O módulo *Keyboard Reader* implementado é constituído por dois blocos principais: *i*) o descodificador de teclado (*Key Decode*); e *ii*) o bloco de armazenamento e de entrega ao consumidor (designado por *Key Transmitter*), conforme ilustrado na Figura 1. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

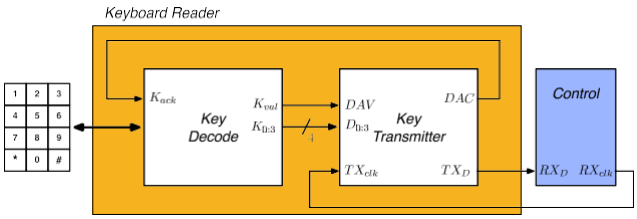
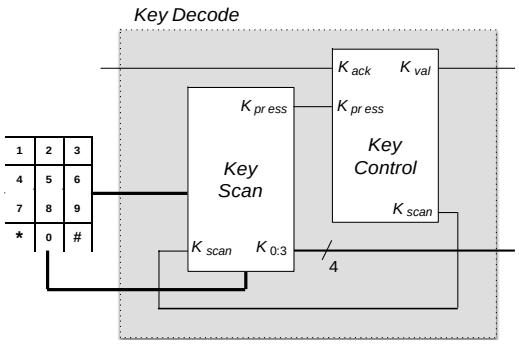


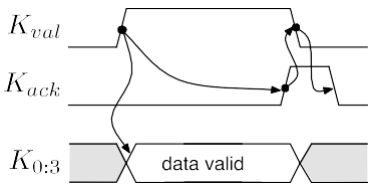
Figura 1 – Diagrama de blocos do módulo *Keyboard Reader*

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 *Key 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 opção 1 das 3 para o bloco Key Scan, pois foi a 1ª versão implementada e neste momento do projeto não tivemos qualquer consideração pelas outras porque não tivemos motivos para tal no estado atual do projeto.

O bloco *Key Control* foi implementado pela máquina de estados representada em *ASM-chart* na Figura 4. O Key Control ativa o sinal Kscan no estado inicial, desde que o Kpress esteja desativo, ou seja, enquanto não houver tecla premida, o que faz com que o CE do contador esteja ativo e estejamos a varrer o teclado inteiro. Quando detetamos uma tecla premida, paramos **imediatamente** de ativar Kscan e passamos para outro estado onde ficamos à espera de Kack para passarmos a outro estado ainda, onde ficamos à espera que a tecla seja libertada para que possamos recomeçar o varrimento ao teclado inteiro.

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

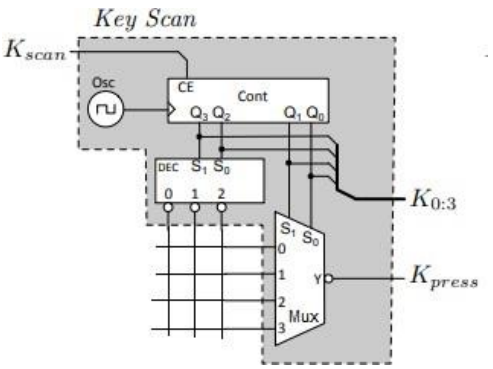


Figura 3 - Diagrama de blocos do bloco *Key Scan*

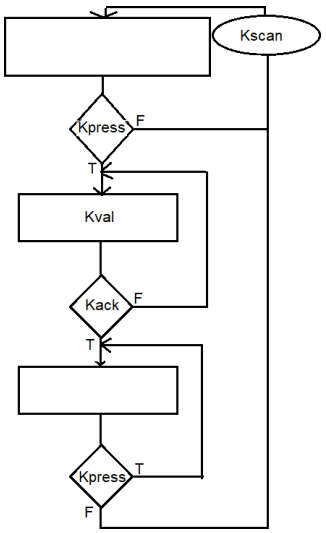


Figura 4 – Máquina de estados do bloco *Key Control*

Com base nas descrições do bloco *Key Decode* implementou-se o módulo *Keyboard Reader*.

2 Key Transmitter

O módulo *Key Transmitter* implementa uma estrutura de armazenamento de dados, com capacidade de uma palavra de quatro bits. A escrita de dados no *Key Transmitter* inicia-se com a ativação do sinal *DAV* (*Data Available*) pelo sistema produtor, neste caso pelo *Key Decode*, indicando que tem dados para serem armazenados. Assim que possivel armazenar informação, o *Key Transmitter* escreve os dados $D_{0:3}$ em memória interna. Concluída a escrita em memória, ativa o sinal *DAC* (*Data Accepted*) para informar o sistema produtor que os dados foram aceites. O sistema produtor mantém o sinal *DAV* ativo até que *DAC* seja ativado. O *Key Transmitter* só desativa *DAC* depois de *DAV* ter sido desativado.

