Universidade Federal Fluminense – UFF
Escola de Engenharia – TCE
Curso de Engenharia de Telecomunicações – TGT

Programa de Educação Tutorial – PET
Grupo PET-Tele

Tutorial sobre o ambiente de simulação ModelSim

(Versão: A2020M05D25)

Autores: Gabriel Bueno dos Santos Doria Oliveira
Lucca Sabbatini Reid Rodrigues
Vinícius Corrêa Figueira

Tutor: Alexandre Santos de la Vega

Niterói – RJ
Maio / 2020

Sumário

1 Introdução **2**
1.1 Motivações . . . 2
1.1.1 *Programmable Logic Device* (PLD) . . . 2
1.1.2 *Hardware Description Language* (HDL) . . . 2
1.1.3 Simulador *ModelSim* . . . 3
1.2 Objetivo . . . 3

2 *Download*, instalação e licenciamento do ModelSim **3**
2.1 *Download* . . . 3
2.2 Instalação . . . 6
2.3 Licenciamento . . . 9

3 Ferramentas básicas do ModelSim **12**
3.1 Introdução à interface gráfica . . . 12
3.2 Simulação básica . . . 13
3.3 Simulação com projetos . . . 18
3.4 Ferramentas para visualizar formas de onda . . . 23

4 Exemplos de simulações no ModelSim **25**
4.1 *Decoder* para *display* de 7 segmentos em Verilog . . . 25
4.2 Multiplexador 4x1 . . . 28
4.2.1 Implementação do MUX 4x1 em Verilog . . . 29
4.2.2 Implementação do MUX 4x1 em VHDL . . . 31
4.3 Meio somador ou *Half Adder* ou HA . . . 35
4.3.1 Implementação do HA em Verilog . . . 36
4.3.2 Implementação do HA em VHDL . . . 37
4.3.3 Simulação . . . 39
4.4 Somador completo ou *Full Adder* ou FA . . . 39

Referências **43**

1 Introdução

O Programa de Educação Tutorial (PET) [PETa], do Ministério da Educação (MEC), exige que os grupos PET desenvolvam atividades que contemplem, de forma indissociável, itens de Pesquisa, de Ensino e de Extensão. Além disso, os grupos devem estimular uma evolução positiva dos seus integrantes, dos demais alunos do seu curso de graduação, do próprio curso e da sua instituição.

Nesse sentido, o PET-Tele [PETb] procura desenvolver atividades e/ou atender a demandas que cumpram tais exigências.

A seguir, são apresentadas as motivações e o objetivo para o trabalho em questão.

1.1 Motivações

A seguir, são apresentadas as motivações básicas para o desenvolvimento desse trabalho.

1.1.1 *Programmable Logic Device* (PLD)

Um Dispositivo Lógico Programável (*Programmable Logic Device* ou PLD) é um circuito integrado digital que tem a capacidade de ser reconfigurado, para exercer funções lógicas diferentes a cada configuração.

Os tipos de PLD mais comumente utilizados são o CPLD (*Complex Programmable Logic Device*) e o FPGA (*Field Programmable Gate Array*).

A configuração de um PLD é feita a partir de uma especificação do circuito lógico a ser implementado. Essa especificação pode vir de um desenho esquemático ou de uma linguagem textual adequada, conforme é discutido a seguir.

O tutor tem sido responsável por disciplinas que envolvem sistemas digitais, que são: Circuitos Digitais e Processamento Digital de Sinais. Ao trabalhar o conteúdo de tais disciplinas, são abordadas e trabalhadas as implementações que envolvem PLDs.

Recentemente, o PET-Tele adquiriu um *kit* de desenvolvimento DE10-Lite, baseado em FPGA, com os objetivos de adquirir, sedimentar e propagar conhecimento sobre o assunto, bem como utilizá-lo em projetos do grupo. Como primeiro passo na utilização do kit, o grupo elaborou um manual de utilização que se encontra à disposição para *download* gratuito [BMS+].

1.1.2 *Hardware Description Language* (HDL)

Quando os primeiros simuladores de circuitos analógicos foram desenvolvidos ainda não existiam as interfaces gráficas. Portanto, os circuitos eram descritos por meio de um código textual.

Com o desenvolvimento das interfaces gráficas, os circuitos analógicos e digitais puderam ser descritos por um desenho esquemático.

Porém, uma Linguagem de Descrição de *Hardware* (HDL) além de servir para descrever textualmente o funcionamento de um circuito elétrico-eletrônico, ainda pode ser usada para outros fins, tais como documentação e vários tipos de verificação.

As HDLs mais comumente utilizadas em circuitos digitais são VHDL, Verilog e SystemVerilog (uma extensão de Verilog). Muito já se discutiu sobre o uso de cada uma delas e mesmo sobre qual delas seria a melhor linguagem. No momento, parece existir um consenso de que é necessário que um bom projetista deva ter um conhecimento mínimo sobre todas elas.

O grupo já tinha um conhecimento básico de VHDL e verificou que, para a utilização do *kit* baseado em FPGA, seria útil aprender Verilog e SystemVerilog.

1.1.3 Simulador *ModelSim*

Existem diversos aplicativos computacionais dedicados ao projeto de circuitos e sistemas digitais.

O grupo tem experiência com o ambiente de desenvolvimento integrado (IDE) Max+plus II [Alta], da fabricante Altera [Int], e a sua evolução, denominada de Quartus II [Altb].

Por sua vez, uma outra ferramenta muito utilizada nesta área de projetos é o simulador ModelSim [Gra15], disponibilizado pela empresa Mentor Graphics [Men]. O ModelSim é apto a reconhecer circuitos digitais descritos em VHDL, Verilog e SystemVerilog. Utilizando-se tais linguagens, é possível também descrever sinais de excitação para os circuitos, realizar simulações e observar o seu comportamento.

Dada a larga aceitação do simulador ModelSim em ambientes de projeto e o seu total desconhecimento por parte dos integrantes do PET-Tele, o grupo decidiu realizar um grupo de estudos sobre tal ferramenta.

1.2 Objetivo

O objetivo deste tutorial é servir de documento básico para os primeiros passos no uso do simulador ModelSim.

Inicialmente, é mostrado como adquirir, como fazer o *download* do simulador e como lincenciá-lo. Em seguida, são descritas algumas de suas funcionalidades. Por fim, são apresentados alguns exemplos de código, usando as linguagens VHDL e Verilog.

2 *Download*, instalação e licenciamento do ModelSim

A seguir, são brevemente descritos os procedimentos para *download*, instalação e licenciamento do simulador ModelSim.

2.1 *Download*

O ModelSim possui uma versão gratuita para estudantes. Para fazer o seu *download*, deve-se preencher um formulário destinado a iniciar a requisição da licença de estudante. O formulário pode ser acessado por meio da seguinte *URL*:

```
https://www.mentor.com/products/request?&fmpath=/company/higher_ed/
modelsim-student-edition-eval&id=c3694f2b-35f0-48a7-bdcd-efd77417ded0
```

O primeiro passo é preencher seus dados para efetuar o cadastro, como pode ser visto na Figura 1. Quando preenchidos todos os campos, basta clicar em "*Submit*".

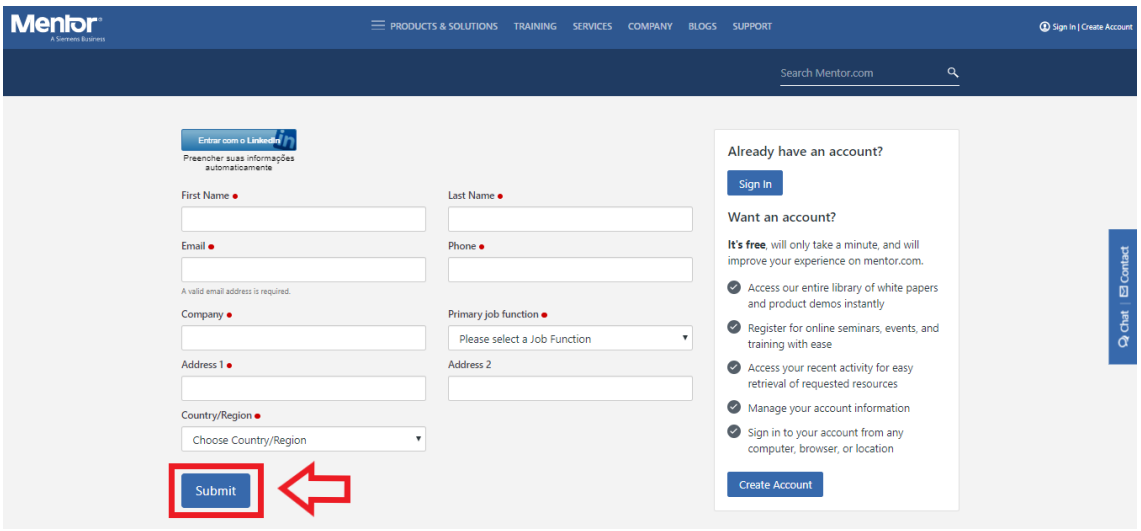

Figura 1: Página de cadastro da Mentor Graphics.

Depois, basta concordar com o acordo de licença, clicando no botão no fim da página apontado na Figura 2.

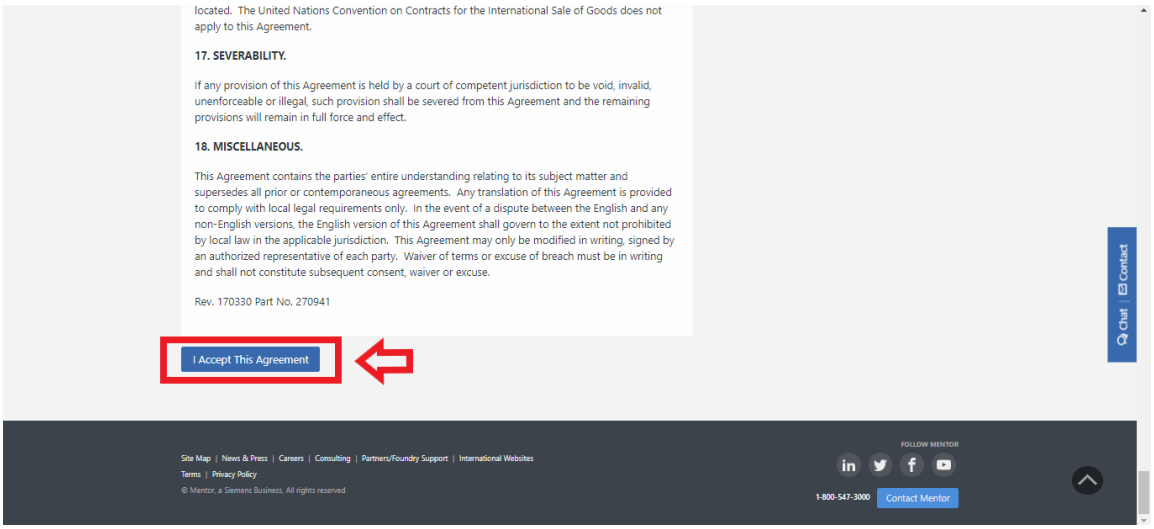

Figura 2: *License Agreement - Mentor Graphics.*

Feito isso, o usuário será redirecionado para a página de requisição de uma URL para o *download* do *software ModelSim*, como mostra a Figura 3.

