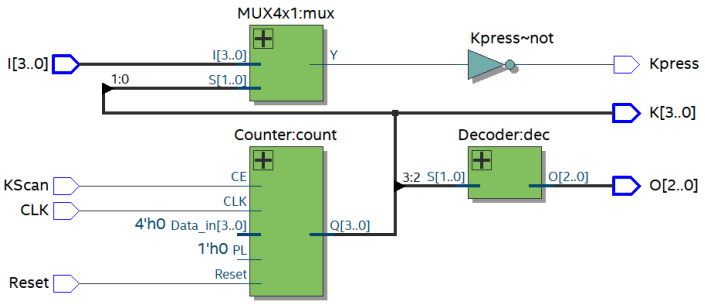


Figura 3 - Diagrama de blocos do bloco *Key Scan*

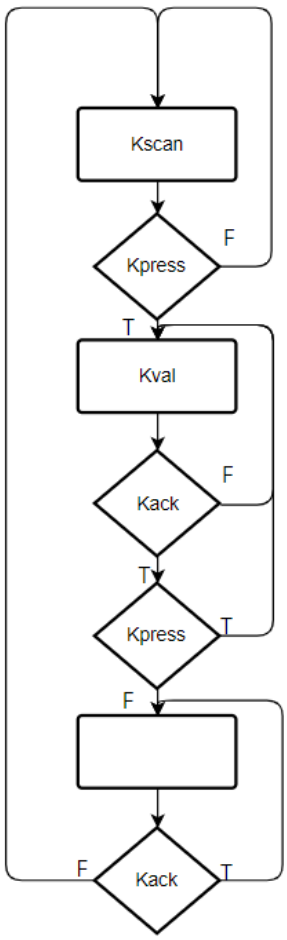


Figura 4 – Máquina de estados do bloco *Key Control*

Com base nas descrições do bloco *Key Decode* implementou-se parcialmente o módulo *Keyboard Reader* de acordo com o esquema elétrico representado no Anexo D.
Alterarámos o clock, dividindo-o no Key Decode por 1000, pois como estava previamente, demorava demasiado a fazer o scan.

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.

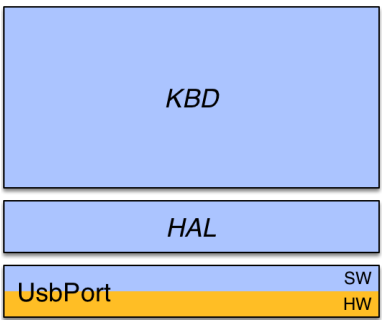


Figura 5 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

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

2.1 *HAL*

No desenvolvimento do HAL não adicionamos funções extra.
Iniciámos com variável inicialize a falso para que programa faça o 1º init, mas no caso da placa não tiver sido reprogramada, dá reset ao estado da placa e coloca lastState (marcador do último input do usbPort).
a 0.
Função readBits lé valor de output nos bits correspondentes a máscara enquanto restantes bits retornam a 0. Desenvolvemos 5 funções com as seguintes funcionalidades:

isBits verifica o valor de um bit em específico.
setBits coloca valores correspondentes a máscara a 1 e deixa os restantes como estavam.
clrBits coloca valores correspondentes à máscara a 0 e deixa os restantes como estavam.
writeBits, escreve valor recebido nos bits indicados pela máscara (restantes bits mantém-se).

A utilização da linha de código:

```
lastState = lastState.or(mask)
```

nas funções setBits, clrBits e writeBits serve para manter valor de lastState atualizado.

2.2 *KBD*

O desenvolvimento o software de KBD começa com a função init que inicia escrita e leitura do usbPort chamando o HAL.init e faz o clearBits(HAL.clrBits) à máscara do Kack (para este começar a 0).
A função getKey lê o código da máscara que está no valor de cada tecla e transforma-o no devido caracter caso seja válida, é dado ack da tecla, espera tecla ser validada e devolve o char correspondente á tecla que leu. Isto no caso de Kval estar ativo. Se Kval estiver inativo retorna NONE.
Por fim, a função waitKey recebe período (timeout) e tem um ciclo que tenta ler teclas. Se ler uma tecla válida retorna essa tecla. Se não, continua à procura até o tempo de o ciclo exceder e retorna NONE.

3 Conclusões

Tivemos de alterar um clock, dividindo-o no Key Decode por 1000, pois como estava previamente, demorava demasiado a fazer o scan.