A implementação do *key Transmitter* é baseada numa máquina de controlo (*Key Transmitter Control*), etc., conforme o diagrama de blocos apresentado na Figura 5.

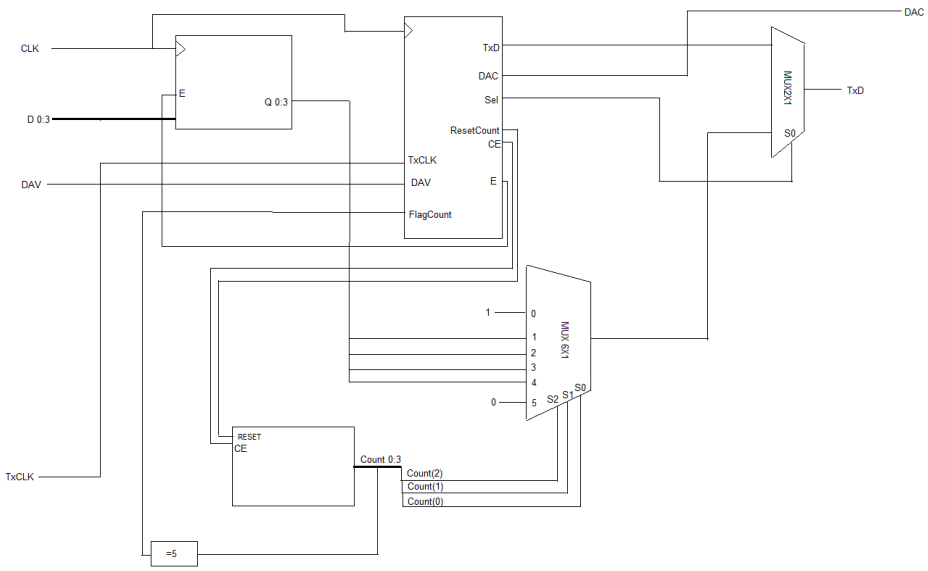


Figura 5 – Diagrama de blocos do *Key Transmitter*

O bloco *Key Transmitter Control* do *Key Transmitter* é também responsável pela interação com o sistema consumidor, neste caso o módulo *Control*.

O bloco Key Transmitter Control foi implementado de acordo com a máquina de estados representada na Figura 6. Estado Inicial : Neste estado , apenas é necessário manter ativo o TxD , visto que apenas durante o envio da trama ,que contem a informação relativa à tecla pressionada, é que o valor altera consoante os bits de informação. Neste estado ativamos também o sinal ResetCount , que é o sinal que faz o Reset ao Count(Contador) , para termos a certeza de quando iniciarmos a contagem, o bloco começar a contar do zero. Apenas avançamos de estado assim o sinal DAV(sinal que indica que irá ocorrer uma escrita) fique ativo. Estado_Store : Neste estado damos início á receção e armazenamento de dados , para que isso aconteça precisamos ativar o sinal E , que faz o Enable do registo para guardar o valor à entrada e mantemos sempre o TxD ativo , exceto durante o envio de uma trama. Como a escrita ocorre em paralelo , basta esperar um ciclo de clock da máquina de estados , que é o mesmo clock que o do registo , para certificarmo-nos que o valor foi guardado com sucesso. Estado_store_end : Este estado simboliza o fim do armazenamento de dados , como tal, para indicarmos a entidade consumidora que este armazenamento ocorreu com sucesso ativamos o sinal DAC(Data Accepted) e mantemos o TxD ativo, pois ainda não estamos num estado de envio de informação. Para avançar de estado apenas temos de verificar a entrada DAV passar a '0' , que indica que a escrita foi concluída. Estado_send : Neste estado damos início ao envio de uma trama logo, como mencionado anteriormente , desativamos o TxD. Durante este estado verificamos a entrada TxCLK, gerada pelo software , pois quando for colocado a '1' indica que o software está pronto a receber a trama. Estado_send end : Neste estado ocorre o fim da transmissão de dados , para tal ativamos o sel(seletor do Mux) para enviarmos o valor de TxD que corresponde ao Start bit, 4 bits de dados e Stop bit, que estão na entrada do MuxKT,

A descrição hardware do bloco *Key Transmitter Control* em VHDL encontra-se no Anexo B.