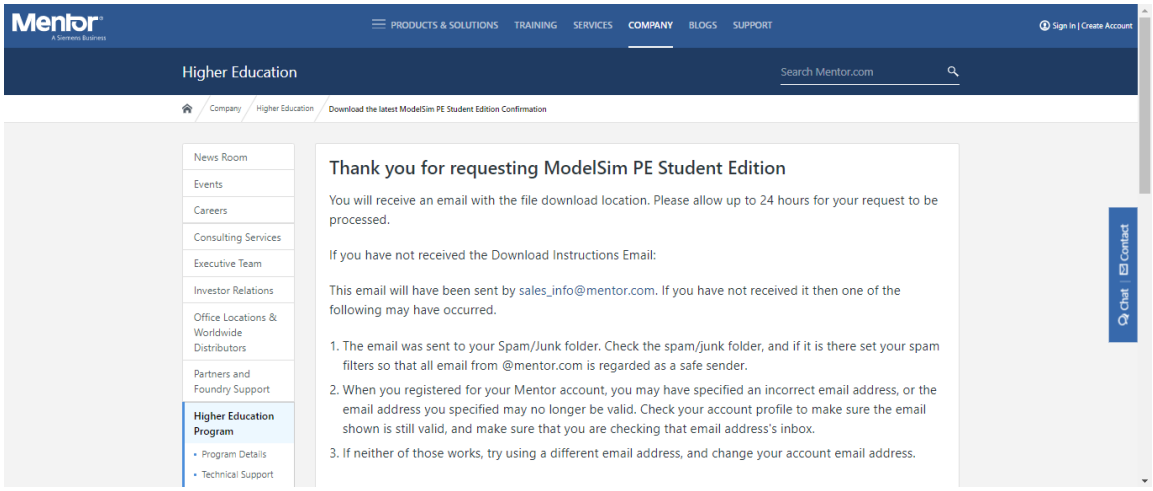

Figura 3: Página de requisição de uma URL para o *download*.

O próximo passo é acessar o endereço de *e-mail* cadastrado pelo usuário e procurar pela mensagem enviada pela Mentor Graphics. O corpo do mensagem será como o da Figura 4. Vale lembrar de checar a caixa de *spam*, pois a mensagem pode ter sido automaticamente arquivada em tal pasta.

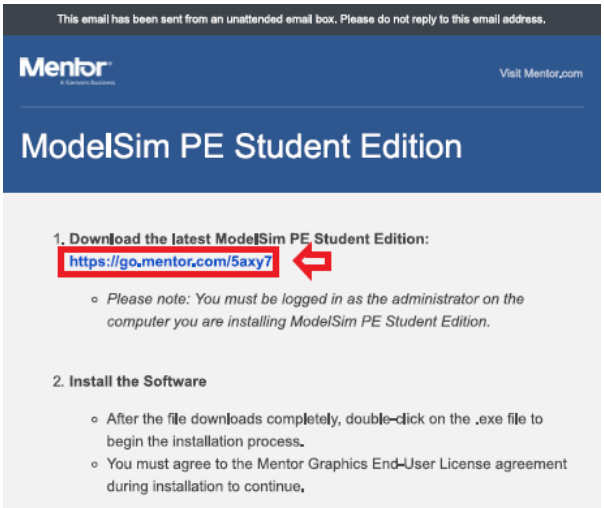

Figura 4: Mensagem enviada pela Mentor Graphics.

Uma vez localizada a mensagem, basta clicar na URL presente nela para que o *download* seja iniciado. Uma notificação semelhante à da Figura 5 será aberta, dependendo do seu navegador.

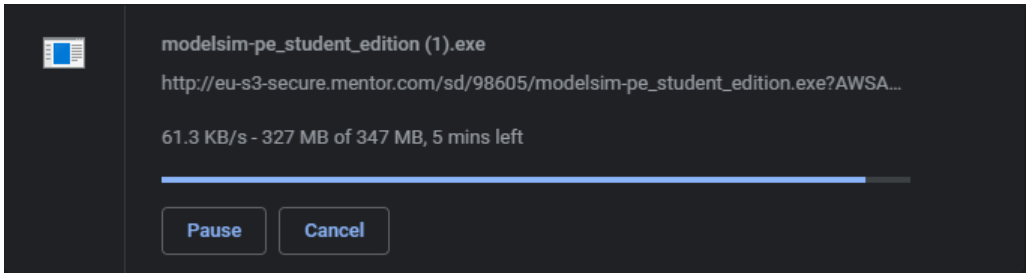

Figura 5: Notificação de *download* em andamento.

2.2 Instalação

Após concluído o *download*, o usuário deve inicializar o instalador. A janela da Figura 6 será aberta.

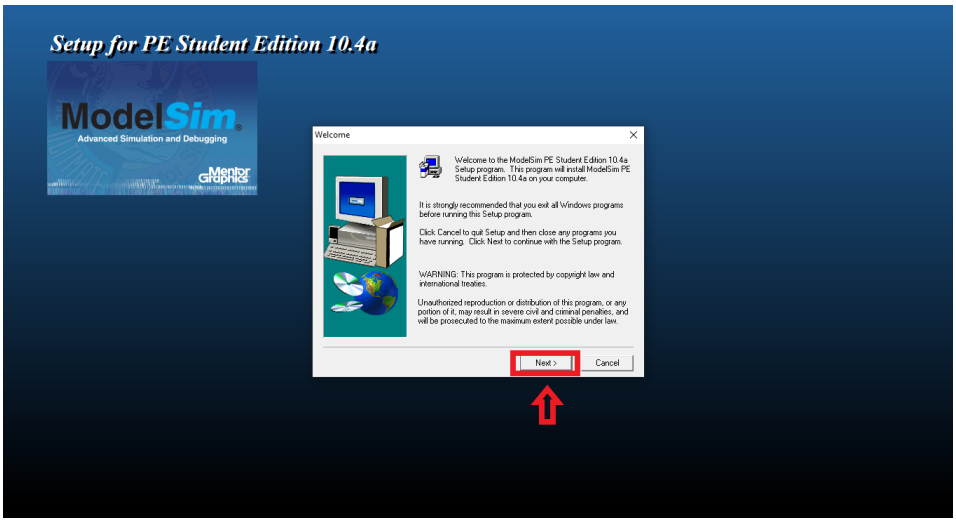

Figura 6: Primeiro passo para instalação do programa.

Clique em "*Next*" para prosseguir com a instalação. A janela da Figura 7 será aberta.

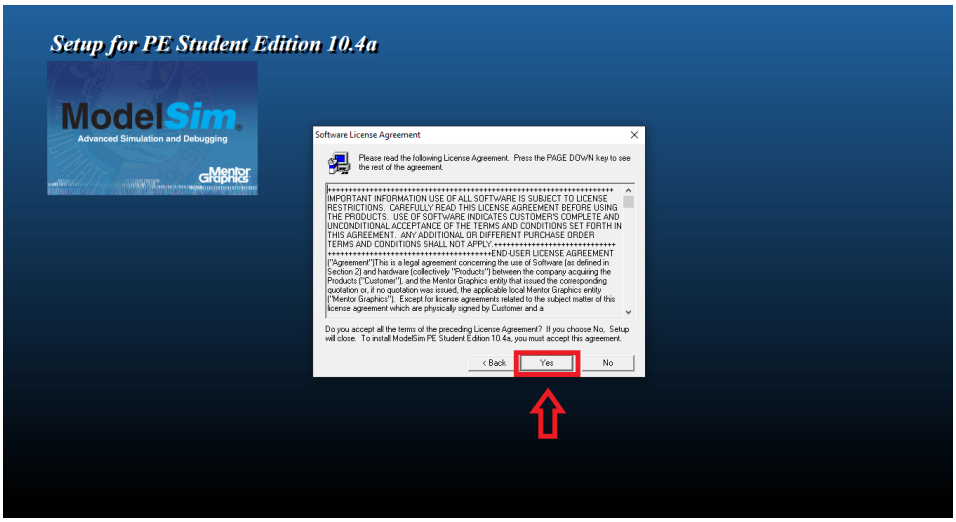

Figura 7: Segundo passo para instalação do programa.

Clique em "*Yes*" para prosseguir com a instalação. A janela da Figura 8 será aberta.

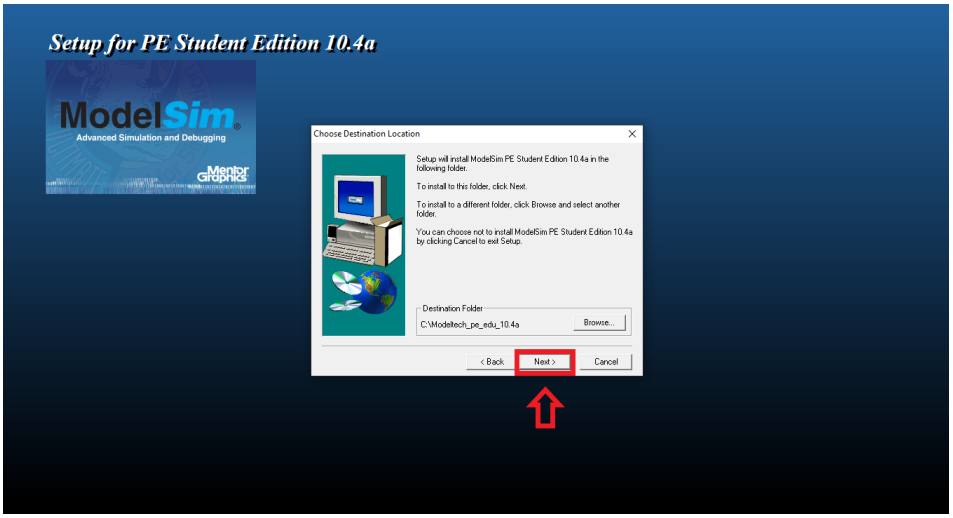

Figura 8: Terceiro passo para instalação do programa.

Clique em "*Next*" para prosseguir com a instalação. A janela da Figura 9 será aberta.

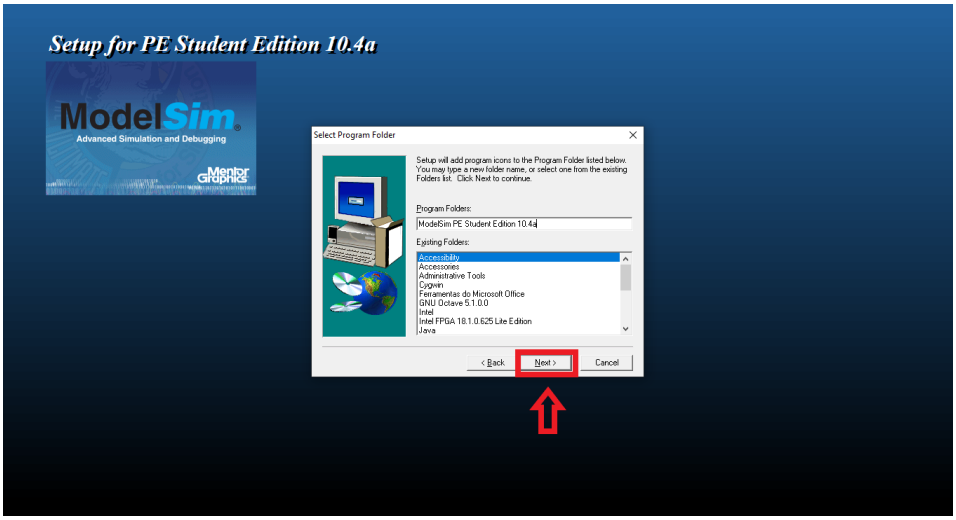

Figura 9: Quarto passo para instalação do programa.

Clique em "*Next*" para prosseguir com a instalação. A janela da Figura 10 será aberta.

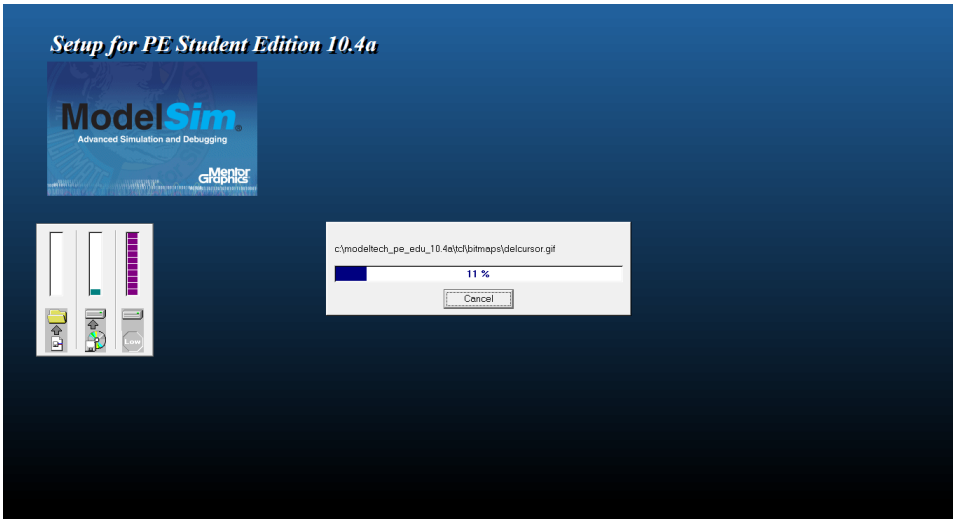

Figura 10: Quinto passo para instalação do programa.

