Lab 1: Layout Físico de Projetos FPGA

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

2016.1
rev. 1.8

Sumário

Introdução

Nos procedimentos anteriores, você aprendeu a manipular entradas e saídas no FPGA usando código Verilog. Neste laboratório, você verá como o código Verilog que você escreveu é mapeado na forma de elementos lógicos do dispositivo FPGA. Ao longo do resto do semestre, você focará em métodos mais abstratos de descrição de circuitos, e as ferramentas utilizadas para transformar esta descrição em um hardware apropriado.

Ao longo deste laboratório você notará que a capacidade de representar um hardware de forma mais abstrata aumenta a produtividade e a flexibilidade do projeto. Por outro lado, também esconde alguns detalhes de implementação. Os procedimentos a seguir o ajudarão a compreender a arquitetura básica do FPGA, assim como auxiliarão você a adotar decisões de projeto otimizadas no futuro. O principal objetivo aqui é apresentar uma visão geral do layout físico de um dispositivo FPGA, tomando como ponto de partida o FPGA **ALTERA Cyclone IV**.

1. Pré-laboratório

Para ajudar a você se familiarizar com o dispositivo FPGA utilizado ao longo deste período, leia atentamente o **Capítulo 2: Logic Elements and Logic Array Blocks in Cyclone IV Devices**, no documento Cyclone IV Device Handbook e responda as seguintes questões:

1. Quantos LEs existem um único LAB?
2. Identifique os modos de operação de cada LE e determine a diferença entre eles?
3. Quantas entradas possui uma LUT no modo normal em um dispositivo da família ALTERA Cyclone IV?
4. Quantos LEs existem em um dispositivo ALTERA Cyclone IV **EP4CE30**? (ver Capítulo 1: Cyclone IV FPGA Device Family Overview, no Cyclone IV Device Handbook)
5. Quantos LEs você estima serem necessários para implementar o módulo `Adder`, descrito no laboratório anterior?

2. Ferramentas EDA

Antes de mergulharmos neste roteiro, vamos nos familiarizar com as o fluxo básico de ferramentas de EDA (*Electronic Design Automation*) para FPGA. Estas ferramentas, de uma forma geral, traduzem um código HDL em uma representação física compatível com a tecnologia do dispositivo. Para isso, realizam um conjunto de transformações no seu projeto através de um fluxo de implementação definido. A cada etapa de transformação, a sua descrição vai se aproximando do que podemos chamar de representação final do seu projeto.

A seguir apresentamos uma breve descrição sobre cada um desses estágios, sob o ponto de vista da síntese de circuitos em FPGA. Apesar de consistir de um conjunto relativamente similar de operações, cada ferramenta adota subdivisões específicas. Este roteiro foi elaborado com base no fluxo de execução da ferramenta **ALTERA Quartus II**.

2.1 Análise & Síntese

A ferramenta de síntese, no nosso caso, Quartus II Analysis & Synthesis (`quartus_map`), é o primeiro software a processar o seu projeto. Além de outras tarefas, ele é responsável por transformar as estruturas lógicas e registradores que você descreveu em Verilog em LEs e outros elementos presentes no dispositivo FPGA.

Por exemplo, se você descreveu um circuito composto de muitas portas, mas que utiliza apenas 4 entradas e uma saída, `quartus_map` irá mapear o seu circuito em um único LE de 4 entradas (dependendo apenas da característica das suas LUTs). Da mesma forma, se você descreveu um registrador, o mesmo será mapeado em um tipo específico de registrador presente em um dos LEs do FPGA. O produto final da etapa de síntese do projeto corresponde à uma *netlist*, um arquivo de texto composto por uma lista de todas as instâncias de componentes primitivos (LEs, LABs etc) no circuito e a descrição de como eles estão conectados.

Além disso, a etapa de síntese é responsável por ler a *netlist* gerada e traduzir cada componente em um equivalente específico do dispositivo FPGA ALTERA Cyclone IV EP4CE30 que nós dispomos no laboratório. Durante este processo, características físicas de cada componente, incluindo os atrasos de conexão, são introduzidos.