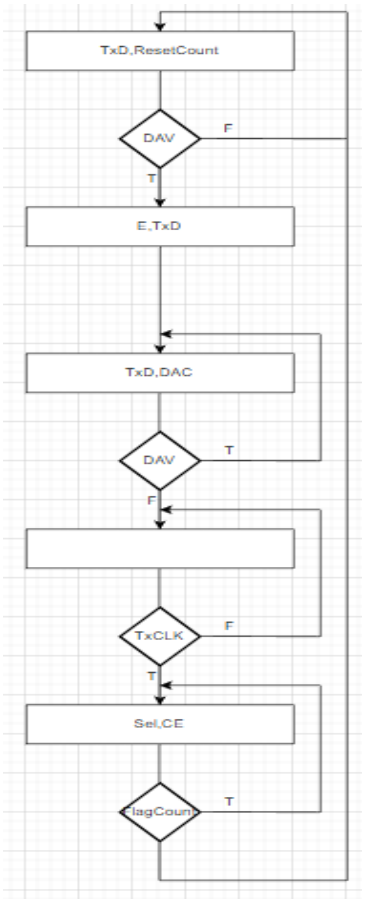


Figura 6 - Máquina de estados do bloco *Key Transmitter Control*

Com base nas descrições do bloco *Key Decode* e do bloco *Key Transmitter* implementou-se o módulo *KeyBoard Reader*.

Durante o desenvolvimento do bloco Key Transmitter , reparamos que existe uma "dessincronia" entre o clock do software e hardware , para resolver este problema tivemos de implementar na nossa máquina de estados do Key Transmitter um estado que verifica o clock gerado pelo software antes de enviar a trama e durante o envio da trama.

3 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* seguindo a arquitetura lógica apresentada na Figura 7.

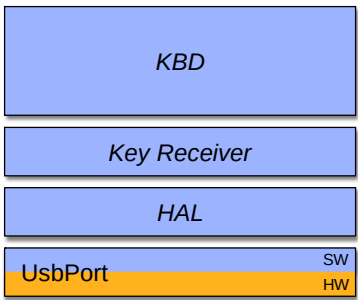


Figura 7 – Diagrama lógico do módulo *Control* de interface com o módulo *Keyboard Reader*

Os módulos de software *HAL*, *KeyReceiver* e *KBD* desenvolvidos são descritos nas secções 3.1., 3.2 e 3.3, e o código fonte desenvolvido nos Anexos D, **Error! Reference source not found.** e **Error! Reference source not found.**, respetivamente.

3.1 *HAL*

O HAL é um modolo software responsável por comunicar entre o Software e Hardware, obtendo dados do porto de entrada e escrevendo dados no porto de saída do UsbPort. Para tal, recorremos às funções readBits(), que dada uma máscara como parâmetro lê os valores da entrada especificados pela máscara do UsbPort, a função isBit(), que verifica se um determinado bit está ativo ou não,

3.2 KeyReceiver

O módulo KeyReceiver lê os bits enviados pelo KeyBoard Reader, para tal usa o módulo abaixo dele, o HAL, lendo um determinado bit do porto de entrada e trnaformando a informação recebida em série em dados. A função rcv(), recebe a trama com o código da tecla premida seguido do startBit, verificando se este está a '1' e antecipado do stopBit, verificando se este está a '0'. Se estas condições estão válidas, a trama é considerada válida e a função retorna o código da tecla premida.

3.3 KBD

O módulo KBD é construído em cima do KeyReceiver, ou seja, usa-o, logo a função getSerialKey() irá usar a função rcv() do KeyReceiver, recebido o código da tecla premida, a função getKeySerial() determina qual a tecla premida,

retornando-a ou retornando NONE caso nenhuma tecla tenha sido premida. A função waitKey() usa a função getSerialKey() mas espera um certo tempo antes de encerrar, caso seja detedada tecla premida, retorna-a, caso contrário retorna NONE.

4 Conclusões

Nesta fase do projeto concluimos que as dessincronias entre CLK's afetam a forma e construção de alguns módulos, exemplo disto foi presenciado na construção do módulo Key Transmitter.

