Insper

Elementos de Sistemas - Projeto C - Ferramental

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

Março, 2018

> **!** Obrigatório ser realizado para o desenvolvimento do projeto D.

Travis

> **!** Facilitador deve fazer porém todos devem acompanhar, usar o monitor !

Até agora vocês utilizaram o Travis de maneira transparente, sem entender muito como ele foi configurado. Nessa nova etapa do desenvolvimento do computador Z01 iremos modificar o arquivo de configuração do Travis para que ele posso executar os novos scripts de teste (testar a ULA).

Note que no diretório raiz do repositório Z01, temos um arquivo chamado **.travis.yml** (usuários LINUX esse arquivo estará oculto), esse arquivo faz toda a configuração do teste de integração contínua e é dividido em algumas partes :

1. Configuração da máquina virtual
2. Instalação das dependências :

- python
- quartus
- java

3. Execução dos scripts de teste

Nessa etapa, estamos interessados em adicionar mais um script de teste no projeto (o script que testa a ULA).

Abra o arquivo .travis.yml e na última linha (seção script), modifique para ficar na seguinte maneira :

```
script:
- python Projetos/0-Infra/testeVHDL.py
- python Projetos/C-LogicaCombinacional/script/testeLogicaCombinacional.py
- python Projetos/D-UnidadeLogicaAritmetica/script/testeULA.py
```

Note que nesse script já tinhamos o teste da infra e o teste do projeto C. Faça um commit e Ainda hoje estarei enviando o código já correto, só não consegui testar.

Qualquer dúvidas por favor mandar e-mail, Obrigado pela atenção!

Professor Auxiliarenvio o código para o github, o travis agora irá executar esse novo script de testes.

Teste

! Todos devem realizar esses passos.

Começaremos a utilizar um arquivo de testes que indicará quais módulos serão testados, com isso iremos conseguir fazer com que o travis passe sem mesmo todos os módulos estarem implementados.

Agora existe um novo arquivo no caminho : Z01/Projetos/D-UnidadeLogicaAritmetica/tests/config.txt

Esse arquivo possui listado todos os módulos a serem testados, porém só aqueles que não estiverem comentados serão executados, possibilitando que testemos somente aquele que iremos implementar.

config.txt

```
# Testes a serem executados
# Descomente os módulos que deseja testar

#Add16.vhd
#ALU.vhd
#comparador16.vhd
#FullAdder.vhd
HalfAdder.vhd
#Inc16.vhd
#inversor16.vhd
#zerador16.vhd
```

Note que originalmente o módulo HalfAdder está descomentado e será testado quando o script de teste for chamado.

Visualizando a simulação

! Crie uma branch nova : halfAdder

É possível visualizarmos a forma de onda de uma simulação gerada. Para isso será necessário invocarmos a parte gráfica do modelsim. Vamos fazer isso com

o HalfAdder, mantenha ele descomentando no arquivo config.txt e ao invocar o script de teste passe o parâmetro –gui, como a seguir :

```
python testeULA.py --gui
```

Esse comando irá executar a simulação e abrir no software modelsim (toda a simulação já é realizada por esse software, mas estava sendo gerenciada pelos scripts de simulação em python).

> **!** Antes de continuar, o módulo do half-adder já deve está implementando e testado (na forma tradicional).

Modelsim

O modelsim não possui uma interface moderna, porém é o software de simulação de hardware mais completo e mais utilizado pela industria. Siga os passos a seguir para conseguirmos visualizarmos a forma de onda :