Aguarde o processo de instalação terminar. Após o fim do processo, a janela da Figura 11 será aberta.

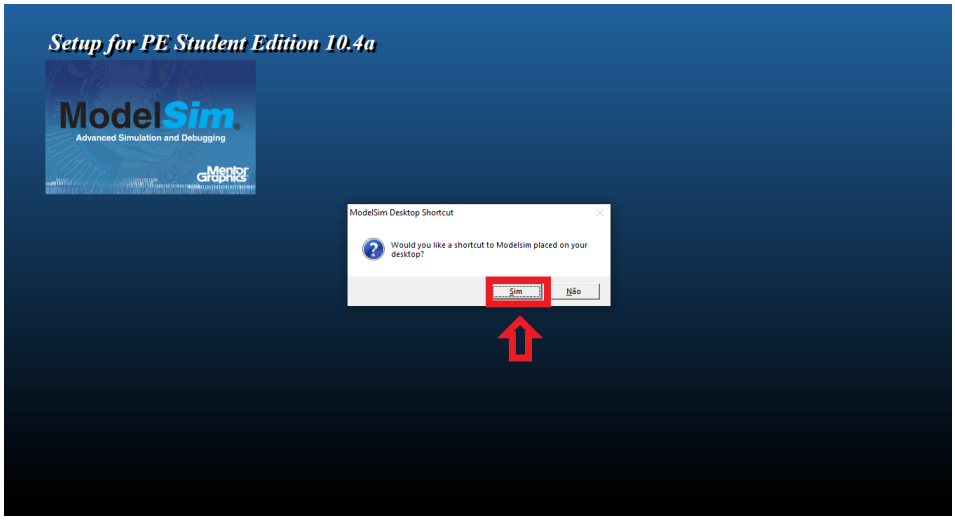

Figura 11: Sexto passo para instalação do programa.

Aperte “Sim” para prosseguir com a instalação. A janela da Figura 12 será aberta.

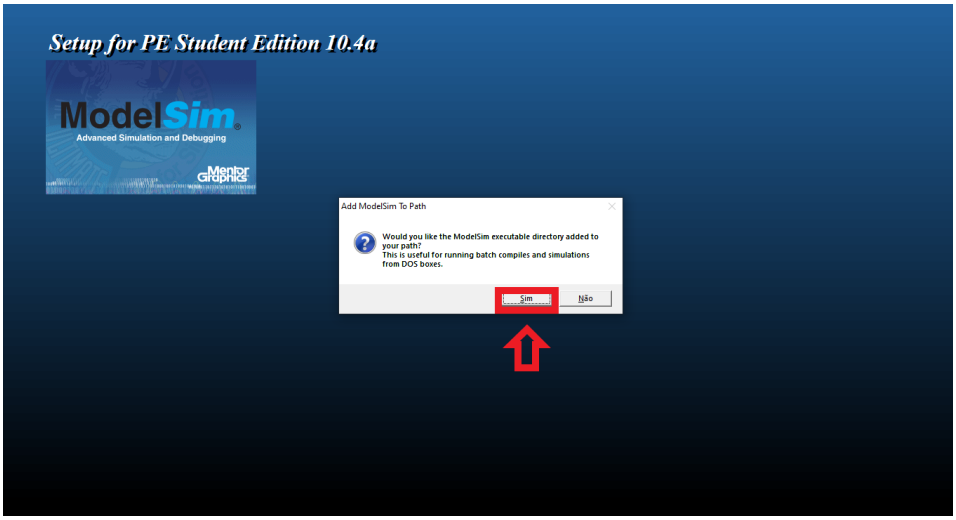

Figura 12: Sétimo passo para instalação do programa.

Clique em "Sim" para prosseguir com a instalação. A janela da Figura 13 será aberta.

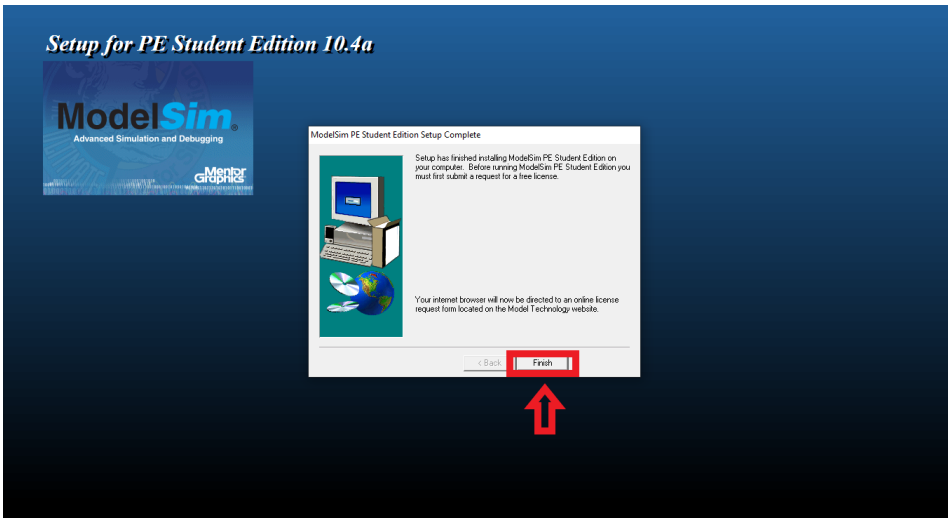

Figura 13: Oitavo passo para instalação do programa.

Após o instalador iniciar o breve processo de configuração do ambiente, o usuário deverá clicar em "*Finish*" para finalizar o processo de instalação.

Terminada a instalação, o usuário será redirecionado para uma *webpage* de cadastro, com a finalidade de requisitar a licença de estudante, sem a qual o programa não poderá ser utilizado. Essa etapa é discutida a seguir.

2.3 Licenciamento

Terminada a instalação, o usuário será redirecionado para uma *webpage* de cadastro, similar àquela da Figura 14, a fim de realizar o preenchimento dos dados do usuário, necessários à requisição de uma licença. Após o preenchimento dos dados, o usuário deve clicar em "*Request License*".

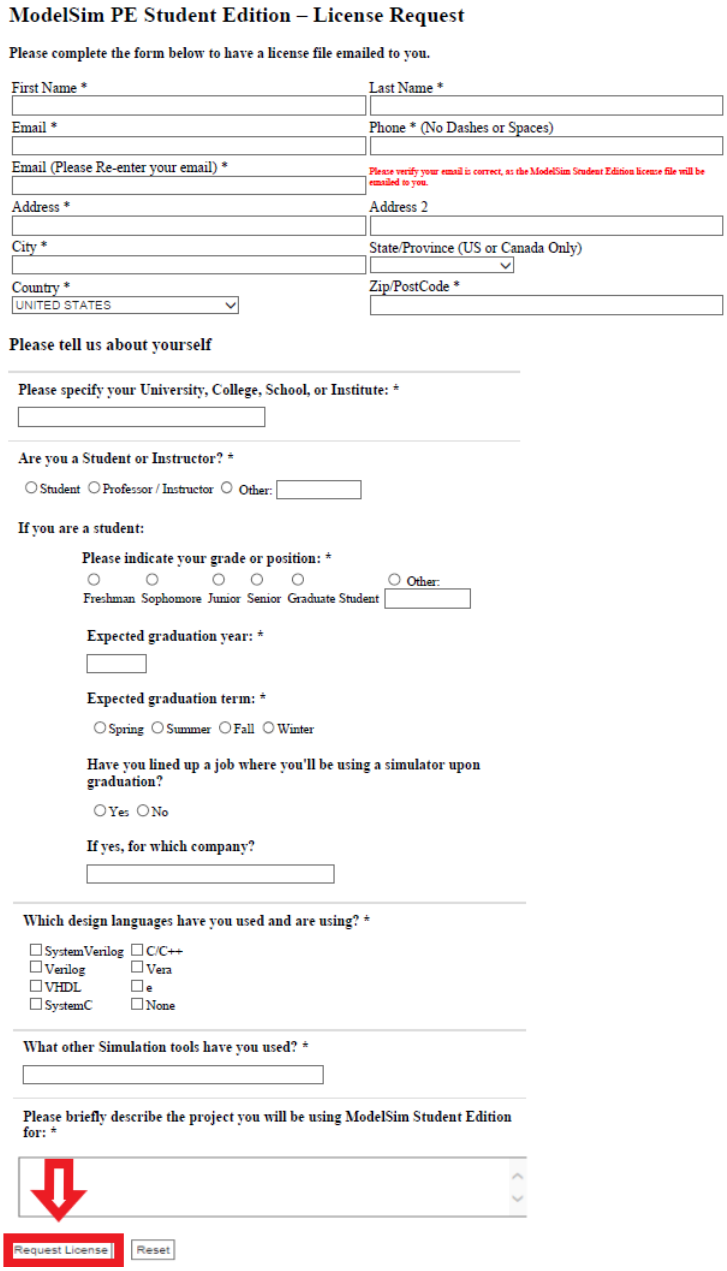

Figura 14: *Webpage* de licenciamento do programa.

Feito isso, o usuário deverá ser redirecionado para uma *webpage* idêntica à da Figura 15. A única diferença deverá ser o endereço de *e-mail* informado, para qual será enviado o arquivo da licença. Caso qualquer outra *webpage* seja exibida, o processo de licenciamento falhou e o usuário deverá fazer uma nova requisição. Para isto, deverá realizar a etapa de instalação outra vez, pois qualquer tentativa de requisição de licença que não seja através da URL fornecida ao final da instalação (após clicar em "*Finish*", mostrado na Figura 13) também não terá êxito.

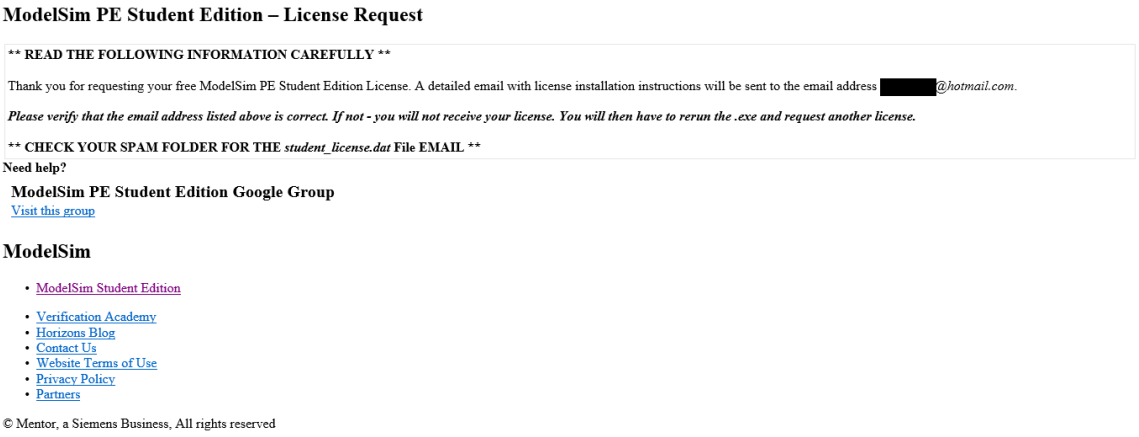

Figura 15: *Webpage* de requisição de licença concedida.

Finalmente, o usuário deverá receber uma mensagem no endereço de *e-mail* informado por ele no formulário da Figura 14. Em anexo, deverá estar presente um arquivo de dados, contendo a senha, como ilustrado na Figura 16. A licença encontrada nesse arquivo é específica para o computador e para a conta que originaram a requisição, sendo impossível substituí-la ou reutilizá-la.