A. Descrição VHDL do bloco *Key Decode*

```
entity KeyDecode is
     Port (
     MCLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
     Kack : in STD_LOGIC;
     Kentry : in STD_LOGIC_VECTOR(3 downTo 0);
     notDecOut : out STD_LOGIC_VECTOR(2 downTo 0);
     K : out STD_LOGIC_VECTOR(3 downTo 0);
     Kval : out STD_LOGIC
      );
end KeyDecode;

Architecture Behavioral of KeyDecode is

   component KeyScan is
        Port (Kscan : in STD_LOGIC;
           MCLK : in std_logic;
           Kentry : in STD_LOGIC_VECTOR(3 downTo 0);
           Kpress : out  std_logic;
           notDecOut : out STD_LOGIC_VECTOR(2 downTo 0);
           K : out STD_LOGIC_VECTOR(3 downTo 0)
            );
        end component;

   component KeyControl is
        Port (RESET : in STD_LOGIC;
         MCLK : in STD_LOGIC;
         Kack : in STD_LOGIC;
         Kpress : in STD_LOGIC;
         Kval : out STD_LOGIC;
         Kscan : out STD_LOGIC
            );
        end component;
 signal Kpress_signal : std_logic;
 signal Kscan_signal : std_logic;

 begin

 U0:      KeyScan
          port map(Kscan => Kscan_signal, MCLK => MCLK, Kentry => Kentry, Kpress => Kpress_signal, notDecOut =>
          notDecOut, K => K);

 U1:      KeyControl
          port map(MCLK => MCLK, RESET => RESET, Kack => Kack, Kpress => Kpress_signal, Kval => Kval, Kscan =>
          Kscan_signal);


 end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity KeyScan is
     Port (Kscan : in STD_LOGIC;
           MCLK : in std_logic;
           Kentry : in STD_LOGIC_VECTOR(3 downTo 0);
           Kpress : out  std_logic;
           notDecOut : out STD_LOGIC_VECTOR(2 downTo 0);
           K : out STD_LOGIC_VECTOR(3 downTo 0)
           );
end KeyScan;

Architecture Behavioral of KeyScan is

        Component counter_4bit is
        Port (clk : in STD_LOGIC;
         clr : in STD_LOGIC;
         CE : in STD_LOGIC;
         q : out STD_LOGIC_VECTOR (3 downto 0)
            );
        end component;

        Component Decoder is
        Port (S1 : in STD_LOGIC;
         S0 : in STD_LOGIC;
         Output : out STD_LOGIC_VECTOR(2 downTo 0)
            );
        end component;

        Component MUX4X1 is
        Port (S1 : in STD_LOGIC;
         S0 : in STD_LOGIC;
         In0 : in STD_LOGIC;
         In1 : in STD_LOGIC;
         In2 : in STD_LOGIC;
         In3 : in STD_LOGIC;
         Y : out STD_LOGIC
             );
       end component;

signal Qsel : std_logic_vector(3 downto 0);
signal decOut : std_logic_vector(2 downTo 0);
signal MuxOut : std_logic;

begin

Kpress <= not MuxOut;
notDecOut <= not decOut;
K <= Qsel;

U0: counter_4bit
   port map (CE => Kscan, q => Qsel, CLK => MCLK, clr => '0');

U1: Decoder
   port map (S1 => Qsel(3), S0 => Qsel(2), Output => decOut);

U2: MUX4X1
   port map (S1 => Qsel(1), S0 => Qsel(0), In0 => Kentry(0), In1 => Kentry(1), In2 => Kentry(2),
   In3 => Kentry(3), Y => MuxOut);

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter_4Bit is
    Port ( clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           CE : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (3 downto 0)
           );
end counter_4Bit;

architecture Structural of counter_4Bit is

    component CounterLogic_4Bit is
       PORT( operandA : IN std_logic_vector(3 downto 0);
             en : IN STD_LOGIC;
             R : OUT std_logic_vector(3 downto 0)
             );
    end component;

    component register_D_R is
       Generic ( WIDTH : positive := 1 );
       Port( CLK : in STD_LOGIC;
             Enable : in STD_LOGIC;
             RST : in STD_LOGIC;
             D : in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
             Q : out STD_LOGIC_VECTOR(WIDTH-1 downto 0));
    end component;

    signal operandA, result : std_logic_vector(3 downto 0);

    begin

    U0: CounterLogic_4Bit
          PORT MAP(    operandA => operandA, R => result, en => CE);

    U1: register_D_R
          Generic map (  WIDTH => 4 )
          Port map(
             CLK => clk,
             Enable => '1',
             RST => clr,
             D => result,
             Q => operandA
          );

    q <= operandA;

end Structural;
```

```
library IEEE;
use  IEEE.STD_LOGIC_1164.ALL;

entity register_D_R is
    generic(
        WIDTH : POSITIVE := 1
    );
    Port ( CLK : in STD_LogiC;
           Enable : in STD_LOGIC;
           D : in STD_Logic_VECTOR (WIDTH-1 downto 0);
           Q : out STD_Logic_VECTOR (WIDTH-1 downto 0);
           RST : in STD_LOGIC);
end register_D_R;

architecture Behavioral of register_D_R is

begin

    Q <= (others => '0') when RST = '1' else D when (rising_edge(clk) and Enable = '1');

end Behavioral;
```

```
library IEEE;
use  IEEE.STD_LOGIC_1164.ALL;

entity CounterLogic_4bit is
    Port ( operandA : in STD_LogiC_VECTOR (3 downto 0);
           en : in STD_LOGIC;
           R : out STD_Logic_VECTOR (3 downto 0)
            );
end CounterLogic_4bit;

architecture Structural of CounterLogic_4bit is

component adder_4bit is
    Port( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          Cin : in STD_LOGIC;
          S : out STD_LOGIC_VECTOR(3 downto 0);