2.2 Posicionamento & Roteamento

Na ferramenta ALTERA Quartus II, estes dois processos são também chamados de **Fitting**. O *fitter*, representado pelo comando `quartus_fit` determina a localização específica de cada componente no dispositivo FPGA (e.g.: LEs e LABs) a partir do projeto mapeado, em um processo chamado de posicionamento (*placement*). Por exemplo, cada LE é atribuído a um LAB específico de acordo com a localização espacial das portas de E/S.

Uma vez concluído a etapa de *placement* do circuito, todas as conexões especificadas na *netlist* precisam ser implementadas. Em geral, o roteamento (*route*) é a tarefa que toma mais tempo de projeto em uma EDA. Otimizações no processo de *place and route* podem ajudar na obtenção de circuito otimizados em termos de velocidade e execução e área ocupada em *chip*.

3. Procedimento de Laboratório

Neste laboratório, você usará uma ferramenta chamada **Chip Planner** para manipular a lógica e os sinais do projeto que você descreveu em Verilog. Sendo assim, este exercício será útil para visualizar o hardware e compreender os processos subjacentes.

3.1 Configuração

Localize o arquivo de laboratório `lab1.zip` no seu quadro no Trello e salve-o em seu computador. Em seguida, execute o seguinte comando:

```
unzip -xvf lab1.tar.gz
```

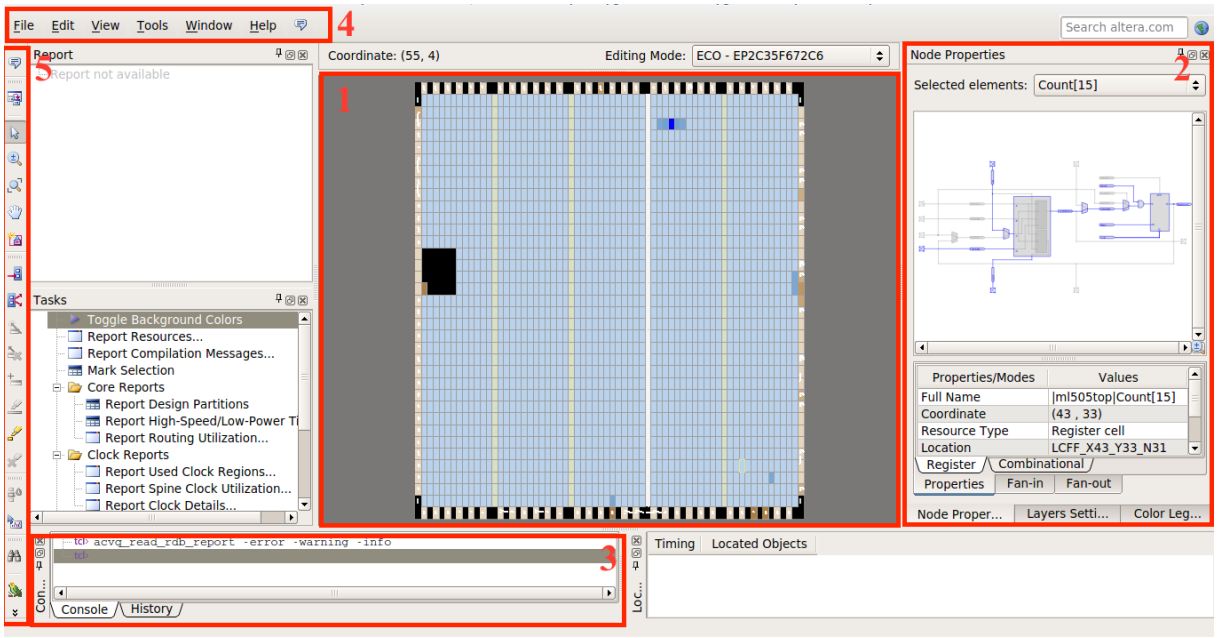
Antes de iniciarmos nossas análises, iremos retomar o laboratório anterior. Para isso, compile o projeto da última semana (se você já não o fez). Note que você terá que compilar os arquivos para completar esta parte do laboratório.

```
cd ~/lab0
make -C fpga
```