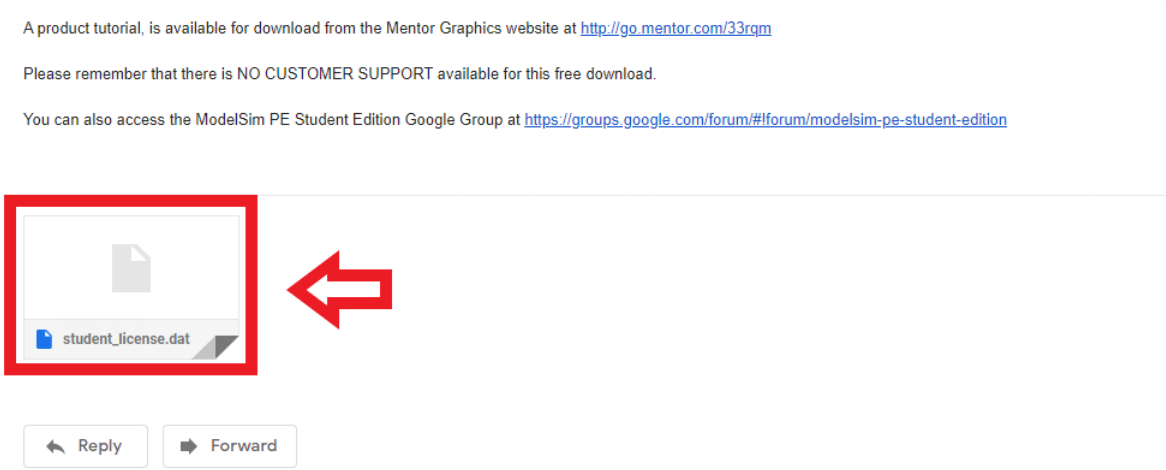

Figura 16: Mensagem com o arquivo da licença anexado.

O arquivo com a licença deve ser baixado e armazenado no seu computador no seguinte diretório:

`C:\Modeltech_pe_edu_<versão_atual>` .

Por exemplo, durante a confecção deste tutorial, a versão mais atual era a 10.4a. Logo, o diretório foi:

`C:\Modeltech_pe_edu_10.4a` .

Feito isso, o processo de licenciamento estará completo e o programa já estará disponível para a utilização.

A licença tem validade de 180 dias, sendo necessária sua renovação após esse período. Para renovar a licença, o usuário deverá acessar uma *webpage* específica, baixar a mais nova versão

do simulador e realizar o processo de instalação novamente. A *URL* de acesso à *webpage* de renovação de licença é a seguinte:

`https://www.mentor.com/company/higher_ed/modelsim-student-edition`

3 Ferramentas básicas do ModelSim

O ModelSim é composto por duas estruturas, utilizadas para elaborar e simular esquemas, que são as bibliotecas (*libraries*) e os projetos (*projects*). É possível, através de tais estruturas, simular o esquema elaborado pelo usuário de três diferentes formas, sendo elas: *Basic Simulation Flow*, *Project Flow* e *Multiple Library Flow*. Apenas as duas primeiras serão objeto de estudo deste documento, visando também mostrar ao leitor em qual tipo de situação é mais conveniente o uso de cada uma. Serão apresentadas, ainda, ferramentas que permitem gerar e visualizar formas de ondas.

3.1 Introdução à interface gráfica

Tendo concluído a instalação do ModelSim, basta iniciar o programa selecionando o ícone. A interface inicialmente exibida é mostrada na Figura 17. Nela, pode ser observada a presença da aba *Library*, com diversas pastas contendo vários arquivos. Essas pastas são as bibliotecas (*libraries*), as quais exercem dois tipos de função no ModelSim. Por um lado, elas podem conter a versão compilada de um circuito descrito por uma HDL, o que é denominado de *design* do circuito. Além disso, elas podem servir de fonte para uso de mais recursos, como o uso de bibliotecas em linguagens de programação. Aqui não será tratada qualquer das bibliotecas mostradas na Figura 17.

A seguir, são apresentadas algumas ferramentas básicas de uso.

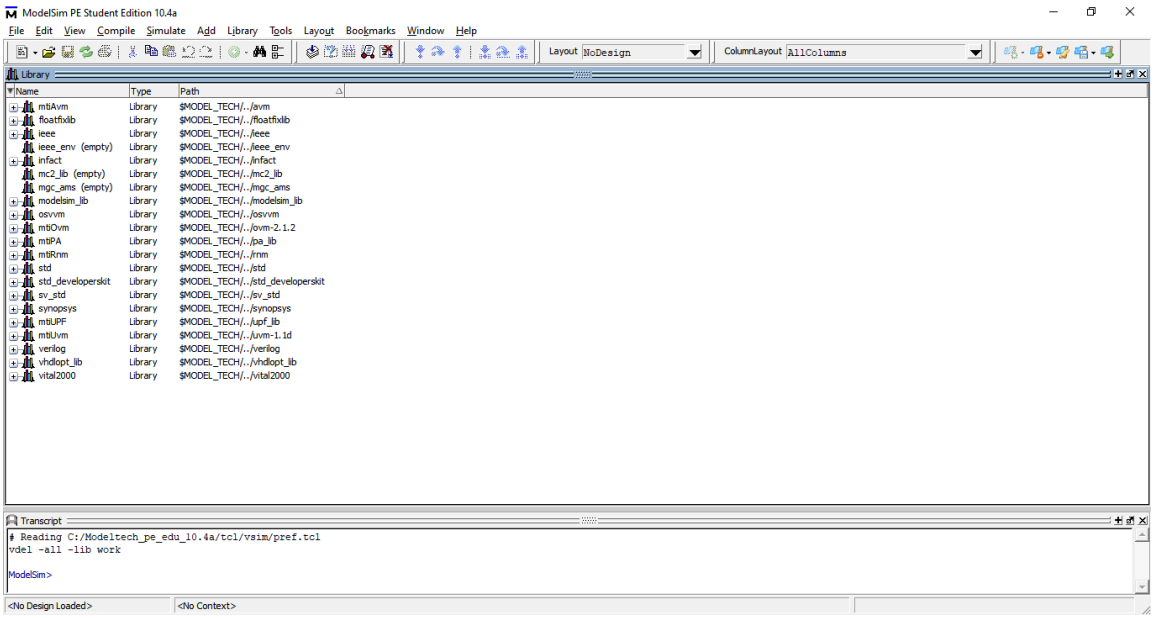

Figura 17: Interface inicial do ModelSim.

Para criar editar um novo documento em uma HDL específica, basta ir ao menu superior, selecionar *File* → *New* → *Source* e escolher a HDL desejada.

Para salvar o novo arquivo, basta ir para *File* → *Save*, tendo em vista a aba do documento selecionado.

Para abrir um arquivo, basta ir para *File* → *Open* e escolher o arquivo desejado para ser aberto.

Conforme ilustrado na Figura 18, há ícones que servem de atalhos para as operações descritas acima.

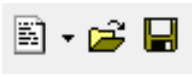

Figura 18: Da esquerda para a direita: *New* (criar arquivo), *Open* (abrir arquivo) e *Save* (salvar arquivo).

É possível observar que o ambiente gráfico do ModelSim pode possuir diversas abas e, no canto superior esquerdo delas, há alguns ícones que mudam a exibição/comportamento da mesma para facilitar o trabalho do usuário. Tais ícones são exibidos na Figura 19.

Figura 19: Da esquerda para a direita: *Zoom*/*Unzoom* (ampliar ou comprimir), *Dock*/*Undock* (destacar ou prender a aba) e *Close* (fechar).

Uma outra ferramenta importante é a de busca, para quando se deseja encontrar uma ocorrência de alguma expressão numa aba, ou documento. O símbolo da mesma é mostrado pela Figura 20.

Figura 20: Ícone de busca.

Ao selecionar o ícone de busca, será aberta uma barra de procura na parte inferior, onde estão as abas. A barra de procura é mostrada pela Figura 21.

Figura 21: Barra de procura. Ela exibe cada ocorrência do termo buscado.

Algumas outras ferramentas, que são relacionadas à criação de documentos de edição, à compilação e à simulação, serão mostrados nas seções seguintes.

3.2 Simulação básica

Nessa parte, será mostrado como fazer uma simulação simples de um circuito. Ela irá envolver apenas uma biblioteca, que será utilizada para armazenar os arquivos relativos à compilação.

O primeiro passo é criar o código do circuito e, em seguida, criar um arquivo de sinais de teste, chamado de *testbench*.

Siga para *File* → *New* → *Source*. Nesse exemplo, será utilizado a linguagem Verilog, mas o procedimento também se aplica às demais linguagens.

Será escrito um código simples, meramente ilustrativo, apenas para realizar um exemplo de simulação. Ele consiste em duas portas AND, com suas saídas ligadas a uma porta OR, sendo uma delas invertida por uma porta NOT. O exemplo pode ser visualizado na Figura 22.

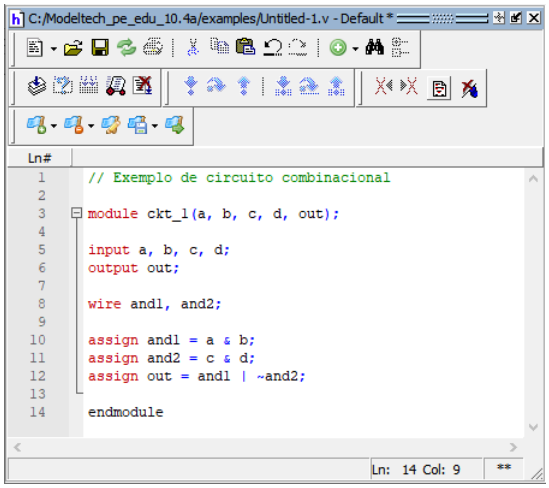

Figura 22: Exemplo de um circuito combinacional codificado em Verilog.

A seguir, será criado um *testbench* para o circuito, seguindo o mesmo processo descrito anteriormente para criar um novo documento de edição. Observe que uma nova aba de edição é criada e na parte inferior, pode-se ver os dois documentos criados e selecionar qual se deseja editar. Isso pode ser observado pela Figura 23.

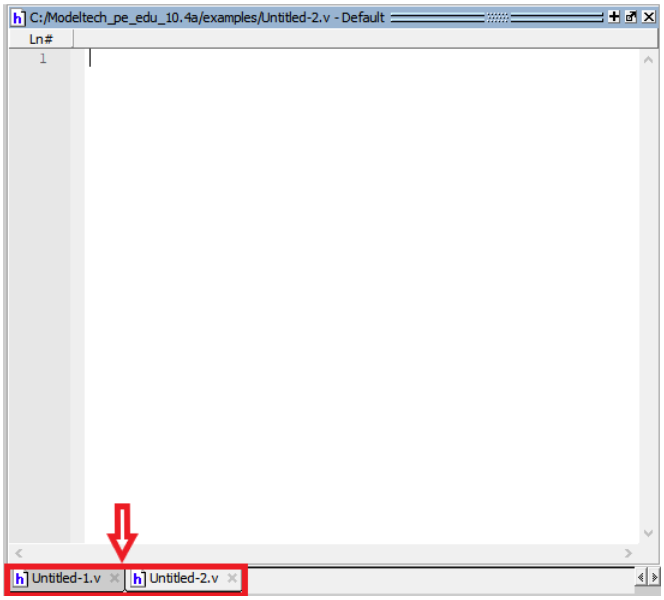

Figura 23: Abas de seleção de documento para edição, contornadas em vermelho.

Com isso, basta apenas criar o *testbench*, conforme é mostrado na Figura 24.

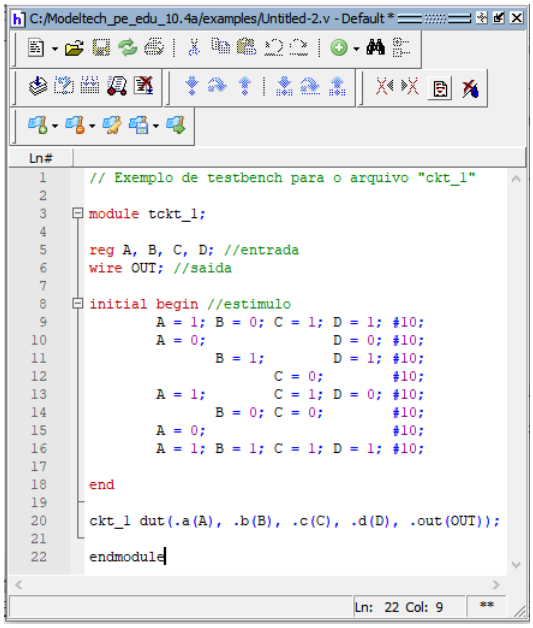

Figura 24: Exemplo de *testbench* para o *design*.