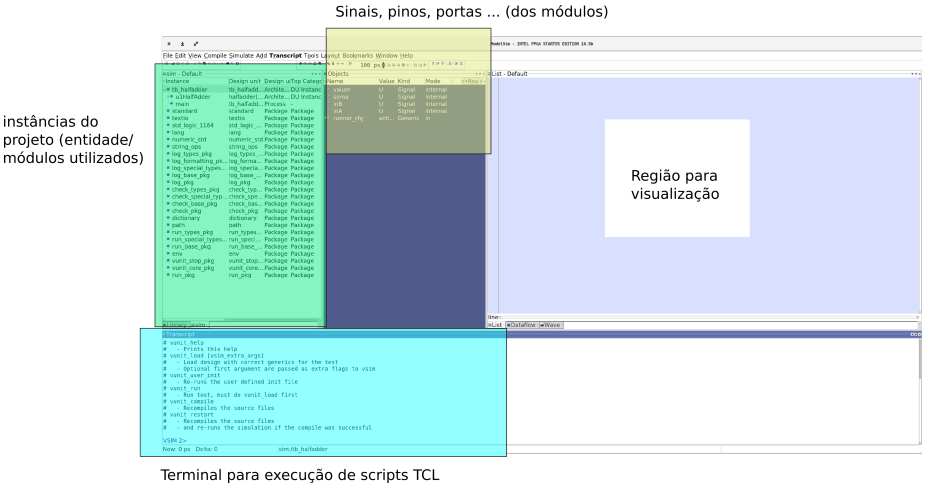


Figure 1: ModelSim

Primeiramente devemos adicionar os sinais que desejamos visualizar, no caso do HalfAdder queremos ver suas entradas (a,b) e suas saídas (soma, vaium).

Para isso clique em Wave :

E depois selecione no menu das instâncias o módulo u1HalfAdder, com isso poderemos selecionar quais pinos/ sinais gostaríamos de visualizar desse bloco. Arraste todos os sinais para o wave form :

Resultando em :

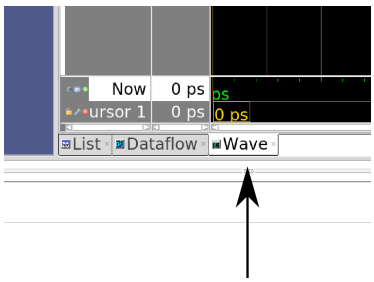


Figure 2: ModelSim

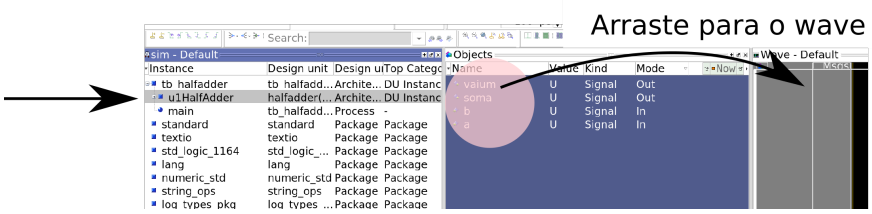


Figure 3: ModelSim

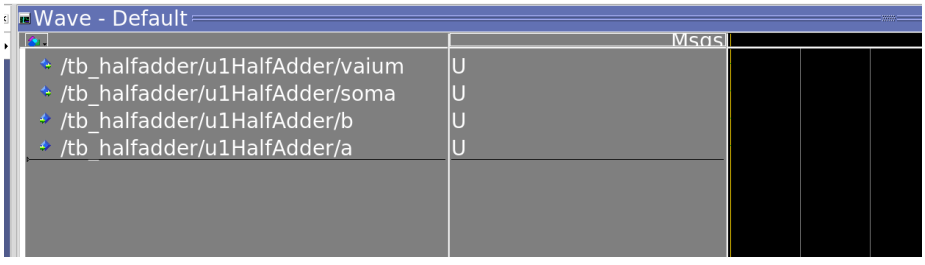


Figure 4: ModelSim

Agora podemos executar a simulação, na região do script TCL execute o seguinte comando : **vunit_run**

```
VSIM 7> vunit_run
```

Podemos agora visualizar a forma de onda :