          Cout : out STD_LOGIC);
end component;

component MUX2x1 is
    Port(Sel : in STD_LOGIC;
         In0 : in STD_LOGIC_VECTOR(3 downTo 0);
         In1 : in STD_LOGIC_VECTOR(3 downTo 0);
         Y : out STD_LOGIC_VECTOR(3 downTo 0)
         );
end component;

signal operandB : std_logic_vector(3 downTo 0);

begin

U0:   adder_4bit
      port map(A => operandA, B => operandB, Cin => '0', S => R, Cout => open);

U1: MUX2X1
      port map(In0 => "0000", In1 => "0001", Sel => en, Y => operandB);

end Structural;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX2X1 is
    Port (
        Sel : in STD_LOGIC;
        In0 : in STD_LOGIC_VECTOR(3 downTo 0);
        In1 : in STD_LOGIC_VECTOR(3 downTo 0);
        Y : out STD_LOGIC_VECTOR(3 downTo 0)
    );

end MUX2X1;

architecture Behavioral of MUX2X1 is
begin

Y <= In0 when sel = '0' else In1;


end Behavioral;
```

```
library IEEE;
use  IEEE.STD_LOGIC_1164.ALL;

entity FA is
    Port ( A : in STD_Logic;
           B : in STD_Logic;
           Cin : in STD_Logic;
           S : out STD_Logic;
           Cout : out STD_Logic);
end FA;

architecture Behavioral of FA is
    signal xor_ab   : std_logic;
    signal and_ab   : std_logic;
begin
    xor_ab    <= A xor B;
    and_Ab    <= A and B;

    S         <= xor_ab xor Cin;

    Cout      <= and_ab or ( Cin and xor_ab );

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX4X1 is
    Port (
        S1 : in STD_LOGIC;
        S0 : in STD_LOGIC;
        In0 : in STD_LOGIC;
        In1 : in STD_LOGIC;
        In2 : in STD_LOGIC;
        In3 : in STD_LOGIC;
        Y : out STD_LOGIC
    );

end MUX4X1;

architecture Behavioral of MUX4X1 is

begin

check : process(S0, S1)
begin

    if(S1 = '0' and S0 = '0') then
    Y <= In0;
    elsif(S1 = '0' and S0 = '1') then
    Y <= In1;
    elsif(S1 = '1' and S0 = '0') then
    Y <= In2;
    else
    Y <= In3;
    end if;

end process;


end Behavioral;
```

```
library IEEE;
use  IEEE.STD_LOGIC_1164.ALL;

entity adder_4bit is
    Port ( A : in STD_LogiC_VECTOR (3 downto 0);
           B : in STD_Logic_VECTOR (3 downto 0);
           Cin : in STD_Logic;
           S : out STD_Logic_VECTOR (3 downto 0);
           Cout : out STD_Logic);
end adder_4bit;

architecture Structural of adder_4bit is
component FA is
    Port ( A : in STD_Logic;
           B : in STD_Logic;
           Cin : in STD_Logic;
           S : out STD_Logic;
           Cout : out STD_Logic);
end component;

signal carry      : std_logic_vector(3 downto 1);
begin
U0:       FA
          port map(A => A(0), B => B(0), Cin => Cin, S => S(0), Cout => carry(1));

U1:       FA
          port map(A => A(1), B => B(1), Cin => carry(1), S => S(1), Cout => carry(2));

U2:       FA
          port map(A => A(2), B => B(2), Cin => carry(2), S=> S(2), Cout => carry(3));

U3:       FA
          port map(A => A(3), B => B(3), Cin => carry(3), S=> S(3), Cout => Cout);

end Structural;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder is
    Port (
        S1 : in STD_LOGIC;
        S0 : in STD_LOGIC;
        Output : out STD_LOGIC_VECTOR(2 downTo 0)
    );

end Decoder;

architecture Behavioral of Decoder is

begin

check : process(S0, S1)
begin

    if(S1 = '0' and S0 = '0') then
        Output <= "001";
    elsif(S1 = '0' and S0 = '1') then
        Output <= "010";
    elsif(S1 = '1' and S0 = '0') then
        Output <= "100";
    end if;
end process;


end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity KeyControl is
    Port (
        RESET : in STD_LOGIC;
        MCLK : in STD_LOGIC;
        Kack : in STD_LOGIC;
        Kpress : in STD_LOGIC;
        Kval : out STD_LOGIC;
        Kscan : out STD_LOGIC
    );
end KeyControl;

architecture Behavioral of KeyControl is

    type  STATE_TYPE is (
          ESTADO_INIT,
          ESTADO_KVAL,
          ESTADO_WAIT_NOTKPRESS
    );

    signal CurrentState, NextState : STATE_TYPE;

begin

    CurrentState <= ESTADO_INIT when RESET = '1' else NextState when rising_edge(MCLK);


    GenerateNextState: process(CurrentState)
    begin

        case CurrentState is
        when ESTADO_INIT           => if(Kpress = '1') then
                                          NextState <= ESTADO_KVAL;
                                      else
                                          NextState <= ESTADO_INIT;
                                      end if;

        when ESTADO_KVAL           => if(Kack = '1') then
                                          NextState <= ESTADO_WAIT_NOTKPRESS;
                                      else
                                          NextState <= ESTADO_KVAL;
                                      end if;

        when ESTADO_WAIT_NOTKPRESS => if(Kpress = '0') then
                                          NextState <= ESTADO_INIT;
                                      else
                                          NextState <= ESTADO_WAIT_NOTKPRESS;
                                      end if;

        end case;
        end process;

    Kval <= '1' when (CurrentState = ESTADO_KVAL) else '0';
    Kscan <= '1' when (CurrentState = ESTADO_INIT and Kpress = '0') else '0';

end Behavioral;
```