Agora, deve-se criar uma pasta para guardar ambos arquivos e salvá-los. É importante lembrar que, ao selecionar a opção de salvar, estará sendo salvo apenas o arquivo correspondente a aba atual. Deve-se ainda garantir que os arquivos salvos tenham o mesmo nome de seus módulos correspondentes. Assim, o *design* deve ter o nome de "ckt_1" e o *testbench* deve ter o nome de "tckt_1".

Feito isso, feche as abas de edição e vá para *File* → *Change Directory*. Deverá ser exibida uma janela similar a da Figura 25.

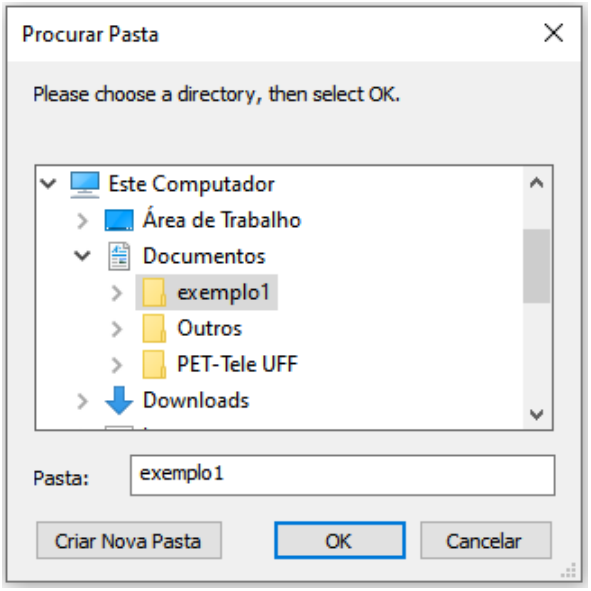

Figura 25: Escolha da pasta que contém os arquivos HDL.

Selecione o diretório que contém os arquivos salvos, que, nesse exemplo, é aquele de nome "exemplo1". Em seguida, vá para *File* → *New* → *Library*, que deverá exibir uma janela como a da Figura 26.

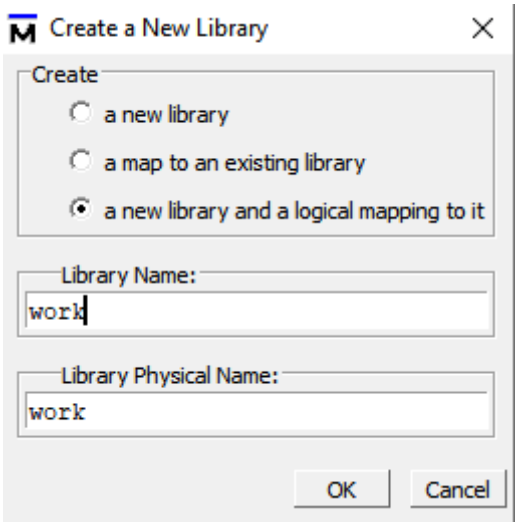

Figura 26: Janela exibida para criar uma nova biblioteca.

Certifique-se de ter a opção *a new library and a logical mapping to it* marcada. Escolha o nome que for desejado para a nova biblioteca. Por padrão, aparece o nome *work* e, nesse exemplo, ele será mantido. Logo em seguida, observe na aba de *Libraries*, que uma nova biblioteca foi criada. Porém, ela está vazia, como mostrado na Figura 27.

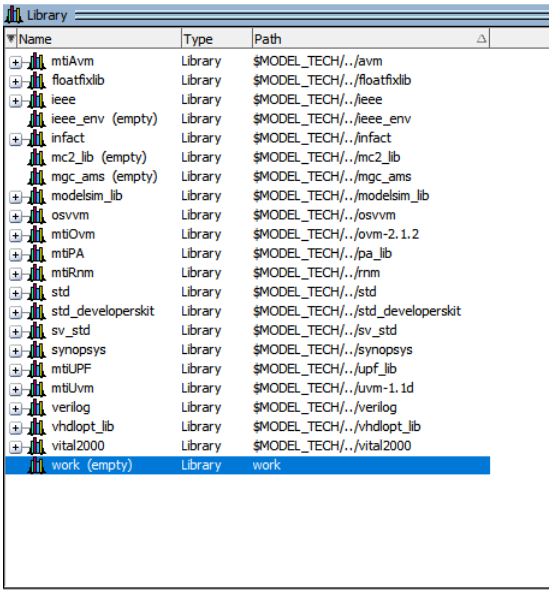

Figura 27: Biblioteca criada em destaque na aba *Library*.

O próximo passo é compilar os arquivos que estão na pasta criada. Vá para *Compile* → *Compile* e selecione ambos os arquivos, certificando-se que eles possuam a extensão correspondente à HDL utilizada. Selecione a opção *Compile*. Após a compilação, selecione a opção *Done* e verifique se não houve erros. Tal procedimento é mostrado pela Figura 28.

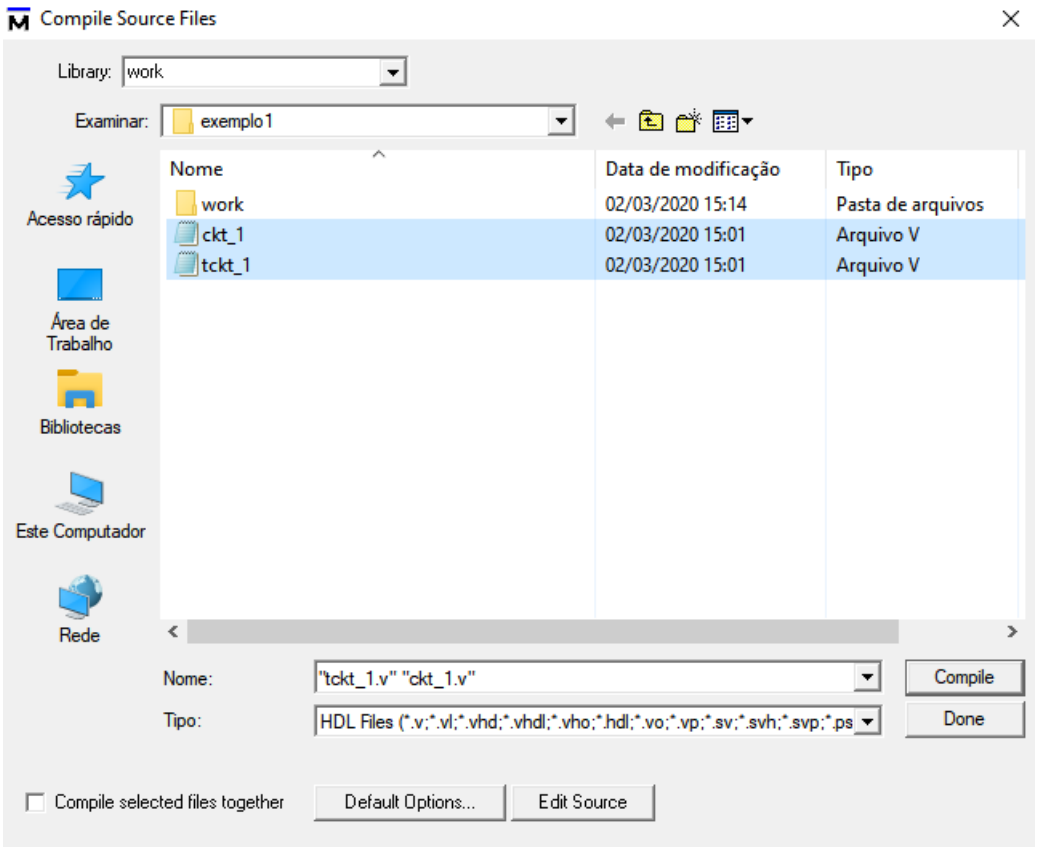

Figura 28: Seleção dos arquivos a serem compilados. Tanto o *design* como o *testbench*.

Observe a biblioteca criada para o projeto, que, nesse exemplo, é a "*works*". Verifique que há o sinal "+" no seu canto esquerdo. Ao clicar no sinal, serão exibidos os arquivos resultantes da compilação do *design* e do *testbench*. Logo em seguida, clique duas vezes no arquivo referente a compilação do *testbench*, que, no caso, é o "tckt_1". Deverá abrir uma interface para a simulação, como mostrado na Figura 29.

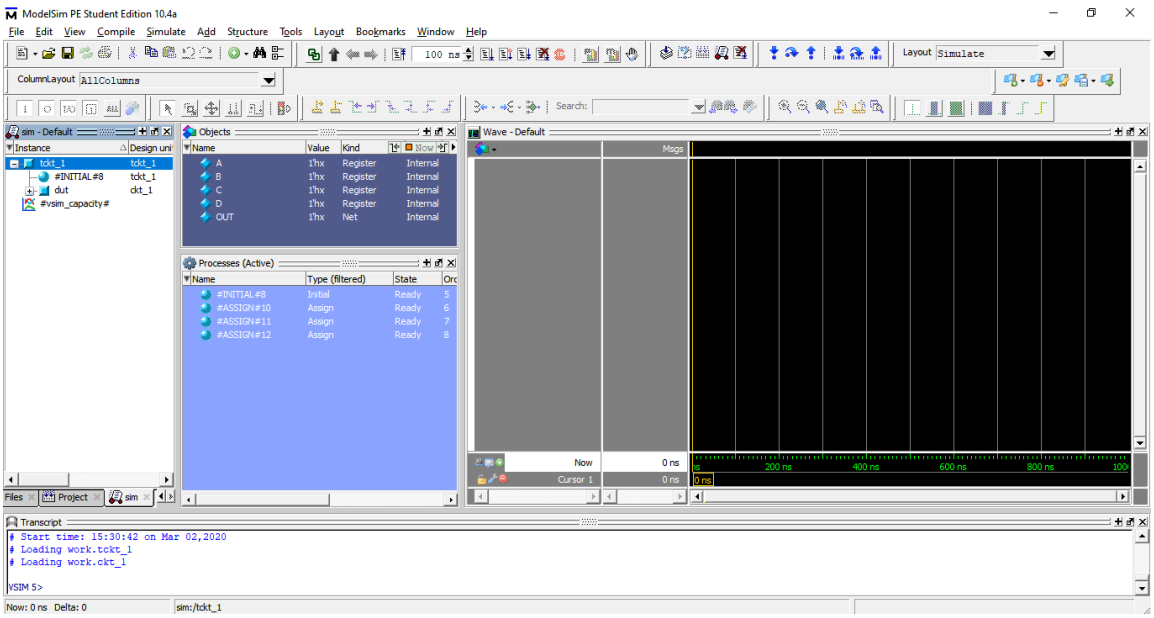

Figura 29: Interface da simulação, com quatro janelas: *Simulation*, *Objects*, *Processes* e *Wave*.

Na aba de *Sim* pode ser observado o arquivo compilado com suas instanciações. Aperte com o botão direito do *mouse* sobre o nome do arquivo compilado (nesse caso "tckt_1") e escolha a opção *Add Wave*. Após esse processo, poderá ser visto na aba *Wave* cada componente correspondente do projeto.

Resta apenas realizar a simulação. Para isso, vá para *Simulate* → *Run* → *Run 100*. O resultado será a forma de onda gerada na Figura 30, onde o gráfico foi ampliado.

