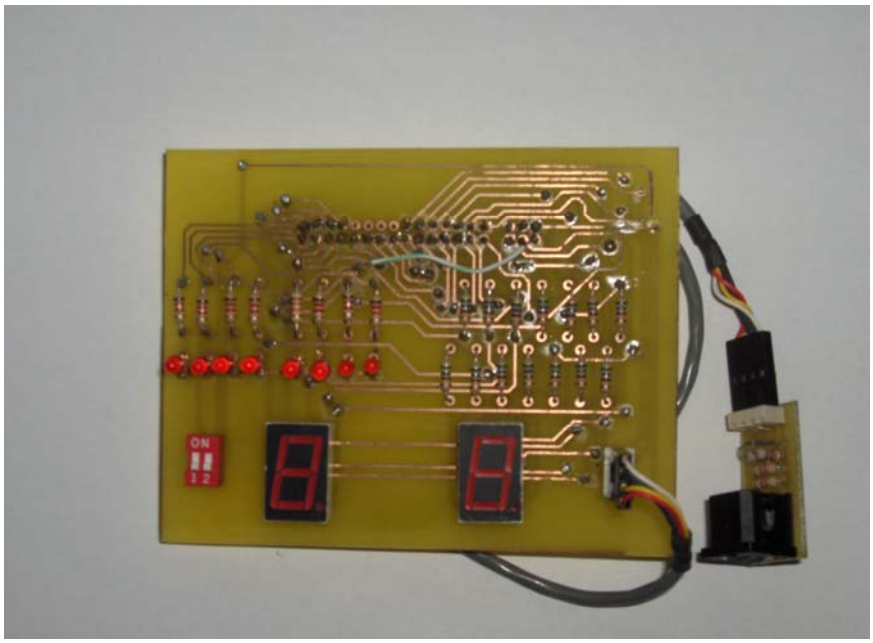
Sistemas Digitais Reconfiguráveis

Suporte de funcionamento e interacção com o teclado



Autor: André Amaral Costa (nº 27578)
Curso: Engenharia de Electrónica e Telecomunicações
Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Data: 26 de Dezembro de 2006
Disciplina: Sistemas Digitais Reconfiguráveis
Docente: Valery Sklyarov

Introdução

Neste trabalho desenvolveu-se hardware e software para interacção com um dispositivo de entrada (teclado) utilizando, para isso, o auxílio da placa DETIUA – S3 desenvolvida no departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

O hardware foi desenvolvido sob a forma de uma PCB, onde foram soldados posteriormente todos os componentes electrónicos, utilizando para o efeito o programa Orcad Release 9.1.

O desenvolvimento do software foi feito utilizando a linguagem VHDL no ambiente Xilinx ISE 8.2i.

Objectivos

O trabalho tem como objectivo a escrita de valores *ascii* ou *scan codes* em *led's* ou em *displays* de sete segmentos, conforme sejam pressionadas as teclas do teclado. A escolha do código a visualizar (*ascii* ou *scan code*) bem como a escolha do dispositivo de saída (*led* ou *display*) é feita recorrendo a dois *DIP switches* colocados no hardware desenvolvido no âmbito deste trabalho.

Descrição do trabalho

• Hardware

Neste trabalho desenvolveu-se uma *PCB* que irá ser conectada à placa DETIUA-S3 desenvolvida no departamento de Electrónica, Telecomunicações e Informática através de pinos acessíveis que fornecem ligações a uma FPGA da família Spartan 3.

A placa desenvolvida contém:

- Conector de 30 pinos (para ligar a placa DETIUA_S3);
- Conector de 6 pinos (para ligar a placa DETIUA_S3);
- Conector PS2 ;
- 2 DIP *switches*;
- 8 *led's* ;
- 2 *display's* de sete segmentos;

Pretende-se portanto ler valores do teclado, que estará ligado ao conector PS2, e escrever os valores lidos nos *led's* ou nos *display's* de sete segmentos. Os *DIP switches* permitem a escolha dos valores de saída (códigos *ascii* ou *scan codes*) e também a escolha do dispositivo de saída (*led's* ou *display's*).

Em anexo encontra-se o esquema eléctrico da placa (anexo 1) bem como o desenho da PCB (anexo 2 e 3) desenvolvida neste trabalho.