B. Descrição VHDL do bloco *Key Transmitter*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity KeyTransmitter is
    Port (
        MCLK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        D : in STD_LOGIC_VECTOR(3 downTo 0);
        DAV : in STD_LOGIC;
        TxCLK : in STD_LOGIC;
        TxD : out STD_LOGIC;
        DAC : out STD_LOGIC
    );

end KeyTransmitter;

architecture Structural of KeyTransmitter is

component KeyTransmitterControl is
        Port(
        MCLK : in STD_LOGIC;
        RESET : in STD_LOGIC;
        FlagCount : in STD_LOGIC;
        DAV : in STD_LOGIC;
        TxCLK : in STD_LOGIC;
        TxD : out STD_LOGIC;
        DAC : out STD_LOGIC;
        Sel : out STD_LOGIC;
        CE : out STD_LOGIC;
        E : out STD_LOGIC;
        ResetCount : out STD_LOGIC
        );
end component;

component MUX2X1_SINGLEBIT is
        Port(
        Sel : in STD_LOGIC;
        In0 : in STD_LOGIC;
        In1 : in STD_LOGIC;
        Y : out STD_LOGIC
        );
end component;

component MUX6X1 is
        Port(
        S2 : in STD_LOGIC;
        S1 : in STD_LOGIC;
        S0: in STD_LOGIC;
        In0 : in STD_LOGIC;
        In1 : in STD_LOGIC;
        In2 : in STD_LOGIC;
        In3 : in STD_LOGIC;
        In4 : in STD_LOGIC;
        In5 : in STD_LOGIC;
        Y : out STD_LOGIC
        );
end component;

component counter_4bit is
        Port(
        clk : in STD_LOGIC;
           clr : in STD_LOGIC;
           CE : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (3 downto 0)
        );
end component;

component Register_D_R
    Generic ( WIDTH : positive := 1 );
        Port (
           CLK : in STD_LogiC;
            Enable : in STD_LOGIC;
            D : in STD_Logic_VECTOR (WIDTH-1 downto 0);
            Q : out STD_Logic_VECTOR (WIDTH-1 downto 0);
            RST : in STD_LOGIC
            );
end component;

signal FlagCount_signal : std_logic;
signal E_signal : std_logic;
signal D_signal : std_logic_vector(3 downTo 0);
signal TxD_signal : std_logic;
signal Sel_signal : std_logic;
signal CE_signal : std_logic;
signal ResetCount_signal : std_logic;
signal Y_signal : std_logic;
signal S_signal : std_logic_vector(2 downTo 0);

begin

FlagCount_signal <= S_signal(2) and (not S_signal(1)) and S_signal(0);


U0:     Register_D_R
        Generic map (  WIDTH => 4 )
        port map(CLK => MCLK, Enable => E_signal, RST => '0', D => D, Q => D_signal);