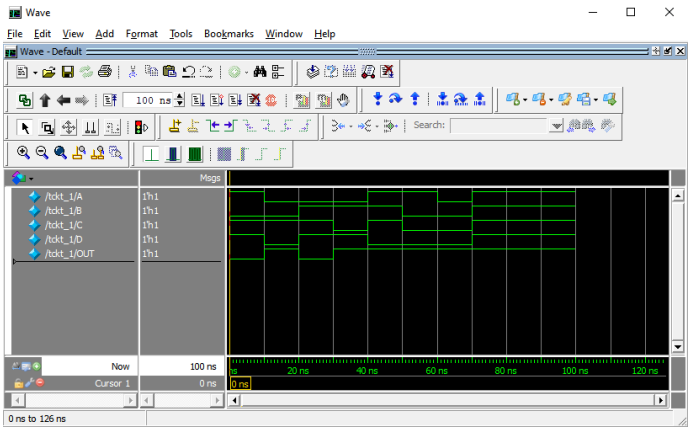

Figura 30: Forma de onda obtida após a simulação.

Na Seção 3.4, são mostradas algumas ferramentas específicas para observar as formas de onda ou facilitar a visualização das mesmas.

Para encerrar a simulação, basta ir para *Simulation* → *End Simulation*.

3.3 Simulação com projetos

Em *designs* que contenham diversos módulos e bibliotecas, convém a utilização de projetos (*projects*). No ModelSim, por padrão, um projeto contém uma biblioteca para compilação e um arquivo com a extensão "mpf", que armazena informações da sessão de uso. Ele pode ter, ainda: arquivos ou pastas com códigos HDL; documentos do tipo *Readme*, dando explicações do projeto; bibliotecas locais e referências a bibliotecas globais.

Na demonstração que se segue, serão aproveitados ambos os arquivos criados na Seção 3.2, denominados "ckt_1"(Figura 22) e "tckt_1"(Figura 24). Esses arquivos serão salvos em uma nova pasta, com o nome "teste_projeto". Em seguida, será iniciado o ModelSim.

Com a tela inicial do programa aberta, vá para *File* → *Change Directory* e escolha a pasta criada de nome "teste_projeto". Após esse passo será criado o novo projeto, vá para *File* → *New* → *Project*. A janela aberta é indicada pela Figura 31. Dê o nome que for desejado ao projeto criado (nesse exemplo foi dado o nome "teste") e escolha onde o novo projeto deverá ser criado em sua máquina. Em seguida, selecione "OK".

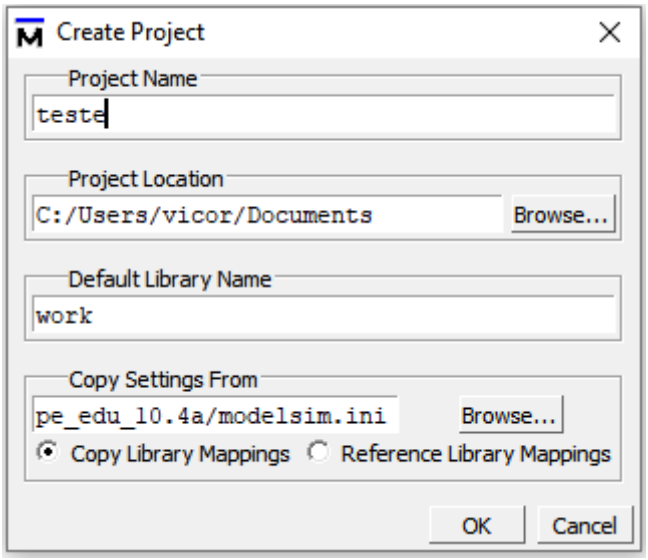

Figura 31: Janela para criação de um projeto. Dê um nome ao mesmo (nesse caso "teste") e escolha o local onde ele deverá ser criado.

Uma nova janela deve ser aberta, ao iniciar o projeto, indicando quatro possíveis opções: *Create New File* (criar um novo arquivo), *Add Existing File* (adicionar um arquivo existente), *Create Simulation* (criar uma simulação) e *Create New Folder* (criar uma nova pasta). Isso é ilustrado na Figura 32. Escolha a opção *Add Existing File*.

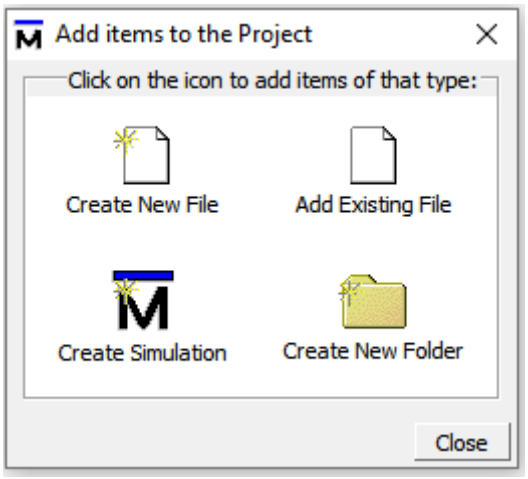

Figura 32: Janela exibida ao se iniciar um projeto.

Como é indicado na Figura 33, será aberta uma aba para inserir o nome dos arquivos. Escolha a opção *Browse*, selecione os arquivos criados anteriormente ("ckt_1.v" e "tckt_1") e aperte "OK".

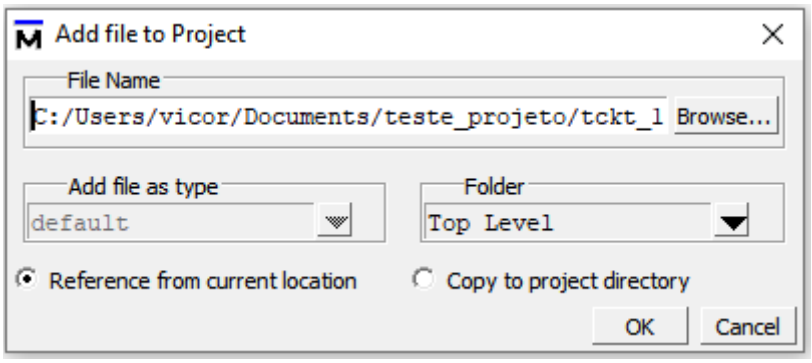

Figura 33: Escolha dos dois arquivos a serem passados para o projeto.

Agora, podem ser observados os arquivos na aba de projetos, como mostrado pela Figura 34. Note que há um sinal de interrogação neles, indicando que os arquivos não foram compilados ainda ou que foram alterados desde sua última compilação correta.

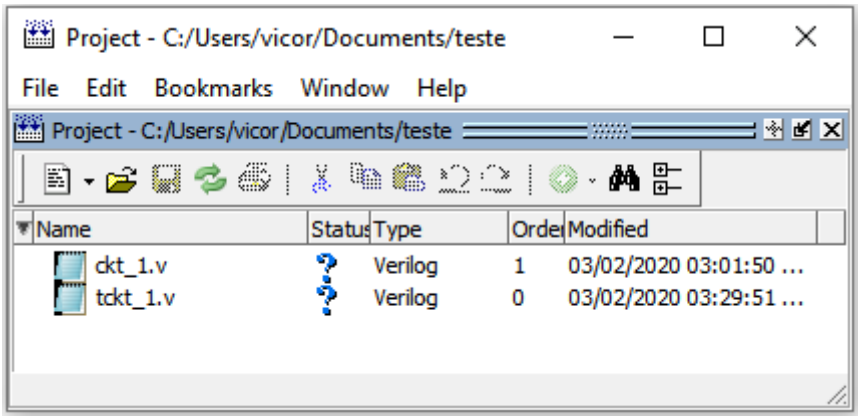

Figura 34: Aba de projetos, com os componentes do atual projeto em exibição. O sinal de interrogação indica que ainda não foram compilados.

Para realizar a compilação dos arquivos importados para o projeto, vá para o menu superior e selecione as opções *Compile* → *Compile All*. Uma forma mais simples de realizar isso é através dos ícones presentes na parte de ferramentas. A Figura 35 mostra alguns ícones e suas funções.

Figura 35: Ícones para compilação. Da esquerda para direita: *Compile* ("Compilar", apenas aplicado ao(s) arquivo(s) selecionado(s)), *Compile Out of Date*, *Compile All* ("Compilar todos", aplica-se a todos os arquivos), *Simulate* ("Simular", aplica-se somente ao(s) arquivo(s) selecionado(s)) e *Break* ("Quebra", interrompe a execução em um determinado ponto).

Sendo assim, execute o ícone *Compile All*. Vale salientar que, caso ocorra algum erro de compilação, o arquivo correspondente será marcado com um "X".

Caso não tenha havido erros de compilação, vá para a aba *Library* e selecione a opção de "+", na pasta *work*, para visualizar os arquivos resultantes da compilação. Clique duas vezes sobre o nome de arquivo "tckt_1", que será carregado e exibido a janela de simulação.

Nesse momento, o processo de análise é o mesmo da seção anterior. Logo, o propósito de mostrar a compilação e a simulação de um projeto foi alcançado. Vá para *Simulate* → *End Simulation*, para encerrar a simulação.

Para adicionar mais componentes ao projeto, basta ir para a aba de projetos e clicar com o botão direito sobre o ambiente. Com isso, aparecerá um menu com opções. Vá para *Add To Project* → *New File*. Deverá ser exibida uma janela como a Figura 36, bastando escolher o nome do arquivo a ser criado e a linguagem utilizada (Verilog, VHDL, etc.).

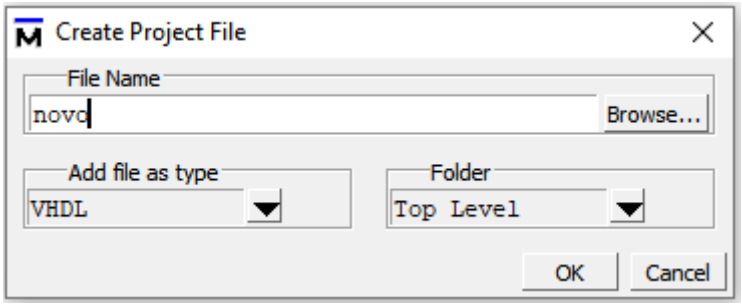

Figura 36: Criando um arquivo de texto dentro do projeto. Devem ser escolhidos o nome do arquivo e a HDL desejada.

Em seguida, o arquivo criado pode ser observado entre os arquivos existentes, com o nome "novo". Isso é mostrado pela Figura 37.

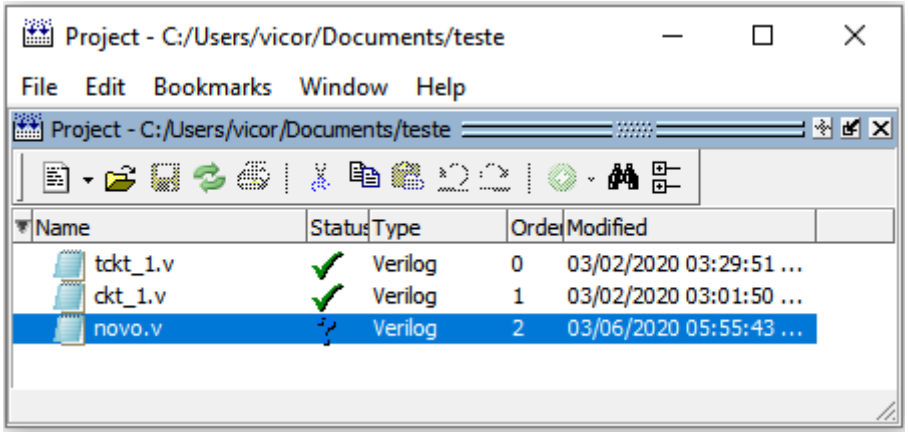

Figura 37: Arquivo de texto criado em destaque, exibido pela aba de projetos.

Para editar algum arquivo, clique com o botão direito no arquivo desejado e vá para *Edit*. Uma janela do editor de texto padrão deverá ser aberta, contendo o arquivo escolhido.

Há também como remover uma pasta ou arquivo do projeto, bastando clicar com o botão direito no mesmo e ir para *Remove From Project*.

Ao se trabalhar com *designs* mais elaborados e contendo um número maior de módulos, uma boa prática de simulação no ModelSim é a criação de diferentes pastas, para facilitar na distribuição dos arquivos segundo suas funcionalidades, evitando que todos dividam um mesmo espaço. Para isso, vá para a aba do projeto e clique com o botão direito em algum espaço em branco. Em seguida, vá para *Add To Project* → *Folder*. Deverá ser exibida uma nova janela, como a mostrada pela Figura 38, onde deverá ser dado um nome a nova pasta.