Agora abra o projeto na interface gráfica do ALTERA Quartus II. A extensão `.qpf` representa um

arquivo de projeto reconhecido pelo Quartus. Em seguida localize o aplicativo `Chip Planner` no menu `Tools`. Ao abrir a ferramenta, você será capaz de visualizar como o seu projeto será mapeado para o dispositivo FPGA.

```
cd ~/lab0/fpga
quartus ml505top.qpf
```



A imagem acima apresenta a visão geral da janela principal do `Chip Planner`. Ela é dividida, forma geral, nas seguintes janelas:

1. `Janela do Dispositivo` - Apresenta o esquemático do FPGA e destaca as partes do dispositivos que estão sendo utilizadas atualmente.
2. `Janela de Propriedades` - Lista as propriedades do componente selecionado.
3. `Saída do Console` - Imprime as mensagens oriundas do comandos internos ou de diagnóstico do processo. Serve também para introduzir comandos de script `TCL`.
4. `Barra de Tarefas` - Conjunto de botões úteis para manipulação das janelas.
5. `Barra de Botões` - Conjunto de botões utilizados para manipular o projeto.

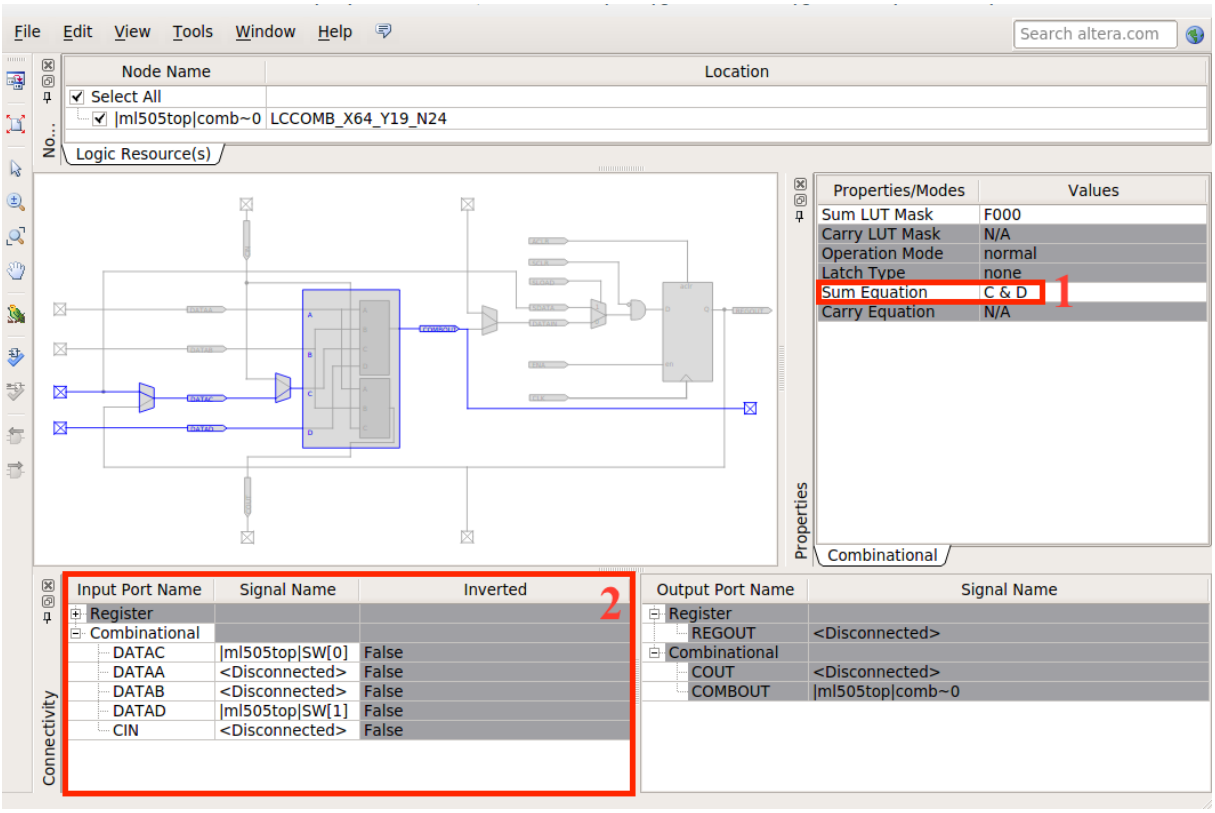
Fique à vontade para explorar a ferramenta e identificar os elementos estudados nos Capítulos 1 e 2 do documento **Cyclone IV Device Handbook**.

