Lab 4: Gate Level Simulation

Universidade Estadual de Feira de Santana
Departamento de Tecnologia, Área de Eletrônica e Sistemas
TEC 499 MI - Sistemas Digitais

2016.1

Sumário

1. Introdução

No laboratório anterior consideramos a realização dos testes do circuito em nível funcional. Isso quer dizer que o simulador levará em consideração apenas características comportamentais que o seu circuito deve apresentar, deixando de lado aspectos de implementação. Apesar de ser útil para solucionar boa parte dos problemas de funcionamento geral do circuito, este tipo de simulação não reflete o comportamento real do circuito no sentido do atendimento às restrições de temporização do dispositivo. Em outras palavras, a simulação funcional não traz uma visão realista do comportamento do circuito quando implementado em um dispositivo FPGA.

O principal fator associado a isso está na ausência de características de atraso associadas a cada tipo de célula lógica, ou além mesmo das diferentes formas de implementação que um LE pode assumir (e.g.: LUTs com 2, 3 ou 4 entradas, por exemplo). Além disso, aspectos oriundos do processo de *Placement and Routing* não estão inseridos neste contexto de simulação.

Neste laboratório, você aprenderá como simular seus módulos após a síntese do circuito no software ALTERA Quartus II e testá-los no ModelSim-Altera antes de descarregá-los no dispositivo FPGA. Além disso, você deverá revisitar conceitos abordados anteriormente no que se refere à análise pós-síntese de um circuito digital em FPGA. Para isso, você irá sintetizar o circuito da Unidade Lógica e Aritmética (ULA), desenvolvido no laboratório anterior e realizar os mesmos testes aplicados na simulação funcional, agora em função da *netlist* sintetizada do seu circuito.

1.1 Netlist Simulation ou Gate Level Simulation

A etapa de geração da *netlist* em projetos de circuitos integrados consiste na verificação da sintaxe do código, seguida da geração da *netlist* a partir de uma semântica específica em código Verilog. Uma *netlist* utiliza expressões lógicas para descrever os circuitos, e inclui primitivas para componentes como somadores, flip-flops, e máquinas de estados.

O processo de síntese consiste em transformar a sua descrição RTL em uma estrutura que será programada em um dispositivo FPGA. A *netlist* de saída do circuito é específica para o dispositivo ou tecnologia de fabricação. Um exemplo de *netlist* é apresentado a seguir. Esta estrutura foi extraída do nosso primeira laboratório e representa a implementação da operação lógica `AND` entre as chaves `Switch[0]` e `Switch[1]` e a saída é produzida no sinal `LEDM_C_inv[0]`.

```
// Location: LCCOMB_X66_Y10_N24
cycloneive_lcell_comb \LEDM_C_inv[0] (
// Equation(s):
// LEDM_C_inv[0] = (\Switch[1]~input_o  & \Switch[0]~input_o )
  .dataa(gnd),
  .datab(gnd),
  .datac(\Switch[1]~input_o ),
  .datad(\Switch[0]~input_o ),
  .cin(gnd),
  .combout(LEDM_C_inv[0]),
  .cout());
// synopsys translate_off
defparam \LEDM_C_inv[0] .lut_mask = 16'hF000;
defparam \LEDM_C_inv[0] .sum_lutc_input = "datac";
// synopsys translate_on
```

No código acima, a instância `cycloneive_lcell_comb` correspondente à uma LUT de quatro entradas do dispositivo FPGA. Note que apenas as entradas `datac` e `datad` são utilizadas. O parâmetro `lut_mask` determina a máscara utilizada para definir a função lógica implementada através da LUT. Em conjunto com bibliotecas específicas da ferramenta de simulação é possível reproduzir exatamente o mesmo comportamento que o circuito teria quando implementado em FPGA.