• Software

O software desenvolvido permite implementar as funcionalidades acima descritas. Este foi desenvolvido na linguagem VHDL no ambiente Xilinx ISE 8.2i webpack.

Este é constituído por 4 blocos distintos, cada um com uma funcionalidade diferente, como se pode ver no esquemático em anexo (anexo 4):

- DCM

Este bloco permite alterar a frequência do relógio de 80 MHz (disponível na placa DETIUA-S3) para 25 MHz e também criar um sinal de *reset* para o resto do circuito. Este bloco já se encontra disponível na biblioteca do Xilinx ISE 8.2i.

- Keyb

Este bloco permite ler o *scan code* do teclado quando uma tecla é pressionada.

Tem como entradas os sinais de dados e relógio do teclado (ps2_data e ps2_clk) bem como o relógio do sistema (clk25) e o sinal de *reset* (rst). Apresenta como saídas um sinal de 8 bits com o *scan code* (SCAN_code(7:0)) lido e um sinal que indica se o teclado está ocupado ou não (busy).

- Cod_converter

Neste bloco é feita a escolha do código de saída pretendido. Este código pode ser alterado actuando sobre o DIP *switch* 0. Tem como entradas o *scancode* lido do teclado (scan_code), o DIP *switch* 0 (dip0) e o sinal de busy do teclado. Este apresenta, na saída, o código já convertido (*ascii* ou *scancode*) num sinal de 8 bits chamado code.

