Insper

Elementos de Sistema - Projeto D - Unidade Lógica Aritmética

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

Março - 2018

> **!** Prazo original : Quarta Feira - 20/3/2018
> Atrasado até : Quarta Feira - 27/3/2018

Descrição

Neste projeto você terá que desenvolver os componentes para a implementação de uma unidade lógica e aritmética, completa, de 16 bit (proposta pelo livro texto). Os alunos deverão subdividir a ALU em módulos e duplas de alunos desenvolverão cada parte. O facilitador escolhido será responsável pela completude e consistência do branch master.

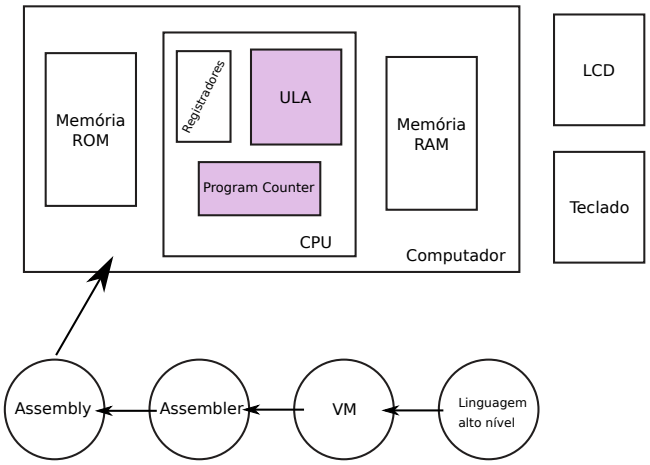

Figure 1: ULA

Integrantes

Este projeto deve ser realizado por todos os integrantes do grupo. As tarefas devem ser criadas no Kanban e os alunos poderão executá-las em duplas, praticando o pair-programming. Porém, evite sempre trabalhar na 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.

Controle de Tarefas e Repositório

Atualize o seu Fork do projeto com o repositório do github:

- https://github.com/Insper/Z01

Nas discussões com o grupo, o scrum master deverá definir os módulos que cada integrante irá desenvolver. Crie uma rotina para commits e pull-requests. Sempre teste os módulos e verifique se está funcionando como o esperado.

Funções do Facilitador (Scrum Master)

- Organizar o github + issues + project;
- Fazer a atualização do fork com o upstream;
- Gerenciar o grupo (atribuir tarefas);
- Verificar e aceitar os pull-requests;
- Fazer o relatório de desempenho de cada integrante do grupo;
- Entregar/Apresentar o projeto no final.

Funções dos Desenvolvedores

- Realizar as tarefas atribuídas pelo scrum-master;
- Realizar os pull-requests;
- Testar os códigos.

Instruções

Entendendo a Organização do Projeto

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

```
/D-ULA
    /Quartus
    /script
        testeULA.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 implementados pelo grupo;
4. tests/tst/*.vhd: Arquivos VHDL que realizam o teste lógico nos arquivos do rtl.

Executando o Script de Teste

Abra o terminal na pasta *D-UnidadeLogicaAritmetica/script* e execute o script python localizado nessa pasta:

```
$ python testeULA.py
```

O mesmo irá compilar os arquivos src/rtl/*.vhd e executar os testes unitários em cada um deles. Nesse momento do teste, como os módulos não estão implementados, o resultado deverá ser falho.

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**.

O que deve ser feito:

Além de implementar os módulos, deve-se gerar uma imagem com a forma de onda desse módulo. Agora somente os módulos que são selecionados para testar serão testados. Para cada nova implementação deve-se criar um novo branch e descomentar somente o módulo que está sendo implementado.

Note que é possível reaproveitar, via port map, os módulos do projeto anterior (C). Esses módulos anteriores, já são incluídos, automaticamente (pelo script), na compilação dos módulos do projeto D.

Módulos

> **!** Esses arquivos estão localizados em D-UnidadeLogicaAritmetica/src/rtl/

Deve-se implementar os seguintes circuitos combinacionais :

- HalfAdder
  - **Arquivo** : HalfAdder.vhd
  - **Descrição** : Adiciona dois bits que resulta em um bit de soma e outro de carry out.
- FullAdder
  - **Arquivo** : FullAdder.vhd
  - **Descrição** : Adiciona três bits, dois referentes às entradas e o outro referente ao carry in. O resultado é um bit com a soma e outro com o carry out.
- Add16
  - **Arquivo** : Add16.vhd
  - **Descrição** : Adiciona dois vetores de 16 bits resultando em um vetor de 16 bits (sem carry out do bit mais significativo - MSB).
- Inc16
  - **Arquivo** : Inc16.vhd
  - **Descrição** : Adiciona '1' a um vetor de 16 bits resultando em um vetor de 16 bits (sem carry out do MSB).
- Inversor16
  - **Arquivo** : Inversor16.vhd
  - **Descrição** : Inverte um vetor de entrada quando o bit de controle **n** (nx ou ny) for igual a '1', e não modifica o vetor de entrada caso contrário. O resultado é um novo vetor de 16 bits.
- Zerador16
  - **Arquivo** : Zerador16.vhd
  - **Descrição** : Zera um vetor de entrada quando o bit de controle **z** (zx ou zy) for igual a '1'. Não modifica o vetor de entrada se o bit for '0'. O resultado é um novo vetor de 16 bits.
- Comparador16
  - **Arquivo** : Comparador16.vhd
  - **Descrição** : Verifica se o vetor de saída (16 bits) é igual a zero (**zr**) e se menor que Zero (**ng**). Caso igual a zero, faz com que o sinal **zr** seja igual a '1' e caso contrário '0'. Se o sinal de entrada for negativo faz com que **ng** receba '1' e '0' caso contrário.

Pseudo código :

```
if(a == 0)
  zr = 1
else
  zr = 0

if (a < 0)
  ng = 1
else
  ng = 0
```

- ALU

- **Arquivo** : ALU.vhd
- **Descrição** : A entidade que faz o mapeamento de todas as demais, interligando os blocos (zerador, comparador, inversor, Add ....) em um único bloco.

Para implementar a ALU será necessário usar os blocos desenvolvidos neste projeto e os blocos desenvolvidos no projeto anterior: And16, Mux16. O script de compilação e teste já faz a inclusão deles. A arquitetura da ULA pode ser vista abaixo:

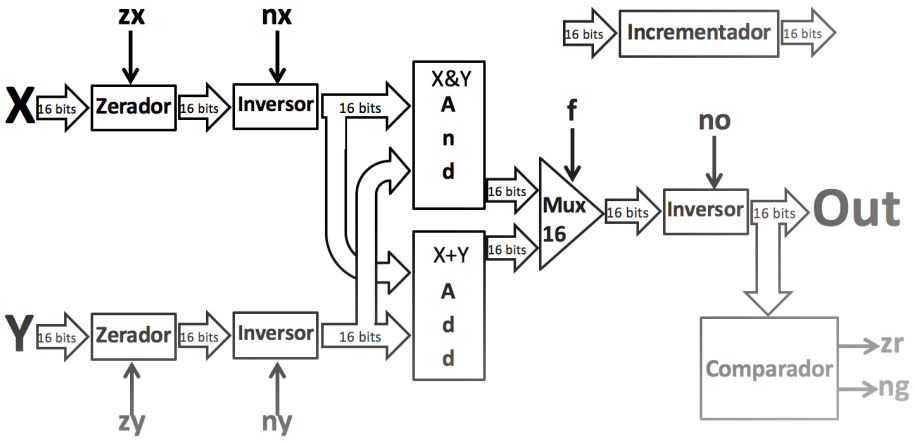

Figure 2: ULA

Forma de onda

Para cada teste realizado, deve-se carregar a interface gráfica e tirar um print da forma de onda do módulo com os testes aplicados a ele (D-Ferramental.pdf). Essa imagem deve ser salva na mesma pasta dos arquivos vhdl (src/rtl/) e com o mesmo nome dos módulos. A pasta no final do projeto deve possuir os seguintes arquivos:

```
/src/rtl/
        Add16.vhd
        Add16.png
        ALU.vhd
        ALU.png
        Comparador16.vhd
        Comparador16.png
        FullAdder.vhd
        FullAdder.png
```

```
HalfAdder.vhd
HalfAdder.png
Inc16.vhd
Inc16.png
Inversor16.vhd
Inversor16.png
Zerador16.vhd
Zerador16.png
```

Testando em HW

Para testar os módulos em hardware, deve-se abrir o projeto (D-UnidadeLogicaAritmetica/Quartus). Ele já inclui todos os módulos desta entrega e também os módulos da entrega passada. O arquivo localizado em /rtl/toplevel.vhd já faz o mapeamento dos pinos da FPGA para os pinos da ULA, assim como o handout da aula 7, para testar no hardware basta compilar e programar a FPGA.

Rubricas para avaliação de projetos

Cada integrante do grupo irá receber duas notas: uma referente ao desenvolvimento total do projeto (Projeto) e outra referente a sua participação individual no grupo (que depende do seu papel).

Projeto

Conceito	
I	- Não implementou os módulos Add16, ULA, Comparador, FullAdder, HalfAdder, Inc16, Inversosr, Zerador.
D	- Implementou todos os módulos menos a ULA.
C	- Configurou o travis e funcionou. - Todos os testes passaram (ok). - ULA funcionando. - Todos os módulos sendo testados no Travis. - Possui a forma de onda de todos os módulos (.png).
B	- Gravou a ULA na FPGA e fez um vídeo demonstrando o seu funcionamento.
A	- Usou port map para reaproveitar outros módulos, por exemplo: - usou o full-adder para criar o add16, usou na ULA o mux do projeto C.

Desenvolvedor

Conceito	
I	- Se comprometeu a fazer algum desenvolvimento (kanban) e não fez.
D	- Criou o branch mas não fez o desenvolvimento completo.
C	- Desenvolveu as rotinas para passarem nos testes! - Acompanhou o Kanban board (github project). Ex: Puxou tarefas, etc. - Respeitou a arquitetura acordada (vale negociação com colegas, para mudanças, contando que estejam documentadas). - Submeteu alterações por pull requests.

Conceito	
B	- Criou um Branch com commits autocontidos (ou seja, focados no problema e não interferindo em outras parte do projeto). Ex: atualizar com pull, fazer iterações… - Associou cada pull-request aos seus issues. - Adicionou a forma de onda gerada nos testes ao pull-request.
A	- Desenvolveu código modularizado, comentado e claro. Usando as ferramentas e técnicas mais apropriadas. Ex: port map.

Facilitador

Conceito	
I	- Não acompanhou o projeto, deixando os colegas sozinhos. - O relatório das atividades não é condizente com o real desempenho dos integrantes (analisado via git).
D	- Não verificou se o pull request é consistente (não altera arquivos de outros). - O relatório das atividades não é claro e não descreve a atuação de cada integrante. - Não conseguiu demonstrar o projeto em operação ou explicar os possíveis problemas.
C	- Atualizou o repositório pelo Fork. - Fez a manutenção do Kanban board (GitHub project). Ex: criar cards. - Acompanhou e deu feedback, mesmo que mínimo, para membros do time. - Aceitou os pull-requests. - Fez o relatório das atividades descrevendo o papel de cada integrante com clareza.
B	- Resolveu conflitos de merge nos pull requests. – Deu feedback através das ferramentas dos sistemas (Git e Slack e Github). Ex: Comentário e pull requests do Git. - Reforçou a arquitetura do sistema. - Fez uma boa distribuição (homogênea) das tarefas.
A	- Agregou informação relevante para o grupo desenvolver melhor o projeto. Ex: ranqueou atividades no Kanban, documentação extra.