Em dispositivos FPGA, a informação da *netlist* é geralmente acompanhada das informações dos atrasos associados a cada elemento utilizado no projeto (célula lógica ou *buffer* de entrada e saída, por exemplo). O código a seguir foi extraído do *Standard Delay Format Output File* (`.sdo`) da mesma implementação da porta `AND` mencionada acima. Este arquivo é um padrão utilizado na indústria para informação de atrasos padrão associados a elementos lógicos de uma determinada

tecnologia. Não entraremos em detalhes sobre o conteúdo deste arquivo, mas você pode ficar a vontade para explorá-lo nas etapas a seguir.

```
(CELL
  (CELLTYPE "cycloneive_lcell_comb")
  (INSTANCE LEDM_C_inv\[0\])
  (DELAY
    (ABSOLUTE
      (PORT datac (3.079:3.079:3.079) (3.337:3.337:3.337))
      (PORT datad (2.763:2.763:2.763) (3.017:3.017:3.017))
      (IOPATH datac combout (0.285:0.285:0.285) (0.281:0.281:0.281))
      (IOPATH datad combout (0.155:0.155:0.155) (0.139:0.139:0.139))
    )
  )
)
```

Estas informações possibilitam ao projetista analisar o circuito levando em conta informações mais realistas, tais como os atrasos associados aos elementos lógicos, ou induzidos pelo processo de roteamento.

2. Recursos

Para obter o conteúdo do laboratório, faça o download dos arquivos no seu Quadro do Trello. Certifique-se de que os diretórios `fpga`, `sim`, `src` e `syn` estão contidos dentro da pasta `lab4`. A estrutura de diretórios de simulação (`sim`) considera dois ambientes diferentes, respectivamente, para os testes pré-síntese e pós-síntese. Dentro do diretório `sim`, você encontrará a seguinte organização:

- `sim/fun` : contém o ambiente para simulação funcional (pré-síntese);
- `sim/syn` : contém o ambiente para simulação da *netlist* (pós-síntese);
- `sim/tb` : contém os arquivos de teste (*test bench*) em Verilog;
- `sim/tests` : contém os vetores de teste;

3. Pré-laboratório

Sugerimos fortemente que você inicie a análise do código Verilog de antemão. Considere que você terá uma semana para se preparar para a apresentação final deste laboratório. Além disso, é importante garantir que todas as falhas do roteiro anterior tenham sido identificadas e devidamente corrigidas.

Antes de começar, copie os arquivos RTL descritos em Verilog do projeto da **ALU** oriundos do **Lab 3** para os diretórios `lab4/src`. Copie também os arquivos de teste (*test bench*) deste mesmo roteiro dentro do diretório `lab4/sim/tb`. Esta etapa é importante, haja visto que os *scripts* de síntese e simulação consideram o cumprimento desta etapa.

Uma vez configurado o seu novo ambiente de projeto, recomendamos a você analisar a sanidade do seu código, realizando a etapa de análise e síntese do circuito para o dispositivo Cyclone IV **EP4CE30**. Este procedimento pode ser realizado a partir da sequência de comandos a seguir:

```
% cd ~/lab4
% make -C fpga map
```

Com isso, você será capaz de analisar as mensagens produzidos pela ferramenta de síntese. Certifique-se de verificar todas as mensagens e considerar mudanças no código, caso identifique que alguma delas possa acarretar em um mal comportamento do circuito após sintetizado.

4. Procedimento

O objetivo principal deste laboratório é orientá-lo de modo a ser capaz de identificar a diferença que existe entre a simulação funcional e a simulação em nível de *netlist* em projetos de circuitos integrados digitais. Durante este procedimento, você deverá utilizar as três técnicas de teste de projeto empregadas no laboratório anterior, no sentido de validar a sua implementação.