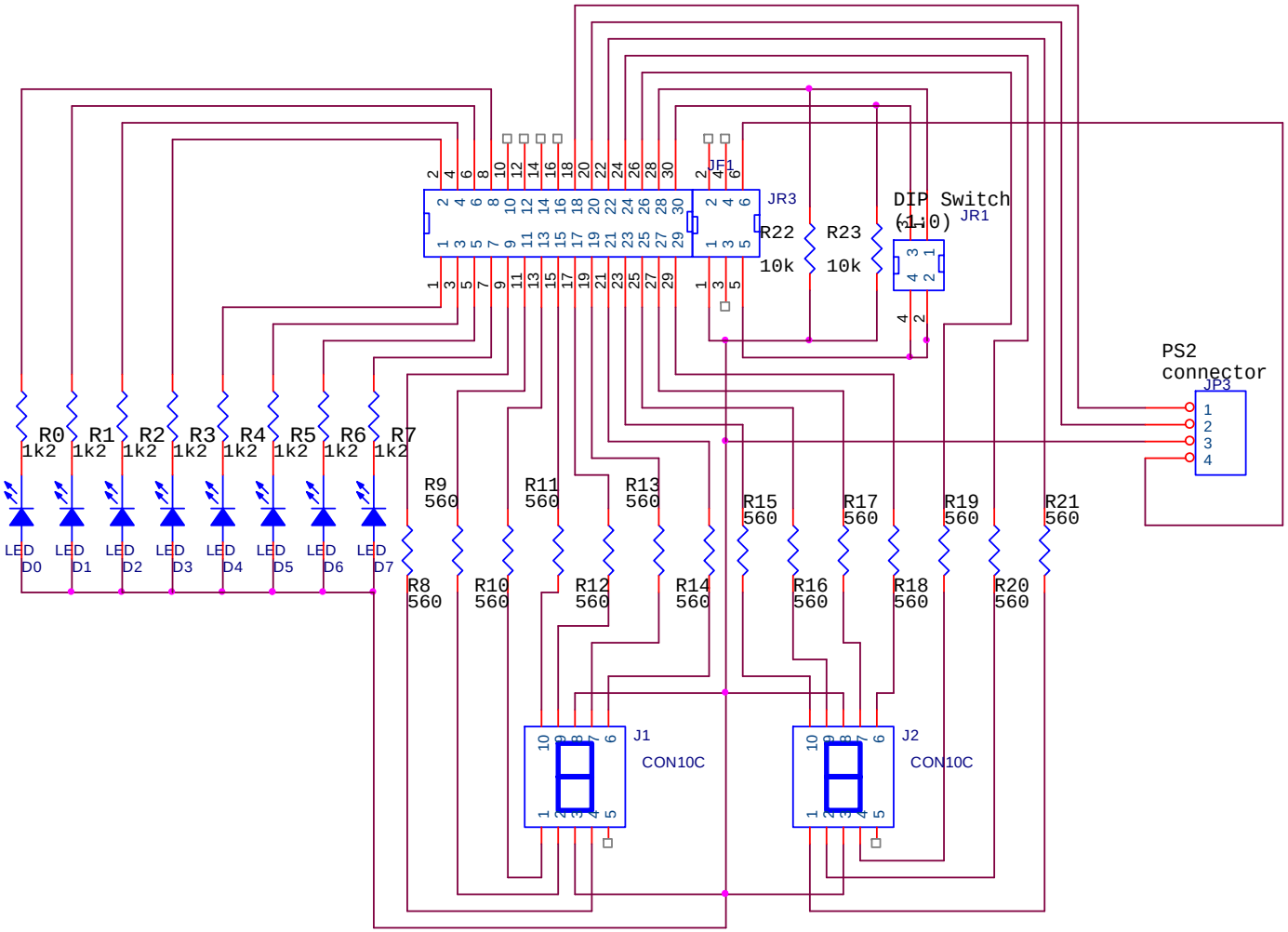
- Choose_out

A escolha do dispositivo de saída pretendido é feita neste bloco. Actuando sobre o DIP *switch* 1 pode-se optar por visualizar o código nos *led's* ou nos *display's* de sete segmentos. Este bloco tem como entradas o código convertido proveniente do bloco Cod_converter (code), o sinal de reset (rst), o sinal do DIP *switch* 1 (dip1). As saídas são dois vectores, um de 8 bits para os *led's* (led) e outros de 14 bits para os dois *display's* de sete segmentos (segment).

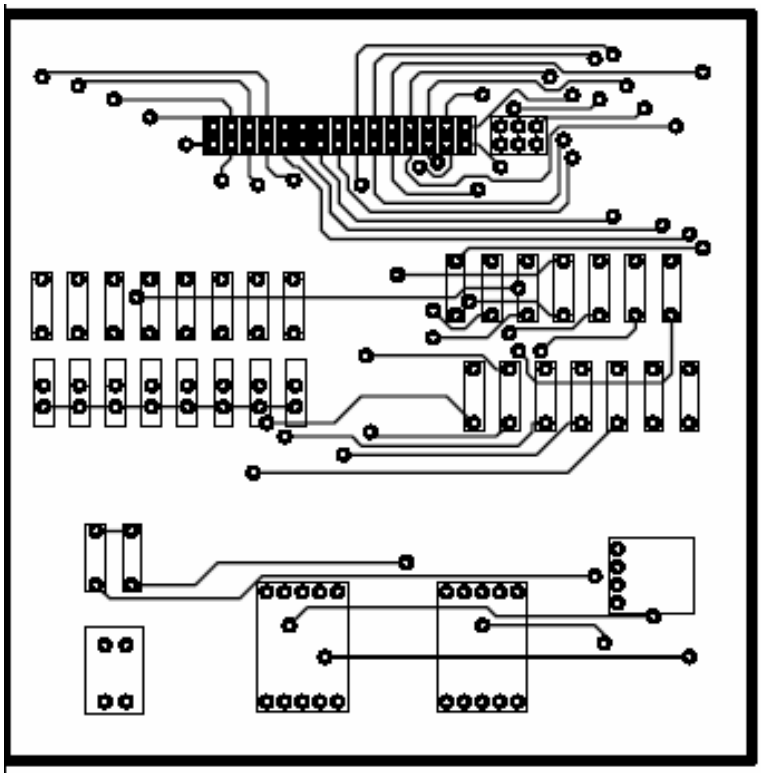
O código feito para estes blocos encontra-se em anexo (anexo 5,6,7).