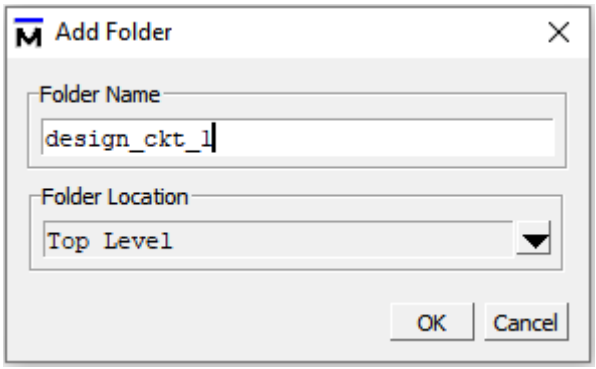

Figura 38: Criando uma nova pasta.

Para adicionar uma sub-pasta, siga o mesmo procedimento descrito anteriormente para adicionar uma nova pasta. Mas, no *Folder Location*, escolha o nome da pasta criada anteriormente ("design_ckt_1"). Em seguida, serão movidos os arquivos para a sub-pasta criada. Selecione os dois arquivos, aperte com o botão direito e vá para *Properties*. A janela aberta pode ser observada pela Figura 39. Vá para o campo *Place in Folder* e escolha o nome da sub-pasta criada, que, no caso desse exemplo, foi "HDL".

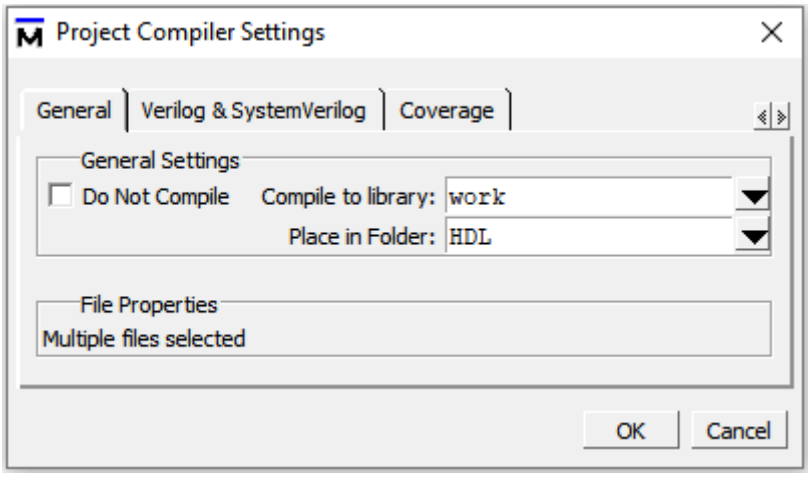

Figura 39: Configurando o local dos arquivos do projeto. Deve ser escolhido em *Place In Folder* o sub-diretório criado.

A local dos arquivos pode ser observado apertando-se "+" nas duas pastas criadas, como mostrado pela Figura 40.

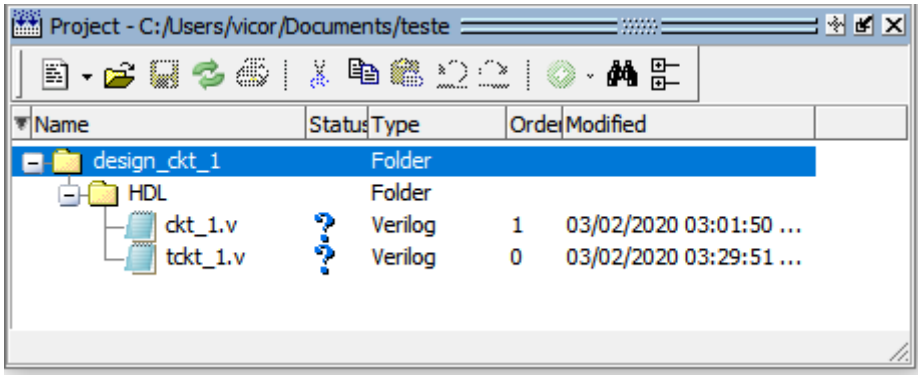

Figura 40: Os arquivos foram movidos para a pasta criada. Observe que o sinal de interrogação aparece novamente, sendo necessário compilar novamente.

3.4 Ferramentas para visualizar formas de onda

Nas seções anteriores, foi mostrado como realizar a compilação e a simulação de um dado *design*. Nessa seção, serão mostradas ferramentas relacionadas à visualização de formas de onda. Para isso, serão aproveitados os arquivos "ckt_1.v" e "tckt_1.v", dados pelas Figuras 22 e 24, respectivamente. Após a simulação dos mesmos, abordada nas seções anteriores, obteve-se a forma de onda ilustrada na Figura 41.

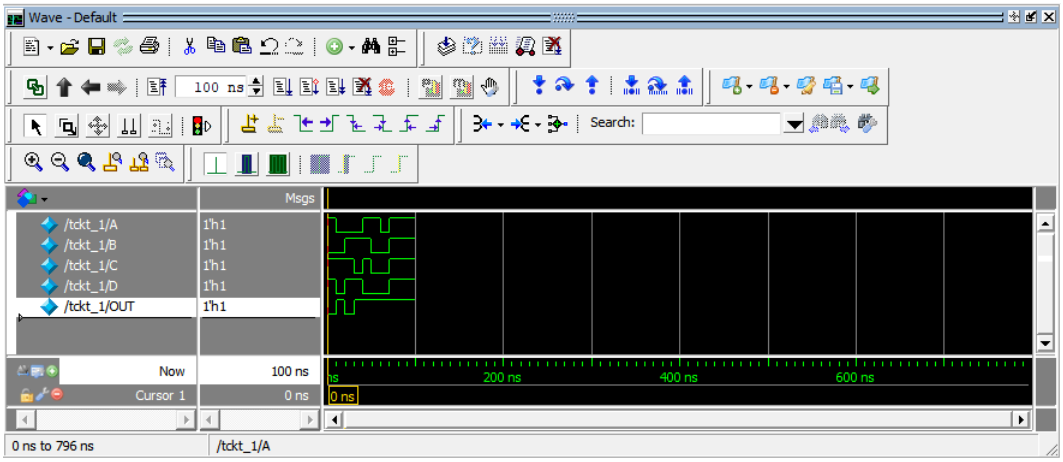

Figura 41: Forma de onda obtida após a simulação dos arquivos "ckt_1.v" e "tckt_1.v".

Diretamente após a simulação, as formas de onda obtidas podem apresentar um problema de escala, de tal forma que não seja possível analisá-las de modo adequado.

O primeiro passo, para facilitar o estudo de uma forma de onda, é destacar a aba *Wave*, conforme ilustrado pela Figura 19. Em seguida, pode-se ampliar a forma de onda obtida, o que pode ser feito através do ícone de ampliação, localizado no canto inferior esquerdo da barra de ferramentas.

Os três ícones relacionados com o escalamento de uma forma de onda são identificados na Figura 42. Nesse exemplo, será usado o *Zoom Full*, embora também pudesse ser usado o *Zoom In*.

Figura 42: Ícones para escalamento de forma de onda. Da esquerda para a direita: *Zoom In* (ampliar), *Zoom Out* (comprimir), *Zoom Full* (ampliar, de modo que ocupe todo o espaço).

Esse mesmo tipo de operação pode ser feita indo para *View* → *Zoom*, onde são exibidas diversas opções diferentes para escalar a visualização. Uma opção útil é a de restaurar a escala anterior. Isso pode ser feito através de *View* → *Zoom* → *Zoom Last*.

Ao clicar em algum ponto da forma de onda, observa-se um cursor que indica o instante da simulação naquele ponto, como pode ser visto na Figura 43. Alguns detalhes podem ser observados nessa figura. No canto esquerdo, há uma referência para cada forma de onda, onde encontra-se selecionada a forma de onda "/tckt_1/C". Deve-se notar, ainda, que são mostrados os valores que cada uma delas assume no instante especificado.

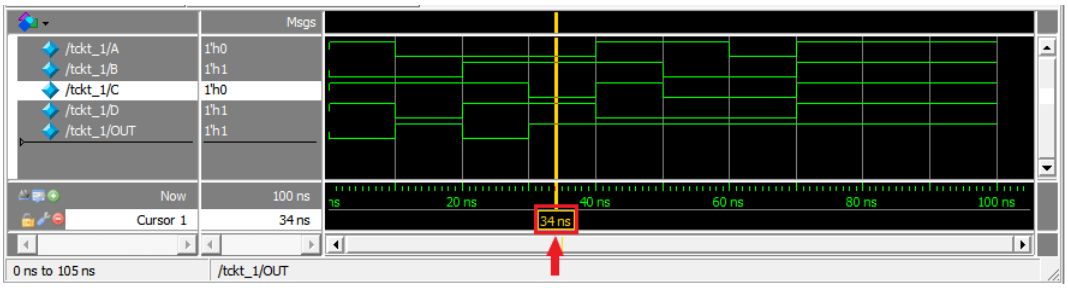

Figura 43: Identificação de um ponto da simulação. O cursor é apresentado por um filete amarelo, enquanto o instante é destacado em vermelho.

Algumas ferramentas úteis, que utilizam o cursor e operam sobre a forma de onda selecionada, ajudam a encontrar as mudanças de valores de "0" para "1" e vice-versa. Elas são ilustradas na Figura 44,

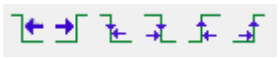

Figura 44: Ícones de borda de onda. Da esquerda para direita: *Find Previous Transition* (transição anterior), *Find Next Transition* (transição posterior), *Find Previous Falling Edge* (descida anterior), *Find Next Falling Edge* (descida posterior), *Find Previous Rising edge* (subida anterior) e *Find Next Rising edge* (subida posterior).

Duas funcionalidades interessantes são a identificacação e o posicionamento de um cursor. O canto inferior esquerdo da Figura 43 é mostrado na Figura 45. Observe que há dois campos em destaque. No primeiro, consta "Cursor 1" e, no outro, "10 ns" (10 nanossegundos). Ao clicar com o botão direito sobre algum deles, pode-se trabalhar sobre o cursor. No caso do primeiro campo, pode-se dar um nome ao cursor e, no segundo, pode-se posicionar o mesmo no instante desejado.

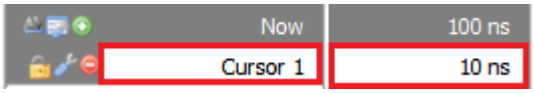

Figura 45: Identificação e posicionamento de um cursor. O campo esquerdo, destacado em vermelho, contendo "Cursor 1", é usado para nomear o cursor. O campo direiro, destina-se a posicioná-lo no instante desejado.

É possível trabalhar com dois indicadores (cursores) ao mesmo tempo. Para adicionar um outro cursor, basta ir em *Add* → *Cursor*. Um atalho para adicionar e remover indicadores, por meio de ícones, é mostrado na Figura 46.

Figura 46: Adição e exclusão de cursores. Da esquerda para direita: *Insert Cursor* (inserir cursor) e *Delete Cursor* (deletar cursor).

A Figura 47 exibe a simulação com o novo cursor. Deve-se observar que é mostrado o intervalo de tempo entre os dois indicadores.

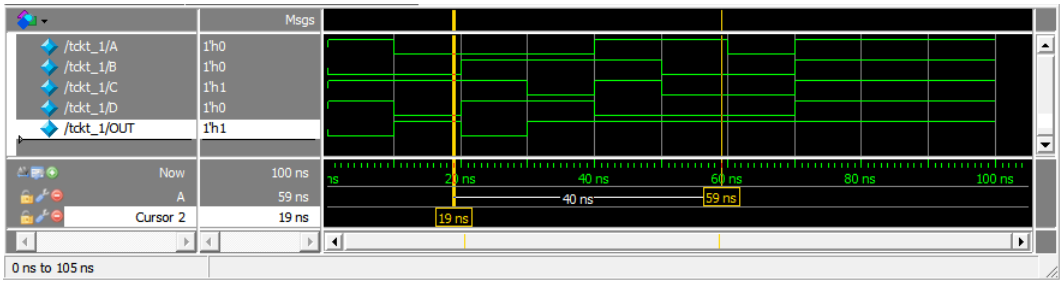

Figura 47: Simulação com dois cursores.

