Insper

Elementos de Sistema - Aula 5 - Handout - Lógica Combinacional

Rafael Corsi - rafael.corsi@insper.edu.br

Março - 2018

Quartus

Esse handout introdutório para o desenvolvimento do projeto C-Lógica-Combinacional, aqui iremos fazer nosso primeiro código em VHDL e vamos programar a FPGA para executar o hardware recém descrito.

Após essa etapa, iremos começar o desenvolvimento do projeto, programando os módulos que virão a ser utilizados no computador Z01.

Entendendo a estrutura de pastas

A pasta do projeto C no repositório Z01 possui a seguinte estrutura :

```
/C-Logica-combinacional
    /Quartus
    /scripts
        testeLogicaCombinacional.py
    /src
        /rtl
            *.vhd
    /tests
        /tst
            *.vhd
```

1. Quartus : Projeto Quartus que faz uso dos arquivos VHDL localizados em src/rtl/*.vhd
2. scripts : Scripts em python que automatizam a execução dos testes
3. src/rtl/*.vhd : Arquivos VHDL que serão implementado pelo grupo
4. tests/tst/*.vhd : Arquivos VHDL que realizam teste lógico nos arquivos do rtl

Executando o script de teste

Abra o terminal na pasta *C-Logica-Combinacional/script* e execute o script python localizado nessa pasta :

```
$ python3 testeLogicaCombinacional.py
```

O mesmo irá executar a compilação dos arquivos src/rtl/*.vhd e executar os testes unitários em cada um desses módulos. Como os módulos não estão implementados o resultado deve ser :

```
==== Summary ==================================
pass lib.tb_nand.all             (3.1 seconds)
fail lib.tb_dmux4way.all         (3.3 seconds)
fail lib.tb_or8way.all           (3.3 seconds)
fail lib.tb_dmux2way.all         (3.3 seconds)
fail lib.tb_mux2way.all          (0.5 seconds)
fail lib.tb_dmux8way.all         (1.7 seconds)
fail lib.tb_mux4way.all          (1.7 seconds)
fail lib.tb_barrelshifter16.all (1.7 seconds)
fail lib.tb_or16.all             (1.8 seconds)
fail lib.tb_mux4way16.all        (1.4 seconds)
fail lib.tb_mux8way16.all        (1.6 seconds)
fail lib.tb_not16.all            (1.5 seconds)
fail lib.tb_nor8way.all          (1.5 seconds)
fail lib.tb_and16.all            (1.9 seconds)
fail lib.tb_barrelshifter8.all   (1.7 seconds)
fail lib.tb_mux8way.all          (1.8 seconds)
fail lib.tb_mux16.all            (1.8 seconds)
================================================
pass 1 of 17
fail 16 of 17
================================================
Total time was 33.8 seconds
Elapsed time was 8.7 seconds
================================================
Some failed!
➜  script git:(master) ✗
```

Figure 1: Script teste

Abrindo o Quartus

Abra o quartus e *File->Open Project->* escolha o projeto localizado na pasta C-Logica-Combinacional/Quartus.

O arquivo que o Quartus irá reconhecer é o : *DE0_CV_Default.qpf*, como no gif a seguir :

Abra arquivo TopLevel.vhd como demonstrado no gif anterior, esse arquivo é o que chamamos de top level (pode-se fazer uma analogia com o main), ele será o primeiro a ser executado na compilação e utilizará os demais módulos.

Um código em VHDL possui basicamente três partes:

1. Declaração de bibliotecas utilizadas

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

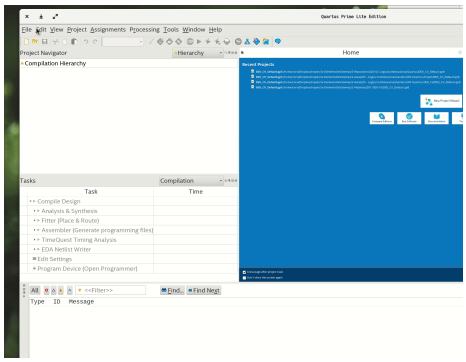

Figure 2: Abrindo o Quartus

2. Declaração das entradas e saídas desse bloco (entidade)

“‘

2. Declaração das entradas e saídas desse bloco (entidade)

```
entity TopLevel is
    port(
        SW      : in  std_logic_vector(9 downto 0);
        LEDR    : out std_logic_vector(9 downto 0)
    );
end entity;
```

3. Implementação da lógica que relaciona as entradas e saídas do módulo (arquitetura)

```
architecture rtl of TopLevel is

begin


end rtl;
```

O código original disponível não realiza nenhuma lógica, sua arquitetura está vazia.

Compilando o código

Para compilarmos esse código VHDL basta irmos em :

Processing -> Start Compilation

A ferramenta irá “realizar” o código, ou seja, interpretar e torna-lo um hardware.

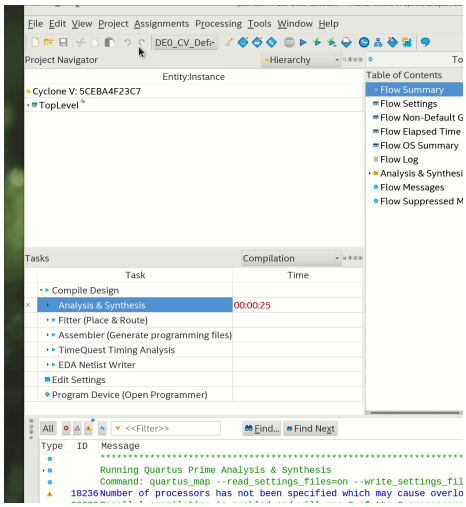

Figure 3: Compilando

RTL View

Podemos gerar a visão rtl do código em vhdl, esse diagrama é a interpretação do código em vhdl pelo compilador e como ele seria supostamente implementando em hardware.