Anexos

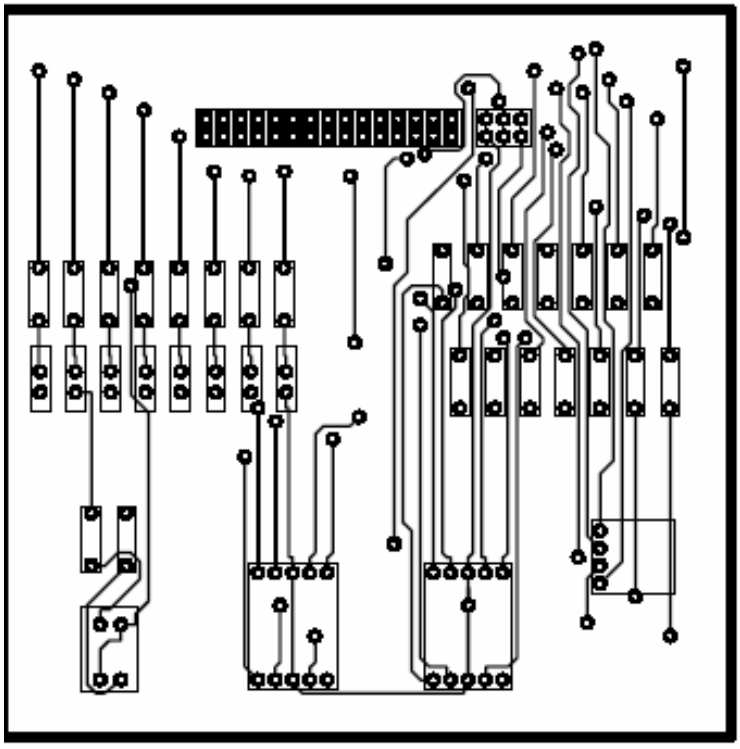
- **Anexo 1: Hardware - Esquema eléctrico da placa desenvolvida**



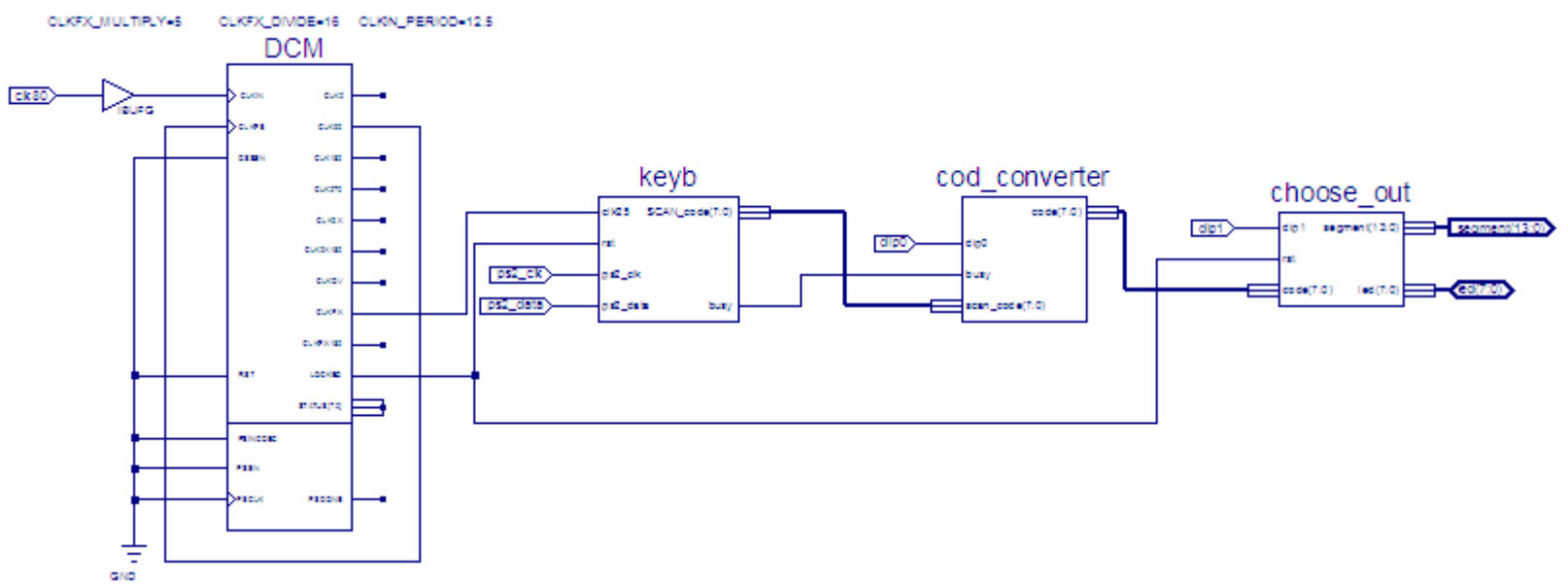
• Anexo 2: Hardware - Desenho da parte superior da PCB (TOP)



• Anexo 3: Hardware - Desenho da parte inferior da PCB (BOT)