3.2 Modificando o Projeto

Agora vamos analisar o projeto em busca dos módulos que você descreveu no último laboratório. Se você ampliar a imagem na `Janela do Dispositivo` (pressionando a tecla `CTRL` e rolando o mouse)

próximo às regiões destacadas e clicar em um dos blocos, verá que o nome dele faz referência a um LAB.

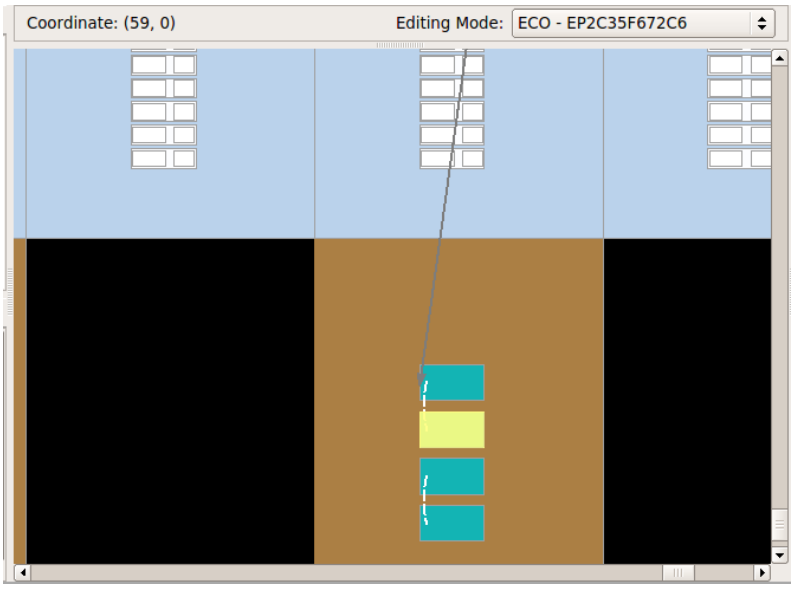
Nas seções anteriores, você descobriu que um LAB é formado por um conjunto de LEs que, por sua vez, possui uma ou mais Look Up Tables (LUTs) que implementam a lógica desejada a partir de um conjunto restrito de entradas. Clique no botão `Find`, na `Barra de Botões` (botão representado por um binóculo). Uma ferramenta de busca será aberta na `Janela de Propriedades`. Introduza a palavra chave `LEDM_C[0]` e clique duas vezes na primeira opção do resultado. Localize então a aba chamada `Fan-in` em `Node Properties`. Na coluna `Signal Name`, identifique o sinal associado à porta de entrada `DATAIN` ou `IN` e clique duas vezes na linha correspondente. Por fim, clique duas vezes na imagem que aparece na região superior da `Janela de Propriedades`. Este procedimento deverá resultar em uma imagem semelhante à figura abaixo.



Na imagem acima, no quadro #1 verifique a função lógica realizada na LUT. No quadro #2, identifique quais as portas de entrada estão envolvidas neste processo. Com isso, você pode deduzir que este LE implementa o seguinte código Verilog:

```
and (LEDM_C[0], Switch[0], Switch[1]);
```

Esta informação pode ser confirmada ao procurar para onde os fios que estão conectados ao LE estão indo. Para isso, você deve retornar à tela inicial do `Chip Planner`, localizar novamente o LE correspondente e então, na aba `Fan-out` clicar duas vezes no valor referente a `COMBOUT`. A representação da ligação da saída do LE com o pino de saída do FPGA será representada, como mostra a imagem destacada a seguir.