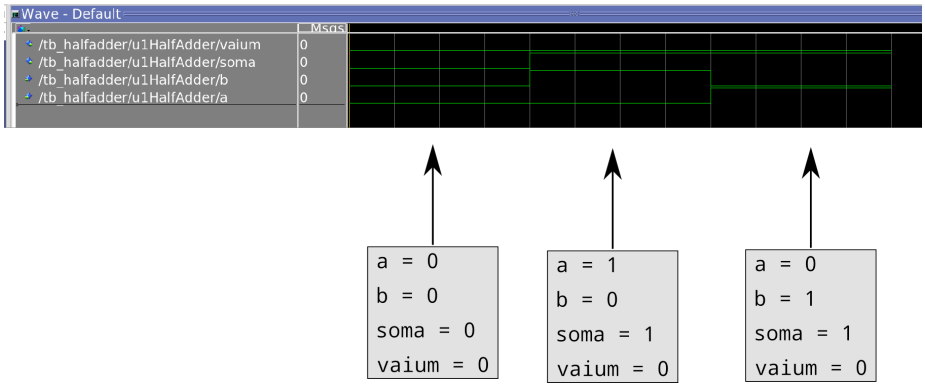


Figure 5: ModelSim

TestBench

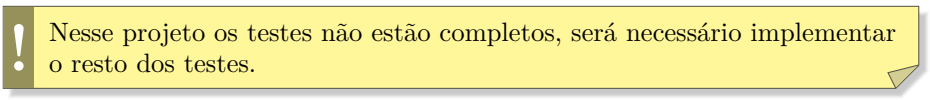
Testbench (bancada de teste) é a forma utilizada para verificarmos se um projeto de HDL (VHDL, Verilog, ...) está certo, o testbench pode ser escrito em várias linguagens inclusive em VHDL.

Os TestBenchs estão localizados nas pastas dos projetos em

/tests/tst/

No projeto D temos os seguintes testbenchs definidos (um para cada módulo a ser implementando):

```
/tests/tst/
tb_Add16.vhd
tb_ALU.vhd
tb_comparador16.vhd
tb_FullAdder.vhd
tb_HalfAdder.vhd
tb_Inc16.vhd
tb_inversor16.vhd
tb_zerador16.vhd
```



tb_HalfAdder.vhd

Note que na simulação do HalfAdder não testamos um dos casos de entrada : 1 + 1, esse teste está incompleto. Vamos ver como esse teste foi criado:

O arquivo tb_HalfAdder faz a inclusão do módulo HalfAdder (component e port map) instanciando esse módulo para uso. No arquivo é criado *estímulos* na entrada do componente e verifica-se se a saída está de acordo com o que deveria ser feito.

```
file : tb_halfAdder.vhd

....

u1HalfAdder: HalfAdder port map(inA, inB, soma, vaium);

....

-- Teste: 1
-- 0 + 0
inA <= '0'; inB<= '0';
wait for 200 ps;
assert(soma = '0' and vaium = '0')  report "Falha em teste: 1" severity error;

-- Teste: 2
-- 0 + 1
inA <= '0'; inB<= '1';
wait for 200 ps;
assert(soma = '1' and vaium = '0')  report "Falha em teste: 2" severity error;

-- Teste: 3
-- 1 + 0
inA <= '1'; inB<= '0';
wait for 200 ps;
assert(soma = '1' and vaium = '0')  report "Falha em teste: 3" severity error;
```

modificando o arquivo

Agora vamos modificar o arquivo para inserir o teste que está faltando : 1+1, para isso inclua as linhas a seguir logo após o teste 3:

```
-- Teste: 4
-- 1 + 1
```

```
inA <= '1'; inB<= '1';
wait for 200 ps;
assert(soma = '0' and vaium = '1')  report "Falha em teste: 4" severity error;
```

Como isso funciona ? Primeiro colocamos os valores desejados na entrada do componente :

```
inA <= '1'; inB <= '1';
```

é necessário aguardar um instante de tempo para que as portas lógicas fiquem com a saída estável (e para possibilitar a visualização na simulação):

```
wait for 200 ps;
```

E então, verifica-se se a saída do módulo está de acordo com o esperado (assert), caso contrário um erro (*severity error*) será gerado com a mensagem "Falha em teste 4":

```
assert(soma = '1' and vaium = '0')  report "Falha em teste: 4" severity error;
```

Testando novamente

Execute novamente a simulação com o modelsim e verifique se o módulo está se comportando corretamente : soma = 0 e vaium = 1.

- Repita os passos anteriores (–gui, wave, ...)

Pode-se também usar o teste sem a interface gráfica, porém ficamos muito sem saber o que está acontecendo