- **Anexo 4: Software – Esquemático**



• Anexo 5: Software – Bloco keyb

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

-- Inputs / Outputs

entity keyb is
   Port ( clk25 : in  STD_LOGIC;  -- Relógio 25 MHz
          rst : in  STD_LOGIC;                -- Sinal de RESET
          ps2_clk : in  STD_LOGIC; -- Sinal de relógio do teclado
          ps2_data : in  STD_LOGIC;          -- Sinal de dados do teclado
          busy : inout  STD_LOGIC;-- Sinal de ocupação do teclado
          SCAN_code : out  STD_LOGIC_VECTOR (7 downto 0)); -- SCANCODE lido
end keyb;

architecture Behavioral of keyb is

          -- Internal signal

          signal improve_signal : std_logic_vector(7 downto 0);
          signal pulse : std_logic; -- Transição descendente do sinal de relógio                signal
count33 : natural range 0 to 32; -- Numero de bits lidos
          signal byte : std_logic_vector(7 downto 0); -- Scancode enviado

begin
          --Processo que detecta transições descendentes do relogio do teclado
          --Previne sentir glitches como dados

          process(clk25,rst)

          begin
                    if (rst='0') then
                              improve_signal<= (others=>'0');
                    elsif     rising_edge(clk25) then
                              improve_signal<= improve_signal(6 downto 0) & ps2_clk;
                              if improve_signal = "10000000" then
                                        pulse<= '1';
                              else pulse <= '0';
                              end if;
                    end if;
          end process;

          -- Processo que lê os scancodes quando alguma tecla é pressionada

          process(pulse, rst)
          begin
```

```
if (rst='0') then
        count33 <= 0;
        SCAN_code<= (others=>'0');
        busy<='0';
elsif rising_edge(pulse) then
        if (ps2_data ='0') then
                if (busy='0') then
                        busy<='1';
                        count33 <= count33 + 1; -- Start BIT
                else null;
                end if;
        else null;
        end if;

        case count33 is
                when 0 => null;

                when 1 to 8 =>
                        count33 <= count33 + 1;
                        byte(count33-1)<= ps2_data;

                when 9 =>
                        count33 <= count33 + 1;
                        SCAN_code <= byte;

                when 10 =>
                        if (ps2_data='1') then count33<= count33+1;
                        else null;
                        end if;

                when 11 =>
                        if (ps2_data ='0') then
                                count33 <= count33 + 1;
                        else null;
                        end if;

                when 12 to 20 =>
                        count33 <= count33 + 1;

                when 21 =>
                        if (ps2_data='1') then count33<= count33+1;
                        else null;
                        end if;

                when 22 =>
                        if (ps2_data ='0') then
                count33 <= count33 + 1;
                        else null;
                        end if;
```

```
                        when 23 to 31 =>
                                count33 <= count33 + 1;

                        when 32 =>
                                if (ps2_data='1') then
                                        count33<= count33+1;
                                        busy<='0';
                                        count33 <= 0;
                                else null;
                                end if;
                end case;
        end if;
    end process;
end Behavioral;
```

• Anexo 6: Software – Bloco Cod_converter

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

entity cod_converter is
   Port (
        dip0 : in  STD_LOGIC;
        code : out  STD_LOGIC_VECTOR (7 downto 0);
        busy:    in std_logic;                                  -- sincronizacao
        scan_code:in std_logic_vector(7 downto 0));       -- keyboard scan code

end cod_converter;

architecture Behavioral of cod_converter is
begin

process(scan_code,busy)
begin
        if busy/='1' then
                if dip0='1' then
                        case scan_code is
                                --Digitos
                                when x"45" => code <= x"30";
                                when x"16" => code <= x"31";
                                when x"1E" => code <= x"32";
                                when x"26" => code <= x"33";
                                when x"25" => code <= x"34";
                                when x"2E" => code <= x"35";
                                when x"36" => code <= x"36";
                                when x"3D" => code <= x"37";
                                when x"3E" => code <= x"38";
                                when x"46" => code <= x"39";

                                --Caracteres
                                when x"1C" => code <=x"41"; --A
                                when x"32" => code <=x"42"; --B
                                when x"21" => code <=x"43"; --C
                                when x"23" => code <=x"44"; --D
                                when x"24" => code <=x"45"; --E
                                when x"2B" => code <=x"46"; --F
                                when x"34" => code <=x"47"; --G
                                when x"33" => code <=x"48"; --H
                                when x"43" => code <=x"49"; --I
                                when x"3B" => code <=x"4A"; --J
                                when x"42" => code <=x"4B"; --K
                                when x"4B" => code <=x"4C"; --L
                                when x"3A" => code <=x"4D"; --M
```

```
                        when x"31" => code <=x"4E"; --N
                        when x"44" => code <=x"4F"; --O
                        when x"4D" => code <=x"50"; --P
                        when x"15" => code <=x"51"; --Q
                        when x"2D" => code <=x"52"; --R
                        when x"1B" => code <=x"53"; --S
                        when x"2C" => code <=x"54"; --T
                        when x"3C" => code <=x"55"; --U
                        when x"2A" => code <=x"56"; --V
                        when x"1D" => code <=x"57"; --W
                        when x"22" => code <=x"58"; --X
                        when x"35" => code <=x"59"; --Y
                        when x"1A" => code <=x"5A"; --Z
                        when x"66"      =>      code <= x"20"; -- BACKSPACE
                        when x"29"      =>      code <= x"20"; -- SPACE
                        when others     =>      null;
                end case;
        elsif dip0='0' then
                code<=scan_code;
        end if;
    end if;
end process;
end Behavioral;
```