Além de analisar a disposição do circuito no dispositivo, o `Chip Planner` possibilita a modificação da função lógica realizada na LUT interna ao LE. Para isso, você deve seguir os passos a seguir:

1. Retorne até a janela do `Resource Property Editor` do LE que implementa a função `AND`.
2. Na propriedade `Sum Equation`, modifique a `LCCOMB LUT` de modo a realizar uma `OR` no lugar da `AND`. Você pode fazer isso substituindo o & pelo caractere #.
3. No menu `Edit`, clique em `Check and Save All Netlist Changes`. Aguarde até que o processo seja finalizado e encerre o `Chip Planner` e o Quartus.

3.3 Embutindo no Hardware e Testando

Agora nós iremos programar seu projeto editado diretamente na ferramenta de análise do *layout* físico do hardware (*floorplaning*). Para isso, basta digitar o comando:

```
cd ~/lab0
make -C fpga program
```

Uma vez programado, localize as chaves DIP correspondentes e a matriz de LEDs da placa Mercurio IV. Seu protótipo deve agora apresentar os resultados correspondentes à realização de uma operação lógica `OR` entre as chaves `SW0` e `SW1` no LED (`R0,C0`). Modifique as posições das duas chaves e note a mudança no comportamento do LED.

3.4 Extendendo o Projeto

No laboratório anterior nós construímos um somador *ripple* no arquivo Adder.v. Para esta próxima parte, copie seu `Adder.v` (descrito no laboratório anterior) sobre o arquivo `Adder.v` e o `FA.v` presentes no diretório `lab1/verilog`. Nesta etapa, nós iremos fazer experimentos com o barramento de entrada do somador e analisar como isso afeta a utilização de recursos para somadores de diferentes tamanhos.

3.4.1 Entendendo o Sistema de Compilação

Após copiar o conteúdo do `Adder.v` desenvolvido no laboratório anterior, você deve estar pensando em tão logo sintetizar o código em uma *netlist*, para verificar a quantidade de recursos do dispositivo que sua implementação utilizará. Antes de realizar estas análises, é importante começarmos a entender sobre o funcionamento do *script* de compilação fornecido junto com os arquivos de laboratório. As instruções a seguir apresentam uma introdução ao sistema de compilação utilizado neste laboratório.

O arquivo distribuídos junto com esta descrição contém um `Makefile` que responsável por automatizar o processo de compilação e depuração do seu projeto. Para aqueles que não são familiarizados com o `make`, você usará o comando passando o caminho onde ele se encontra (em geral no diretório `fpga`) e um alvo como argumentos. Por exemplo, o comando '`make -C fpga map`' irá executar apenas a ferramenta de síntese (`quartus_map`). Outros alvos importantes serão discutidos em seguida e ao longo do curso.

O `Makefile` possui um conjunto de variáveis definidas nas primeiras linhas do arquivo que permitem você controlar o comportamento do sistema de compilação.

```
PROJECT = FPGA_PROJECT_NAME
TOP = PROJECT_TOP_LEVEL
ASSIGNMENT_FILES = $(PROJECT).qpf \$(PROJECT).qsf

FAMILY = "Cyclone IV E"
PART = EP4CE30F23C7
BOARDFILE = MIVPins
```

A variável mais importante para este roteiro é representada pela variável `TOP`. Esta variável é utilizada para modificar o módulo *top level* utilizado na síntese do hardware. Você pode modificar esta variável diretamente no `Makefile` (lembrando-se de alterar de volta uma vez finalizado o teste) ou chamando o comando desta forma:

```
make -C fpga clean
make -C fpga 'TOP=<seu top module aqui>'
```

A primeira coisa que você deve fazer depois de escrever os seus novos módulos Verilog (como o `Adder.v`) é verificar se ele, ao menos, faz algum sentido. O alvo `map` é bastante útil neste sentido. Execute ele da seguinte forma:

```
cd ~/lab1
make -C fpga map
```

Caso encontre algum problema no momento da síntese, você pode usar o alvo `clean` para garantir que o processo seja reiniciado. Isso removerá os arquivos de síntese gerados pelo Quartus, exceto aqueles indispensáveis para a interpretação do projeto.