A. Descrição VHDL do bloco *Key Decode*

O bloco Key Decode tem 3 componentes:
-Teclado(externo)
em vhdl:
-Key Scan(leitor que descodifica a informação obtida para a tecla correspondente do teclado)
-Key Control(diz ao kscan quando fazer scan, qnd tecla carregada o kval passa a 1 avisando o próximo componente sobre qual a tecla a ser lida.

A componente Key Scan só varre o teclado quando var Kscan esta ativa. Quando deteta pressão de tecla, retorna Kpress a 1 e k correspondente à tecla premida.

A componente Key Control é a máquina de estados que recebe kack e kpress, e dá como output Kval e Kscan.

Kval é output do Key Decode, que quando uma tecla é premida é validada(kval) e o kscan é ativo quando não há nenhuma tecla premida.

Kscan interpreta o código único pra cada tecla.
Kcontrol gera informação e dá como output, quando uma tecla premida está codificada, kval a 1 quando deteta uma tecla valida.

Kack é um outro componente fora do Key Decode que diz que quando a informação que enviou como output já tiver sido recebida, já pode passar ao próximo scan do teclado.

B. Atribuição de pinos do módulo *Keyboard Reader*

```
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 CLK

# inputs
set_location_assignment PIN_C10 -to Kack
set_location_assignment PIN_C11 -to Reset

# outputs
set_location_assignment PIN_A8 -to Kval
set_location_assignment PIN_D13 -to K[0]
set_location_assignment PIN_C13 -to K[1]
set_location_assignment PIN_E14 -to K[2]
set_location_assignment PIN_D14 -to K[3]

#Keypad
set_location_assignment PIN_W5     -to I[0]
set_location_assignment PIN_AA14   -to   I[1]
set_location_assignment PIN_W12    -to I[2]
set_location_assignment PIN_AB12   -to   I[3]
set_location_assignment PIN_AB11   -to   O[0]
set_location_assignment PIN_AB10   -to   O[1]
set_location_assignment PIN_AA9    -to    O[2]
```

C. Código *Kotlin - HAL*

```kotlin
import isel.leic.UsbPort

object HAL {
    private var lastState = 0
    fun init() {
        UsbPort.write(0)
        lastState = 0
    }

    fun readBits(mask: Int): Int { // mask = 00001111
        val value = UsbPort.read() // 00011011
        return value.and(mask) // 00001011
    }

    fun isBit(mask: Int): Boolean {  // mask = 00000001
        val value = UsbPort.read() // 00000010
        val newValue = value.and(mask) // 00000000
        if (newValue == 0) { // reads 0, so false
            return false
        }
        return true
    }

    fun setBits(mask: Int) { // mask = 00001111
        val value = lastState.or(mask) // 01000001
        UsbPort.write(value) // 01001111
        lastState = value
    }

    fun clrBits(mask: Int) { //mask = 0000011
        val value = lastState // 01001110
        val newMask = mask.inv() // 11111100
        UsbPort.write(value.and(newMask)) // 01001100
        lastState = value.and(newMask)
    }

    fun writeBits(mask: Int, value: Int) { // mask = 00001111 value = 00001001
        val value2 = lastState // 01001101
```

```
        val newMask = mask.inv() // 11110000
        val newValue = value2.and(newMask) // 01000000
        UsbPort.write(value.or(newValue)) // 01001001
        lastState = value.or(newValue)
    }
}
```

D. Código *Kotlin - KBD*

```
object KBD {
    private const val Kval = 1
    private const val Kack = 1
    private const val K = 30

    fun init() {
        HAL.init()
    }

    const val NONE = 0.toChar()

    fun getKey(): Char {
        if (HAL.isBit(Kval)) {
            val c = when (HAL.readBits(K).shr(1)) {
                0 -> '1'
                1 -> '4'
                2 -> '7'
                3 -> '*'
                4 -> '2'
                5 -> '5'
                6 -> '8'
                7 -> '0'
                8 -> '3'
                9 -> '6'
                10 -> '9'
                11 -> '#'
                else -> NONE
            }
            HAL.setBits(Kack)
            while (HAL.isBit(Kval)) {
                Thread.sleep(10)
            }
            HAL.clrBits(Kack)
            return c
        }
```

```
        return NONE
    }

    fun waitKey(timeout: Long): Char {
        val timeInit = System.currentTimeMillis()
        while (true) {
            val time = System.currentTimeMillis()
            val c = getKey()
            if (c != NONE) return c
            if (time - timeInit >= timeout) return NONE
        }
    }
}
```