• Anexo 7: Software – Bloco Choose_out

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

entity choose_out is
   Port ( code : in  STD_LOGIC_VECTOR (7 downto 0);
          dip1 :   in  STD_LOGIC;
          led :    inout  STD_LOGIC_VECTOR (7 downto 0);
          rst:     in std_logic;
          segment :        inout  STD_LOGIC_VECTOR (13 downto 0));

end choose_out;

architecture Behavioral of choose_out is

          signal code_msb: STD_LOGIC_VECTOR (3 downto 0);
          signal code_lsb: STD_LOGIC_VECTOR (3 downto 0);
begin

process (dip1)
begin
          if dip1='1' then
                    code_msb<=code(7 downto 4);
                    code_lsb<=code(3 downto 0);
                    case code_msb is
                              when "0000" => segment(13 downto 7)<= "1111110";
                              when "0001" => segment(13 downto 7)<= "0011000";
                              when "0010" => segment(13 downto 7)<= "1101101";
                              when "0011" => segment(13 downto 7)<= "0111101";
                              when "0100" => segment(13 downto 7)<= "0011011";
                              when "0101" => segment(13 downto 7)<= "0110111";
                              when "0110" => segment(13 downto 7)<= "1110111";
                              when "0111" => segment(13 downto 7)<= "0011100";
                              when "1000" => segment(13 downto 7)<= "1111111";
                              when "1001" => segment(13 downto 7)<= "0111111";
                              when "1010" => segment(13 downto 7)<= "1011111";
                              when "1011" => segment(13 downto 7)<= "1111111";
                              when "1100" => segment(13 downto 7)<= "1100110";
                              when "1101" => segment(13 downto 7)<= "1111110";
                              when "1110" => segment(13 downto 7)<= "1100111";
                              when "1111" => segment(13 downto 7)<= "1000111";
                              when others => null;
                    end case;
                    case code_lsb is
                              when "0000" => segment(6 downto 0)<= "1111110";
                              when "0001" => segment(6 downto 0)<= "0011000";
```

```
                when "0010" => segment(6 downto 0)<= "1101101";
                when "0011" => segment(6 downto 0)<= "0111101";
                when "0100" => segment(6 downto 0)<= "0011011";
                when "0101" => segment(6 downto 0)<= "0110111";
                when "0110" => segment(6 downto 0)<= "1110111";
                when "0111" => segment(6 downto 0)<= "0011100";
                when "1000" => segment(6 downto 0)<= "1111111";
                when "1001" => segment(6 downto 0)<= "0111111";
                when "1010" => segment(6 downto 0)<= "1011111";
                when "1011" => segment(6 downto 0)<= "1111111";
                when "1100" => segment(6 downto 0)<= "1100110";
                when "1101" => segment(6 downto 0)<= "1111110";
                when "1110" => segment(6 downto 0)<= "1100111";
                when "1111" => segment(6 downto 0)<= "1000111";
                when others => null;
        end case;
        led<="00000000";
    else
        led<=code;
        segment<="00000000000000";
    end if;
end process;
end Behavioral;
```