Para isso :

Tools -> Netlist Viewers -> RTL viewer

Ele irá gerar o seguinte diagrama

Figure 4: Compilando

Onde podemos analisar que não existe nenhuma lógica que relaciona entrada a saída.

Modificando o projeto

Vamos modificar o projeto e inserir para :

```
---------------
-- implementacao
---------------
begin

  LEDR(0) <= SW(0);

end rtl;
```

1. Compile
2. Gere o RTL Viewer novamente

O resultado deve ser o seguinte :

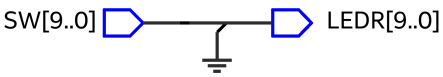

Figure 5: Compilando

Onde o valor do LEDR0 será o próprio valor de entrada chave SW0.

Programando a FPGA

Para programar na FPGA conecte-a ao seu computador via cabo USB e vá em :

Tools -> Programmer

Ele deve abrir uma nova interface :

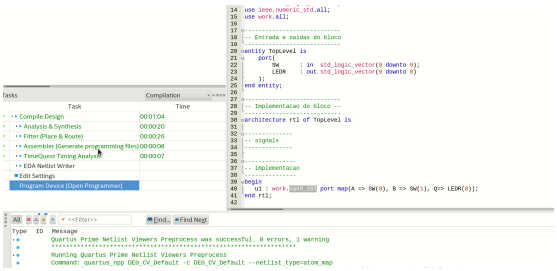

Figure 6: Programando

Desafios

Para cada desafio proposta a seguir, verifique se o RTL corresponde a lógica que deseja implementar, após isso teste na FPGA :

- Compile
- Verifique o RTL
- Programe a FPGA

1

Faça a saída LEDR(0) ser o inverso da entrada SW(0)

2

Faça a saída LEDR(0) ser a entrada SW(0) ou SW(1)

3

Faça a saída LEDR(0) ser a entrada SW(0) ou SW(1) e o LEDR(1) ser a chave SW(1)

4

Faça cada LED da placa acompanhar sua respectiva chave, exe. LEDR(0) recebe a chave SW(0) assim por diante até o último LED(9).
Existe uma forma fácil de fazer isso ?

Adicionando um novo componente ao projeto

O desenvolvimento de projetos de hardware assim como os de softwares devem ser feitos de forma modular, onde especifica-se e implementa-se pequenos módulos (entidades) que são combinadas em sistemas cada vez mais complexos até chegar ao **TopLevel**.

Esse projeto define uma série de pequenos módulos, cada um com sua especificidade (localizados em C-LogicaCombinacional/src/rtl/) :

- And16.vhd
- BarrelShifter16.vhd
- BarrelShifter8.vhd
- DMux2Way.vhd
- DMux4Way.vhd
- DMux8Way.vhd

- Mux16.vhd
- Mux2Way.vhd
- Mux4Way16.vhd
- Mux4Way.vhd
- Mux8Way16.vhd
- Mux8Way.vhd
- **Nand.vhd**
- Nor8Way.vhd
- Not16.vhd
- Or16.vhd
- Or8Way.vhd
- TopLevel.vhd

O módulo **nand.vhd** já foi dado feito para vocês, e seu conteúdo é o seguinte :

```
Library ieee;
use ieee.std_logic_1164.all;

entity nand_z01 is
   port(
        A : in  std_logic;
        B : in  std_logic;
        Q : out std_logic
   );
end nand_z01;

architecture rtl of nand_z01 is
begin
   q <= not (a and b);
end rtl;
```

> **!** Note que o VHDL não é case sensitive

É uma entidade que possui duas entrada (a,b) e uma saída (q) e implementa um nand entre as entradas.

Inserindo no toplevel

Podemos inserir essa nand no toplevel da seguinte maneira :

```
begin

  u1 : work.nand_z01 port map(A => SW(0), B => SW(1), Q=> LEDR(0));

end rtl;
```

Essa linha de código pode ser lida como :

Cria um componente chamada de **uq**, mapeando a entrada SW(0) na porta A da nand, a entrada SW(1) na entrada B da NAND e a saída Q da NAND é mapeada para o LEDR(0)

- Compilando podemos analisar o RTL gerado

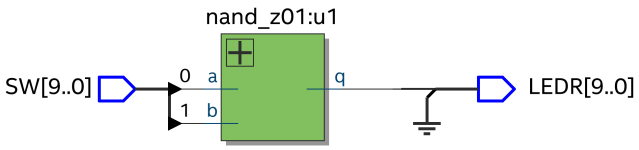

Figure 7: RTL NAND TopLevel

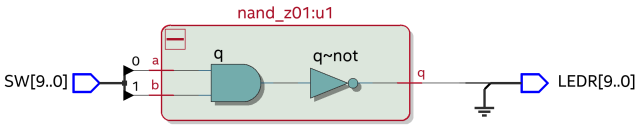

Figure 8: RTL NAND Expandido

Desafios extras

5

Faça TODOs os LEDs acenderem quando a seguinte combinação de entrada for :

```
SW9                 SW0
 1 0 0 1 1 0 1 0 1 0
```

6

Faça um código VHDL do circuito a seguir :