4.1 Sintetizando o Projeto

Uma vez analisadas todas as mensagens pré-síntese física, o próximo passo consiste em realizar as etapas posteriores do fluxo de implementação FPGA (a esta altura, consideramos que você seja capaz de fazer isso sem orientação). Em resumo, a sequência comandos a seguir primeiro removerá os arquivos do diretório de compilação e em seguida realizará as etapas de Análise e Síntese, *Place and Route*, Geração do Arquivo de Programação (*bitstream*) e a *Static Timming Analisys*. Este último será de grande importância para análise das restrições de tempo associadas a sua implementação.

```
% make -C fpga clean
% make -C fpga
```

4.2 Static Timming Analisys

Nos laboratórios anteriores nós descobrimos como a *Static Timming Analisys* (STA) pode nos ajudar a identificar aspectos de temporização do circuito. Uma das principais utilidades da STA está na possibilidade de determinação do atraso associado ao caminho crítico do circuito. Em dispositivos FPGA, o conceito de caminho crítico é levemente diferenciado, levando em consideração a quantidade de níveis lógicos em função do encadeamento de LUTs.

Neste laboratório você deve utilizar o relatório da STA para definir o intervalo de transição entre os estímulos de entrada do seu *test bench*. No laboratório anterior definimos o comando `timescale` como referência de unidade de tempo e precisão associada ao teste. Além disso, utilizamos os comandos `parameter` e `localparameter` para determinar o período de um ciclo de clock no trecho de código a seguir.

```
parameter Halfcycle = 5; // half period is 5ns
localparam Cycle = 2*Halfcycle;
reg Clock;
// Clock Signal generation:
initial Clock = 0;
always
  #(Halfcycle) Clock = ~Clock;
```

As definições acima foram deixadas de lado, visto que o circuito da ALU é puramente combinacional. Todavia, é possível utilizar deste tipo de estrutura no sentido de determinar o intervalo base entre as transições que produzem os estímulos de entrada no seu DUT. No exemplo acima, considerando que o comando `timescale` foi definido como 1ns com precisão de 1ps, temos que cada ciclo de clock será igual a 10ns.

Antes de prosseguir, localize no relatório da STA o maior atraso associado ao seu circuito. A análise de propagação de sinal encontra-se no `Static Timing Analyzer Report` no arquivo `fpga/ml505top.sta.rpt`. Localize o item `Propagation Delay` na seção `Multicorner Timing Analysis Summary`. Anote aproximadamente o máximo valor encontrado para cada coluna. Substitua agora o valor do parâmetro `Halfcycle` pelo correspondente à metade do valor que você encontrou. Este valor indica que o circuito implementado não será capaz de garantir um resultado estável na saída da ALU em intervalos de transição menores que o maior atraso. Note que isso pode ser válido para uma combinação ou mesmo para um conjunto de combinações de entrada e saída.

4.3 Geração da Netlist

Uma vez implementado o circuito, você poderá utilizar o comando a seguir para produzir a *netlist* correspondente ao *top level* do seu circuito.

```
% make -C fpga netlist
```

Este procedimento produzirá uma saída formada por um conjunto de arquivos no diretório `/lab4/syn/src`. Por enquanto, consideraremos apenas os arquivos com a extensão `*.vo` e `*.sdo`. O primeiro, corresponde à *netlist* em Verilog do circuito sintetizado e o último contém os padrões de atraso, associados ao dispositivo mapeado e o circuito implementado, no formato SDF (*Standard Delay Format*). Ambos os arquivos serão utilizados posteriormente para a simulação pós-síntese no ModelSim.

5. Testando o Projeto

Ambos os diretórios de simulação compartilham das mesmas estruturas de teste. A diferença está presente nos *scripts* de compilação e nos comandos executados no ModelSim para simulação em modo *gate level*. O principal elemento a ser considerado dentro do `Makefile` é a variável `TOP_LEVEL`. Modifique o seu valor de modo que este corresponda ao nome do módulo *top level* do seu circuito. Além disso, no arquivo `Makefile` utilizado para simulação pós-síntese (diretório `sim/syn`), você deve verificar o valor do atributo `NETLIST_REGION`. O valor associado a ele deve corresponder ao nome da instância do bloco que está sendo testado (DUT/DUV) utilizada nos arquivos de *test bench*.