U1:     KeyTransmitterControl
        port map(MCLK => MCLK, RESET => RESET, FlagCount => FlagCount_signal, DAV => DAV, TxCLK => TxCLK,
        TxD => TxD_signal, DAC => DAC, Sel => Sel_signal, CE => CE_signal, E => E_signal,
        ResetCount => ResetCount_signal);

U2:     MUX2X1_SINGLEBIT
        port map(Sel => Sel_signal, In0 => TxD_signal, In1 => Y_signal, Y => TxD);

U3:     MUX6X1
        port map(In0 => '1', In1 => D_signal(0), In2 => D_signal(1), In3 => D_signal(2), In4 => D_signal(3),
        In5 => '0', S2 => S_signal(2), S1 => S_signal(1), S0 => S_signal(0), Y => Y_signal);

U4:     counter_4bit
        port map(clk => TxCLK, clr => ResetCount_signal, CE => CE_signal, q(2 downTo 0) => S_signal);


end Structural;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX2X1_SINGLEBIT is
    Port (
        Sel : in STD_LOGIC;
        In0 : in STD_LOGIC;
        In1 : in STD_LOGIC;
        Y : out STD_LOGIC
    );

end MUX2X1_SINGLEBIT;

architecture Behavioral of MUX2X1_SINGLEBIT is
begin

Y <= In0 when sel = '0' else In1;


end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX6X1 is
    Port (
        S2 : in STD_LOGIC;
        S1 : in STD_LOGIC;
        S0: in STD_LOGIC;
        In0 : in STD_LOGIC;
        In1 : in STD_LOGIC;
        In2 : in STD_LOGIC;
        In3 : in STD_LOGIC;
        In4 : in STD_LOGIC;
        In5 : in STD_LOGIC;
        Y : out STD_LOGIC
    );

end MUX6X1;

architecture Behavioral of MUX6X1 is

begin

check : process(S0, S1, S2)
begin

    if(S2 = '0' and S1 = '0' and S0 = '0') then
    Y <= In0;
    elsif(S2 = '0' and S1 = '0' and S0 = '1') then
    Y <= In1;
    elsif(S2 = '0' and S1 = '1' and S0 = '0') then
    Y <= In2;
    elsif(S2 = '0' and S1 = '1' and S0 = '1') then
    Y <= In3;
    elsif(S2 = '1' and S1 = '0' and S0 = '0') then
    Y <= In4;
    elsif(S2 = '1' and S1 = '0' and S0 = '1') then
    Y <= In5;
    end process;


    end Behavioral;
```

```
GenerateNextState: process(CurrentState)
begin

    case CurrentState is
    when ESTADO_INIT            => if(DAV = '1') then
                                       NextState <= ESTADO_STORE;
                                   else
                                       NextState <= ESTADO_INIT;
                                   end if;

    when ESTADO_STORE           =>     NextState <= ESTADO_STOREEND;


    when ESTADO_STOREEND        => if(DAV = '0') then
                                       NextState <= ESTADO_SEND;
                                   else
                                       NextState <= ESTADO_STOREEND;
                                   end if;

    when ESTADO_SEND            => if(TxCLK = '1') then
                                       NextState <= ESTADO_SENDEND;
                                   else
                                       NextState <= ESTADO_SEND;
                                   end if;


    when ESTADO_SENDEND             => if(FlagCount = '0') then
                                       NextState <= ESTADO_INIT;
                                   else
                                       NextState <= ESTADO_SENDEND;
                                   end if;

    end case;
    end process;

      TxD <= '1' when (CurrentState = ESTADO_INIT) else '0';
      ResetCount <= '1' when (CurrentState = ESTADO_INIT) else '0';
      E <= '1' when (CurrentState = ESTADO_STORE) else '0';
      TxD <= '1' when (CurrentState = ESTADO_STORE) else '0';
      TxD <= '1' when (CurrentState = ESTADO_STOREEND) else '0';
      DAC <= '1' when (CurrentState = ESTADO_STOREEND) else '0';
      Sel <= '1' when (CurrentState = ESTADO_SENDEND) else '0';
      CE <= '1' when (CurrentState = ESTADO_STOREEND) else '0';