3.5 Sintetizando seu Código

As ferramentas da ALTERA que o alvo `map` invoca produz uma série de arquivos de saída. Alguns alertas (*warnings*) e erros são óbvios, outras vezes não são. Por enquanto, manteremos nosso foco nos arquivos de relatório, compreendidos pela extensão `.rpt`, localizados dentro do diretório `fpga`. Você pode visualizar as informações deste arquivo em seu editor de textos preferido.

Infelizmente, a ALTERA não disponibiliza uma ferramenta independente para análise dos relatórios fora da interface gráfica principal do Quartus. Caso ache pertinente, você pode abrir o projeto na GUI do Quartus usando o comando `quartus fpga/ml505top.qpf` e localizar a janela de relatórios Caso enfrente dificuldades na visualização dos arquivos de relatório, peça ajuda ao seu professor.

Os arquivos de relatório fornecem uma síntese completa das informações geradas a partir das ferramentas. Por enquanto, estamos interessados nos erros e alertas gerados durante o processamento do código Verilog. Para automatizar este processo, no terminal, você pode usar os comandos a seguir para buscar por alertas ou erros durante a compilação:

```
cat fpga/mltop505.map.rpt | grep "Warning"
cat fpga/mltop505.map.rpt | grep "Error"
```

Erros **devem** ser corrigidos antes de você continuar o laboratório. Pergunte ao seu professor caso tenha dúvidas sobre a importância de um alerta (*warning*).

Um vez identificados e corrigidos todos os erros do estágio de síntese, você agora tem certeza de que o seu código Verilog está, ao menos, sintaticamente correto. Infelizmente, isso não significa muito, uma vez que compiladores Verilog são sensíveis a aceitar estruturas sem sentido (e retornar informações completamente sem sentido).

Outra etapa importante nesta análise é investigar o diagrama esquemático do circuito. Uma vez concluída a síntese, você deve executar o Quartus usando o comando '`quartus fpga/ml505top.qpf`'. Isso fará com que o Quartus seja executado e o nosso novo projeto aberto. Uma vez dentro do Quartus, siga até o menu `Tools` → `Netlist Viewers` → `RTL Viewer`.

Esse procedimento lhe dará uma descrição direta da hierarquia de blocos do seu projeto. Por padrão o visualizador apresenta inicialmente o módulo *top level* do circuito, permitindo que você acesse os módulos `Adder` e `FA`. Verifique se o seu módulo `Adder` está implementado corretamente. Não interprete como um problema o fato de alguns fios estarem desconectados.

Note ainda que esta visualização corresponde à uma representação direta da lógica descrita em HDL, sem levar em consideração características físicas do dispositivo FPGA. Entenderemos melhor estas transformações nas sessões a seguir.

3.6 Verificação

Uma vez que seu circuito esteja corretamente implementado, é hora de verificá-lo através da programação e uso na placa. O módulo `ml505top` instanciou cópias tanto do `Adder` quando do `BehavioralAdder`, uma versão alternativa do somador escrito em Verilog comportamental. Ele simplesmente diz `Out = A + B` e deixa que a ferramenta interprete a declaração. Você irá aprender mais sobre Verilog comportamental nos próximos laboratórios; por enquanto, você só precisa saber que é um nível mais alto de descrição do seu circuito.

O módulo *top level*, por sua vez, compara as saídas dos dois somadores para todas as possíveis entradas, e sinaliza um erro se houver alguma diferença. Nos próximos laboratórios, você vai conhecer outros métodos de teste para seus circuitos.

Para programar o seu projeto na placa, comece pelo comando '`make -C fpga`' no diretório `lab1`. Este processo deve compilar todo o seu projeto até a geração do *bitstream* de programação. Ao concluir este processo, programe o dispositivo FPGA executando o comando '`make -C fpga program`'.