5.1 Simulação Pré-síntese

Antes de iniciarmos a simulação pós-síntese, é necessário garantir que as alterações realizadas no código Verilog tenham afetado o funcionamento do circuito executando um teste funcional. Para isso, entre no diretório de simulação correspondente e analise os testes produzidos.

```
% cd sim/
% make -C func
```

Certifique-se de que todos os testes tenham passado antes de partir para a próxima etapa. Note ainda que, qualquer modificação na descrição em RTL deve ser refletida na *netlist* do circuito, a partir de uma nova síntese. Para isso, remeta novamente às Seções 4.1 e 4.3. Lembre-se ainda que, eventualmente, será necessário limpar (`clean`) os arquivos de compilação do Quartus antes de realizar uma nova síntese ou uma simulação.

5.2 Simulação Pós-síntese

Como vimos anteriormente, na simulação pós-síntese, a ferramenta passa a considerar não mais apenas a descrição funcional do circuito representada por uma *netlist*. Neste novo ambiente, estaremos em contato direto com características de temporização do dispositivo. O *script* de compilação dentro do diretório `sim/syn` já está configurado para buscar os arquivos nos diretórios definidos anteriormente e você não deve se preocupar em alterá-los. Sinta-se à vontade para estudar os códigos dentro de cada *script* de teste e fazer modificações de acordo com sua conveniência.

Dessa vez, execute o comando a seguir para obter os resultados da simulação pós-síntese.

```
% cd sim/
% make -C syn
```

Se nenhuma alteração, além daquelas sugeridas até agora, for feita no código, você deve se deparar com uma série de mensagens semelhantes a que segue.

```
# FAIL: Incorrect result for opcode 000000, funct: 100011:
# A: 0xdbfa08fd, B: 0x318c32a8, DUTout: 0xaa6dd655, REFout:
```

Note que, no laboratório anterior, utilizamos a definição discreta de unidade de tempo para indicar o intervalor de transição entre os estímulos de entrada. Você deve ter utilizado um comando semelhante ao `#1;`. Isso indica ao simulador que um estímulo estará ativo na entrada durante um intervalo de 1ns, considerando o `timescale` definido.

Substitua os comandos `#1;` pela diretiva `#Cycle;`. Isso dirá para a ferramenta de simulação aguardar um intervalo de tempo igual ao valor indicado pela ferramenta como o maior atraso de propagação de um sinal de entrada até a saída. Se tudo correr como esperamos, você deverá ver a saída:

```
# ALL TESTS PASSED!
```

Após esta etapa, remova o comando `$finish` do procedimento `checkOutput` e realize as análises e seguir.

- Reduza o valor do parâmetro `Halfcycle` em 20%.
- Retorne o valor original e em seguida aumente em 20 % o valor do parâmetro `Halfcycle`.

Para cada uma destas análises, anote o total de testes que apresentaram falha, se for o caso. Pode ser mais fácil definindo um contador de erros ou contabilizar as falhas manualmente no arquivo `transcript` correspondente.

6. Acompanhamento

Agora que você finalizou os procedimentos, está pronto para alçar voos mais altos no projeto e síntese de circuitos integrados digitais. Na sessão de acompanhamento você deve:

1. Apresentar o ambiente de simulação completo ao seu professor, contendo o código Verilog da ALU os testes e a *netlist* produzida pelo Quartus. Destaque as modificações realizadas nos arquivos de *test bench* para validar a síntese pós-simulação.
2. Analisar o comportamento do circuito quando o período foi menor que o maior atraso de propagação.
3. Aponte o total de elementos lógicos utilizados (LEs, LABs, FF, etc), considerando os respectivos modos de operação.
4. Determinar a frequência máxima de operação do circuito.
5. Especificar a taxa de transferência do seu circuito em função da frequência de operação, considerando um caminho de dados de 32 bits.

Você também deve ser capaz de demonstrar que ambos os ambiente de teste utilizam o mesmo conjunto de *test benches*.