Insper

Elementos de Sistema - Projeto 3 - Lógica Combinacional

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

Agosto - 2017

Prazo de entrega : 30/8/2017

Descrição

Esse projeto tem como objetivo trabalhar com portas lógicas e sistemas digitais combinacionais (sem um clock) em FPGA e VHDL. Os elementos lógicos desenvolvidos nessa etapa serão utilizados como elementos básicos para a construção do computador.

O desenvolvimento será na linguagem VHDL com o git. Os alunos deverão se organizar para implementar todos os elementos propostos. O facilitador escolhido será responsável pela completude e consistência do branch master do fork do grupo.

Integrantes

Este projeto é para ser realizado por todos os integrantes do grupo. Tarefas devem ser criadas no Trello e alunos poderão executar elas em duplas, pratique pair-programming, porém evite sempre trabalhar com a mesma dupla, ou seja, cada projeto deverá ser realizado por uma nova dupla. Se organizem para tentar desenvolver o projeto de forma que todos aprendam o máximo possível, embora só um integrante de uma dupla possa fazer os commits para o github. Converse com os colegas a respeito do projeto, mas não aceite código pronto.

Controle de Tarefas e Repositório

Atualize seu Fork do repositório do projeto do github:

- https://github.com/Insper/Z0

Nas discussões com os outros alunos, escolha alguns módulos para desenvolver. Defina o desenvolvimento do módulo no Trello. Crie uma rotina para commits e testes dos módulos e verifique se está fazendo o esperado:

- https://trello.com/engcompinsper2017

Instruções trabalho em equipe

Cada dupla deve desenvolver pelo menos duas entidades, não podendo as duas serem do mesmo tipo. Após validada a entidade, submeta um pullquest para que o facilitador do projeto aprove e faça o merge no branch master.

Intruções

A pasta contém dois diretórios distintos : src/ e Quartus/. O diretório src contém os arquivos fontes que deverão ser editados para implementar o projeto. O diretório quartus/ contém o projeto que possibilitará compilar os módulos e testar em hardware.

src

Deve-se implementar os seguintes circuitos combinacionais :

- AND 16 bits
  - **Arquivo** : And16.vhd
  - **Descrição** : And bit a bit entre duas palavras de 16 bits.
- OR de 16 bits
  - **Arquivo** : Or16.vhd
  - **Descrição** : OR bit a bit entre duas palavras de 16 bits.
- NOT de 16 bits
  - **Arquivo** : Not16.vhd
  - **Descrição** : NOT bit a bit entre duas palavras de 16 bits.
- NOR 8 Way
  - **Arquivo** : Nor8Way.vhd
  - **Descrição** : NOR entre 8 bits, resulta em uma única saída
- OR 8 Way
  - **Arquivo** : Or8Way.vhd
  - **Descrição** : OR entre 8 bits, resulta em uma única saída
- Barrel Shifter 8 bits
  - **Arquivo** : BarrelShifter8.vhd
  - **Descrição** : Shifta um vetor de 8 bits para a direita e para esquerda.

- Barrel Shifter 16 bits
  - **Arquivo** : BarrelShifter16.vhd
  - **Descrição** : Shifta um vetor de 16 bits para a direita e para esquerda.
- Demultiplexiador de 2 saídas
  - **Arquivo** : DMux2Way.vhd
  - **Descrição** : Demultiplexa uma entrada binária em duas saídas.
- Demultiplexiador de 4 saídas
  - **Arquivo** : DMux4Way.vhd
  - **Descrição** : Demultiplexa uma entrada binária em quatro saídas.
- Demultiplexiador de 8 saídas
  - **Arquivo** : DMux8Way.v8d
  - **Descrição** : Demultiplexa uma entrada binária em oito saídas.
- Demultiplexiador de 8 saídas para um vetor de 16 bits de entrada
  - **Arquivo** : DMux2Way.vhd
  - **Descrição** : Demultiplexa uma entrada de 16 bits em oito saídas de 16 bits cada.
- Multiplexador 2 entradas de um bit cada
  - **Arquivo** : Mux2Way.vhd
  - **Descrição** : Multiplexa 2 entradas binárias em uma saída binária
- Multiplexador 4 entradas de um bit cada
  - **Arquivo** : Mux4Way.vhd
  - **Descrição** : Multiplexa 4 entradas binárias em uma saída binária
- Multiplexador 8 entradas de um bit cada
  - **Arquivo** : Mux8Way.vhd
  - **Descrição** : Multiplexa 8 entradas binárias em uma saída binária
- Multiplexador 16 entradas de um bit cada
  - **Arquivo** : Mux16Way.vhd
  - **Descrição** : Multiplexa 16 entradas binárias em uma saída binária
- Multiplexador 4 entradas de 16 bits cada
  - **Arquivo** : Mux4Way16.vhd
  - **Descrição** : Multiplexa 4 entradas de 16 bits cada em uma saída de 16 bits.
- Multiplexador 8 entradas de 16 bits cada
  - **Arquivo** : Mux8Way16.vhd
  - **Descrição** : Multiplexa 8 entradas de 16 bits cada em uma saída de 16 bits.