end Behavioral;
```

figura 8 - KeyTransmitterControl

C. Descrição VHDL do módulo *KeyboardReader*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity KeyboardReader is
      Port (
         Kentry : in STD_LOGIC_VECTOR(3 downTo 0);
         notDecOut : out STD_LOGIC_VECTOR(2 downTo 0);
         TxCLK : in STD_LOGIC;
         MCLK : in STD_LOGIC;
         RESET : in STD_LOGIC;
         TxD : out STD_LOGIC
      );
end KeyboardReader;

architecture Structural of KeyboardReader is

component KeyDecode is
   Port (
      MCLK : in STD_LOGIC;
     RESET : in STD_LOGIC;
     Kack : in STD_LOGIC;
     Kentry : in STD_LOGIC_VECTOR(3 downTo 0);
     notDecOut : out STD_LOGIC_VECTOR(2 downTo 0);
     K : out STD_LOGIC_VECTOR(3 downTo 0);
     Kval : out STD_LOGIC
      );
end component;

component KeyTransmitter is
   Port (
      MCLK : in STD_LOGIC;
      RESET : in STD_LOGIC;
      D : in STD_LOGIC_VECTOR(3 downTo 0);
      DAV : in STD_LOGIC;
      TxCLK : in STD_LOGIC;
      TxD : out STD_LOGIC;
      DAC : out STD_LOGIC
      );
end component;

signal Kack_signal : std_logic;
signal DAV_signal : std_logic;
signal D_signal : std_logic_vector(3 downTo 0);

begin


U0:      KeyTransmitter
         port map(MCLK => MCLK, RESET => RESET, DAV => DAV_signal, D => D_signal, TxCLK => TxCLK,
         DAC => Kack_signal, TxD => TxD);

U1:      KeyDecode
         port map(MCLK => MCLK, RESET => RESET, Kack => Kack_signal, Kentry => Kentry, notDecOut => notDecOut,
         Kval => DAV_signal, K => D_signal);

end Structural;
```

D. Código *Kotlin – HAL*

```kotlin
object HAL {
    var lastOutput = 0
    fun init() {
        UsbPort.write(0)
    }

    fun isBit(mask: Int): Boolean {
    val value = UsbPort.read()
    val final = value and mask
    if (final != 0) return true else return false
    }

    fun readBits(mask: Int): Int {
        val value = UsbPort.read()
        val final = value and mask
        return final
    }

    fun writeBits(mask: Int, value: Int) {
        var finalOutput = 0
        finalOutput = (mask.inv() and lastOutput) or (value and mask)
        lastOutput = finalOutput
        UsbPort.write(finalOutput)
    }

    fun setBits(mask: Int) {
        var finalOutput = 0
        finalOutput = lastOutput or mask
        lastOutput = finalOutput
        UsbPort.write(finalOutput)
    }

    fun clrBits(mask: Int) {
        var finalOutput = 0
        finalOutput = mask.inv() and lastOutput
        lastOutput = finalOutput
        UsbPort.write(finalOutput)
    }
}
```

A. Código *Kotlin – KBD*

```
object KBD { // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
    const val NONE = 0;
    // Inicia a classe
    fun init() {
    KeyReceiver.init()
    }

    // Implementa a interação série com o Key Transmitter, utilizado na 2ªfase do
projeto
    private fun getKeySerial(): Char {
        val key = KeyReceiver.rcv()
        return when(key) {
            0 -> '1'
            1 -> '4'
            2 -> '7'
            3 -> '*'
            4 -> '2'
            5 -> '5'
            6 -> '8'
            7 -> '0'
            8 -> '3'
            9 -> '6'
            10 -> '9'
            11 -> '#'
            else -> NONE.toChar()
        }
    }
    // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
    fun getKey(): Char {
      return getKeySerial()
    }
    // Retorna quando a tecla for premida ou NONE após decorrido 'timeout'
milisegundos.
    fun waitKey(timeout: Long): Char {
    val timeinit = Time.getTimeInMillis()
        while (Time.getTimeInMillis() - timeinit < timeout) {
            val key = getKeySerial()
            if (key != NONE.toChar()) return key
        }
        return NONE.toChar()
    }

}
```

A. Código *Kotlin – KeyReceiver*

```
object KeyReceiver { // Recebe trama do Keyboard Reader.
    private val TxD_MASK = 1 shl 5
    private val TxCLK_MASK = 1 shl 6

    // Inicia a classe
    fun init() {
        HAL.init()
    }
    // Recebe uma trama e retorna o código de uma tecla caso exista
    fun rcv(): Int {
        var result = 0
        if (!HAL.isBit(TxD_MASK)) {
            for (i in 0..5) {
                HAL.setBits(TxCLK_MASK)
                Thread.sleep(200)
                result = result or HAL.readBits(TxD_MASK).shl(i)
                HAL.clrBits(TxCLK_MASK)
                Thread.sleep(200)
            }
        if ((result and 1) == 0) return -1
            if ((result and (1 shl 5)) != 0) return -1
            val mask = 30
            result = result and mask
        }
        return result
    }

}
```