Uma vez que o dispositivo tenha sido programado, o LED RGB poderá acender nas cores verde ou vermelha. O teste atribui a cor verde a indicação de sucesso, enquanto a cor vermelha indica uma falha. Se o LED RGB acender na cor vermelha, você pode verificar:

a) Se você descreveu a definição do `generate` corretamente

b) Modificando a estrutura do teste para somar as mesmas duas contantes e ligar o resultado a um LED não utilizado da matriz de LEDs.

Solicite ajuda ao seu professor, caso necessite de assistência.

3.7 Análise do Circuito

Agora que o seu circuito está completamente funcional, nós podemos realizar uma análise de utilização de recursos do dispositivo. Isso lhe dará uma melhor percepção sobre qual hardware as ferramentas inferiram a partir do seu código Verilog.

Neste momento será necessário alterar o valor da variável `TOP` no `Makefile`. Alterne o valor desta variável entre os módulos `Adder` ou `BehavioralAdder` de acordo com o que se pede. Lembre-se também que é possível alterar o módulo *top level* do circuito diretamente na chamada ao comando `make`. Em ambos os casos, lembre-se de limpar o diretório de compilação, utilizando a diretiva `clean`.

Para ambos os circuitos, analise a implementação do circuito final utilizando a ferramenta `Technology Map Viewer`, acessível através da interface gráfica do Quartus, no menu `Tools` → `Netlist Viewers`. Note agora que a nomenclatura dos blocos fazem referência à células lógicas do dispositivo FPGA. Clique duas vezes em uma das células lógicas e investigue como sua descrição está sendo implementada no dispositivo em termos de recursos físicos.

Uma vez concluída a análise preliminar de ambos os circuitos, para o circuito `Adder`, anote os valores a seguir considerando `Width = 16` e `Width = 32`. O parâmetro `Width`, neste caso, representa a largura do barramento de entrada e saída e consequentemente a quantidade de somadores do circuito. Você pode mudar o tamanho no código Verilog:

```
module Adder #(
  parameter Width = 16  // alternar entre 16 e 32
) ...
```

- Número de LEs ocupados
- Número de LEs ocupados por modo de operação
- Número de LABs totalmente ou parcialmente utilizados

- Determine o máximo atraso de propagação do circuito

Você pode encontrar os números referentes à utilização de LEs e LABs na sessão `Fitter Resource Usage Summary` do arquivo `fpga/ml505top.fit.rpt`. 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. Neste caso, cada coluna corresponde à uma combinação entre as transições de subida (R - *rise*) e de descida (F - *fall*). Cada combinação representa o tempo que o sinal leva para a primeira transição na entrada de um elemento do circuito produzir a segunda transição na saída deste mesmo elemento.

Localizar este valor manualmente pode facilmente se tornar uma tarefa tediosa. Neste caso, pode ser conveniente analisar o relatório usando a GUI do Quartus. Para isso, execute o comando `quartus fpga/ml505top.qpf`. Na Janela de Tarefas (`Tasks`), localize o item `TimeQuest Timing Analysis` e, em seguida, clique em `View Report`. A partir daí acreditamos que você é ser capaz de localizar o relatório indicado acima.

Ainda no Quartus abra a implementação do seu somador usando o `Chip Planner` e anote as funções lógicas presentes em qualquer uma das LUTs. Para lembrar como abrir o `Chip Planner`, consulte as seções anteriores.

Agora, realize o mesmo procedimento para o módulo `BehavioralAdder` (lembrando de compilar o projeto com o novo módulo `TOP`). Note as diferenças no hardware gerado e os recursos utilizados a partir dos módulos `Adder` e `BehavioralAdder`.

4. Acompanhamento

Para a sessão de acompanhamento, mostre ao seu professor os itens solicitados na **Seção 3.7 Análise do Circuitos**. Em seguida, responda às seguintes questões em uma folha de papel:

1. O número de LEs usados para implementar o módulo `Adder` foi o mesmo que você esperava que ele utilizasse durante sua estimativa? Justifique.
2. Descreva as características de implementação do módulo `BehavioralAdder` levando em consideração a composição de elementos lógicos do dispositivo.
3. Por que o módulo `BehavioralAdder` mapeou menos LEs quando comparado ao módulo estrutural `Adder` que você descreveu?