Testando em SW

Para executarmos uma unidade de teste em cada módulo, executamos o comando

```
python testes.py
```

Esse comando executa um teste unitário em cada um dos módulos, verificando se sua implementação está correta. O resultado é exibido na tela como : **pass** ou **fail**.

Testando em HW

Para testar os módulos em hardware, deve abrir o projeto (03-Logica-Combinacional/quartus/03-Logica-Combinacional.qpf) e testar individualmente cada módulo.

Para selecionar o módulo que deseja testar, basta alterar de *False* para *True* sua referência no **generic** da *entity*, compilar o projeto e programar.

```
generic(
    -- Logic
    TEST_NAND       : boolean := False;     -- Nand.vhd;
    TEST_AND16      : boolean := False;     -- And16.vhd
    TEST_OR16       : boolean := False;     -- Or16.vhd
    TEST_NOT16      : boolean := False;     -- Not16.vhd
    TEST_OR8WAY     : boolean := False;     -- Or8Way.vhd
    TEST_NOR8WAY    : boolean := False;     -- Nor8Way.vhd
    -- BarrelShifter
    TEST_BS8        : boolean := False;     -- BarrelShiter8.vhd
    TEST_BS16       : boolean := False;     -- BarrelShiter16.vhd
    -- Dmux
    TEST_DMUX2      : boolean := False;     -- DMux2Way.vhd
    TEST_DMUX4      : boolean := False;     -- Dmux4Way.vhd
    TEST_DMUX8      : boolean := False;     -- Dmux8Way.vhd
    TEST_DMUX8WAY16 : boolean := False;     -- Dmux8Way16.vhd
    -- Mux
    TEST_MUX2WAY    : boolean := False;     -- Mux2Way.vhd
    TEST_MUX4WAY    : boolean := False;     -- Mux4Way.vhd
    TEST_MUX8WAy    : boolean := False;     -- Mux8Way.vhd
    TEST_MUX16      : boolean := false;     -- Mux16.vhd
    TEST_MUX4WAY16  : boolean := False;     -- Mux4Way16.vhd
    TEST_MUX8WAY16  : boolean := False      -- Mux8Way16.vhd
  );
```

Para testarmos por exemplo o módulo Or16.vhd, devemos inserir True na linha :

```
    TEST_OR16       : boolean := True;     -- Or16.vhd
```

Alerta: Apenas um módulo pode ser testado por vez !

Cada módulo quando carregado na FPGA utilizará os botões e LEDs de uma maneira específica. Consulte a página : 03-Testes-HW-Logica-Combinacional para maiores de talhes de como cada módulo utiliza os I/Os.

DICAS

Algumas dicas para o projeto :

AND16.vhd

A seguir a implementação da And16.vhd

```
-- Elementos de Sistemas
-- by Luciano Soares
-- And16.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity And16 is
    port (
            a:   in  STD_LOGIC_VECTOR(15 downto 0);
            b:   in  STD_LOGIC_VECTOR(15 downto 0);
            q:   out STD_LOGIC_VECTOR(15 downto 0)
        );
end entity;

architecture arch of And16 is
begin
    q <= a and b;
end architecture;
```

Avaliação :

Itens necessários para o aceite

- Implementar em grupo todos os módulos funcionando.

Validação

- Demonstrar em hardware duas implementações escolhidas pelo Professor.
- Teste no :

Rubrica

Nota máxima	Descritivo
A	- Entregue no prazo - Todos os itens corretos - Implementações ótimas (otimizados)

Nota máxima	Descritivo
B	- Entregue no prazo
	- Todos os itens corretos
C	- Entregue fora do prazo, ou
D	- 80% dos itens entregues corretamente
I	- Não entregue
	- Menos que 80% dos itens corretos