Circuitos no FPGA

Felipe H Gomes
José Maxwell
Marcelo Oliveira

Setembro 2017

1 Introdução

Nesse relatório é apresentado alguns circuitos implementados no FPGA pela linguagem VHDL. Usamos dos nossos conhecimentos que aprendemos nas ultimas aulas juntamente com as vídeos aulas propostas pelo nosso professor afim de entender melhor o uso da linguagem VHDL e o funcionamento da placa DE2-115.

2 Circuitos VHDL Implementados

2.1 PORTA NAND

O problema proposto nesse exercício é implementar um circuito referente a porta lógica **NAND**. A porta NAND recebe duas entradas binárias e retorna uma saída binária a partir da operação lógica " E". No circuito implementado é possível observar a operação a partir do LED.

```
library ieee;
use ieee.std_logic_1164.all;


--Entity

entity porta_nand is
   port
   (

      SW   : IN  STD_LOGIC_VECTOR(2 downto 0);
      LEDG : OUT STD_LOGIC_VECTOR(1 downto 0)
      );


end entity;

--Arquitetura
architecture comportamento of porta_nand is
begin

   LEDG(0) <= SW(1) nand SW(0);
end comportamento;
```

Figure 1: Código VHDL Circuito NAND

2.2 BIT VECTOR

Esse problema descreve um circuito que implementa três operações lógicas, **XOR OR AND**. As saídas resultantes é possível verificar a partir dos LEDs.

```
library ieee;
use ieee.std_logic_1164.all;

entity bit_vector_ports is
port(
   SW       :  in std_logic_vector (5 downto 0);
   LEDG     :  out std_logic_vector (2 downto 0)
   );
end entity;

architecture circuito of bit_vector_ports is
begin
   LEDG(0) <= SW(5) and SW(4);
   LEDG(1) <= SW(3) or SW(2);
   LEDG(2) <= SW(1) xor SW(0);
end architecture;
```

Figure 2: Código VHDL Circuito Bit Vector

2.3 MUX

Esse circuito é implementado um multiplexador que controla o uso das chaves, é definido duas chaves como chaves de controle onde irão ser responsáveis pela ativação dos demais LEDs. As chaves 16 e 17 são definidas como chaves de controle. Há quatro combinações possíveis :

- **"11"** Apenas os LEDs 15,14,13,12 estão liberadas e as demais bloqueadas.
- **"10"** Apenas os LEDs 11,10,09,08 estão liberadas e as demais bloqueadas.
- **"01"** Apenas os LEDs 07,06,05,04 estão liberadas e as demais bloqueadas.
- **"00"** Apenas os LEDs 03,02,01,00 estão liberadas e as demais bloqueadas.

```
library ieee;
Use ieee.std_logic_1164.all;

entity mux is
port (
   SW    :  in std_logic_vector (17 downto 0);
   LEDG  :  out std_logic_vector (3 downto 0)
   );
end entity;

architecture circuito of mux is
begin
   LEDG <=  SW (15 downto 12) when SW (17 downto 16) = "11" else
            SW (11 downto 8)  when SW (17 downto 16) = "10" else
            SW (7  downto 4)  when SW (17 downto 16) = "01" else
            SW (3  downto 0);
end architecture;
```

Figure 3: Código VHDL Circuito Mux

2.4 MUX 2

Nesse exercício foi feito um multiplexador que controla quatro chaves, sendo elas representadas pelos seus respectivos números binários. E é controlada por duas chaves de controles que a partir das combinações abaixo possibilita o controle do LED.

- **"00"** Apenas a chave 0 controla o LED.
- **"01"** Apenas a chave 1 controla o LED.
- **"10"** Apenas a chave 2 controla o LED.
- **"11"** Apenas a chave 3 controla o LED.

```
library ieee;
Use ieee.std_logic_1164.all;

entity mux2 is
port(
   SW : in std_logic_vector (5 downto 0);
   LEDG : out std_logic_vector(1 downto 0));
end entity;

architecture hardware of mux2 is
begin
   with SW (5 downto 4) select LEDG(0) <= SW(0) when "00" ,
                                          SW(1) when "01"  ,
                                          SW(2) when "10"  ,
                                          SW(3) when others;
end hardware;
```

Figure 4: Código VHDL Circuito Mux2

2.5 Paridade

Nesse circuito foi definido um LED como saída, se tiver um número par de chaves ligadas o LED ficará apagado, se houver um número impar de chaves ligadas o LED ficará aceso.

```
library ieee;
Use ieee.std_logic_1164.all;

entity paridade_par is
port(
   SW :  in std_logic_vector(0 to 4);
   LEDG  :  out std_logic_vector(1 downto 0)
);

end entity;

architecture hardware of paridade_par is

   signal aux : std_logic_vector(0 to 3);
   begin
   aux(0) <= SW(0) xor SW(1);

   Gen: for i in 1 to 3 generate
      aux(i) <= SW(i + 1) xor aux(i - 1);
   end generate Gen;

   LEDG(0) <= aux(3);
end architecture;
```

Figure 5: Código VHDL Circuito Paridade

2.6 Somador

Esse circuito é implementado um somador binário, onde recebe três entradas e executa a operação de soma dos bits. Há dois LEDs de saída que resultam nas operações possíveis.

```
library ieee;
use ieee.std_logic_1164.all;

entity somador_completo is
port(

   SW       :  in std_logic_vector (0 to 2);
   LEDG     :  out std_logic_vector (0 to 1) );

end entity;

architecture logica of somador_completo is

   signal temp : std_logic;

begin

   temp <= sw(0) xor sw(1);
   LEDG(0) <= temp xor sw(2);
   LEDG(1) <= (sw(2) and temp) or (sw(0) and sw(1));

end logica;
```

Figure 6: Código VHDL Circuito Somador

3 Conclusão

A partir dessa atividades prática que tivemos, podemos ter um entendimento sobre as funcionalidades da placa DE2-115, tivemos um boa introdução referente á linguagem VHDL e como o FPGA é uma poderosa ferramenta para o desenvolvimento de circuitos.