É possível fixar um indicador em um determinado instante. Para isso, basta clicar com o botão direito sobre a forma de onda, no instante desejado, e escolher a opção *Lock* (nome do indicador). Note que a cor do indicador, ao ser fixado, mudará para vermelho. Para liberar um indicador, basta realizar o processo contrário, utilizando a opção *Unlock* (nome do indicador).

Para remover o indicador, basta clicar com o botão direito sobre o indicador e selecionar *Delete* (nome do indicador).

4 Exemplos de simulações no ModelSim

A seguir, são apresentados códigos para descrição e simulação de alguns circuitos digitais comuns, empregando as linguagens VHDL e Verilog. Todos os códigos têm, como arquitetura alvo, o *kit* DE10-Lite, da Terasic, baseado em uma FPGA.

4.1 *Decoder* para *display* de 7 segmentos em Verilog

Um *decoder* (decodificador) é um circuito digital que recebe um conjunto de N *bits* como entrada e retorna um conjunto de M *bits* como saída. Ele possui diversas aplicações, tais como em: memórias, multiplexação e demultiplexação. Nesse exemplo, ele será utilizado para converter uma entrada de 4 *bits*, que representam um número decimal, em uma saída de 8 *bits*, que serão aplicados em um *display* de 7 segmentos. A finalidade do projeto é exibir o número decimal recebido no *display* de 7 segmentos.

Um *display* de 7 segmentos é ilustrado na Figura 48, onde podem-se observar 7 segmentos para formar um dígito numérico decimal e um oitavo para representar um ponto decimal (DP).

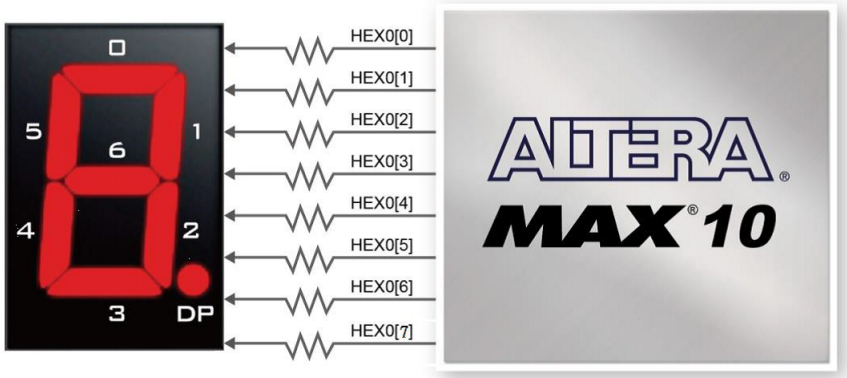

Figura 48: Modelo de *display* de 7 segmentos que foi utilizado para o exemplo. Imagem retirada de [Ter18].

O circuito do *decoder* foi codificado na linguagem Verilog e é listado a seguir. Foi suposto que os *bits* de entrada serão controlados por quatro chaves mecânicas, enquanto os *bits* de saída serão aplicados aos circuitos de controle do *display*, todos presentes no *kit*.

O *decoder* foi descrito utilizando-se um código comportamental. Portanto, foi descrito o funcionamento do circuito e não os componentes lógicos que poderiam compô-lo. Foram declaradas uma entrada *SW* de 4 *bits* e uma saída *HEX1* de 8 *bits* .

A respeito da descrição comportamental, vale lembrar que o bloco de comando *always @** é ativado toda vez que qualquer uma das entradas muda de estado. Por sua vez, o comando *case (SW)* verifica se o valor corrente de *SW* é compatível com algum dos valores do lado esquerdo dos separadores ":". Caso seja, a saída *HEX1* assume o valor correspondente.

```
//////////////////////////////////////
// Exemplo básico de decoder na FPGA
// Arquivo disp_sw.v
//////////////////////////////////////

module disp_sw(

////////////// SEG7 //////////
output  reg [7:0] HEX1,

////////////// SW //////////
input       [3:0]   SW
);

// Design implementation

always @*
case (SW)
        4'b0000 : HEX1 = 8'b11000000;   // 0
        4'b0001 : HEX1 = 8'b11111001;   // 1
        4'b0010 : HEX1 = 8'b10100100;   // 2
        4'b0011 : HEX1 = 8'b10110000;   // 3
```

```
        4'b0100 : HEX1 = 8'b10011001;   // 4
        4'b0101 : HEX1 = 8'b10010010;   // 5
        4'b0110 : HEX1 = 8'b10000010;   // 6
        4'b0111 : HEX1 = 8'b11111000;   // 7
        4'b1000 : HEX1 = 8'b10000000;   // 8
        4'b1001 : HEX1 = 8'b10011000;   // 9

endcase


endmodule

//////////////////////
// Final do arquivo //
//////////////////////
```

Uma vez que se deseja realizar uma simulação antes de configurar a FPGA do *kit*, foi criado uma arquivo *testbench* para testa o código. Ele é apresentado a seguir.

O sinal *sw_in*, de 4 *bits*, simula o conjunto de chaves mecânicas do *kit*. Ao contrário do bloco *always (list)*, que é executado toda vez que houver uma variação no valor de um dos sinais da lista, o bloco *initial* é executado apenas uma vez, no início da simulação. A cada linha do bloco *initial*, o sinal *sw_in* recebe um novo valor, sendo que o valor #50 denota 50 unidades de tempo para continuar a execução. Finalizando o código, há uma instanciação do módulo *disp_sw* sobre o qual se deseja realizar a simulação.

```
////////////////////////////////////////////////////////
// Exemplo simples de testbench para o arquivo disp_sw.v
// Arquivo t_disp_sw.v
////////////////////////////////////////////////////////

module t_disp_sw ();

reg [3:0]   sw_in;
wire [7:0]  hex_out;

initial begin
    sw_in = 4'b0000; #50;    // 0
    sw_in = 4'b0010; #50;    // 2
    sw_in = 4'b0100; #50;    // 4
    sw_in = 4'b0001; #50;    // 1
    sw_in = 4'b0011; #50;    // 3
    sw_in = 4'b1000; #50;    // 8
    sw_in = 4'b1001; #50;    // 9
    sw_in = 4'b0110; #50;    // 6
    sw_in = 4'b0111; #50;    // 7
    sw_in = 4'b0101; #50;    // 5

end
```

```
disp_sw dut (.SW(sw_in), .HEX1(hex_out) );  // instancia do modulo

endmodule

//////////////////////
// Final do arquivo //
//////////////////////
```

A partir dos arquivos acima, realizou-se a simulação no ModelSim, criando-se uma biblioteca para guardar os resultados da compilação de ambos códigos. A forma de onda resultante da simulação é mostrada pela Figura 49.

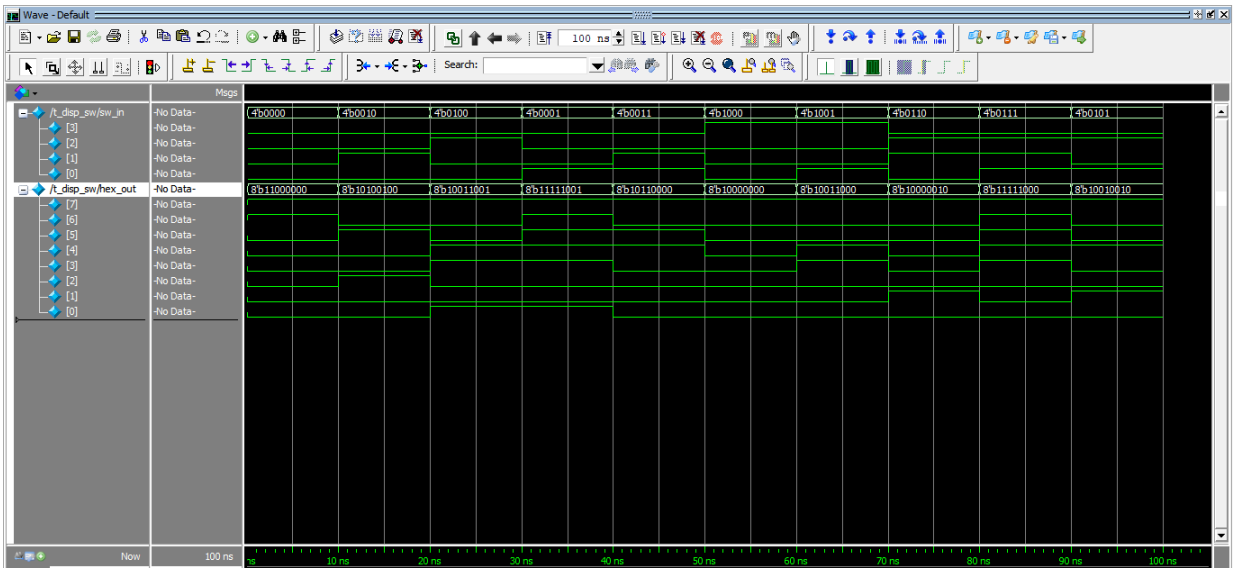

Figura 49: Forma de onda obtida da simulação do *decoder*. São mostrados dois tipos de forma de onda. O que compõe o conjunto de *bits*, mostrado em binário, e as formas de onda individuais de cada componente.

4.2 Multiplexador 4x1

Um multiplexador (MUX) é um circuito combinacional que conta com N entradas, 1 saída e M sinais de controle. O objetivo deste circuito é copiar uma das entradas para saída, de acordo com o controle e o padrão de entradas.

Um MUX 4x1 é um multiplexador que possui 4 entradas (E0, E1, E2 e E3), 2 sinais de controle (C0 e C1) e 1 saída (S). Nesse caso, todos os sinais são de 1 *bit*. Ele é definido por:

$$S = \begin{cases} E0 & , \text{ se } C0 = 0 \text{ e } C1 = 0 \\ E1 & , \text{ se } C0 = 0 \text{ e } C1 = 1 \\ E2 & , \text{ se } C0 = 1 \text{ e } C1 = 0 \\ E3 & , \text{ se } C0 = 1 \text{ e } C1 = 1 \end{cases} .$$

Uma possível implementação para o multiplexador 4x1 definido acima pode ser visto na Figura 50.

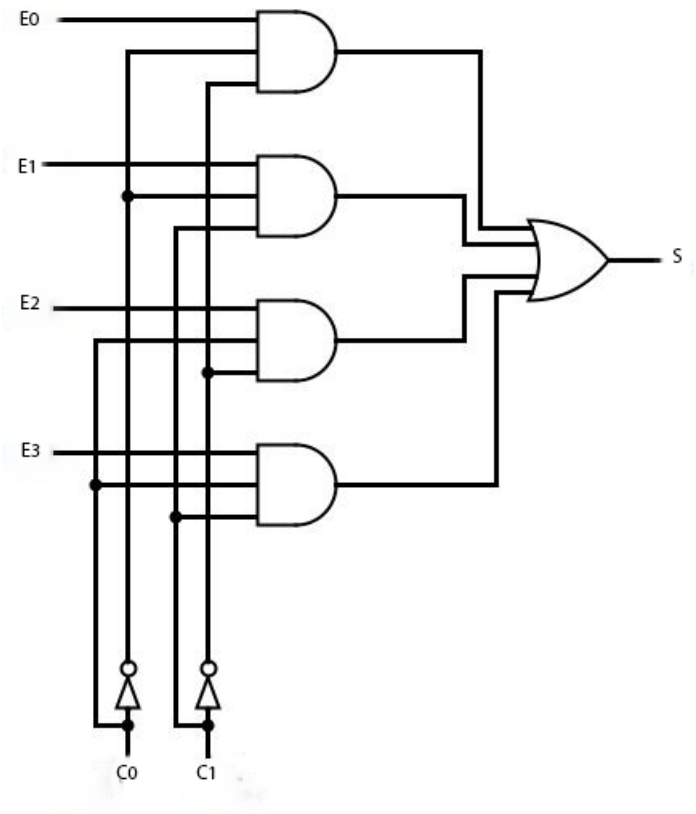

Figura 50: Esquemático de uma possível implementação para o multiplexador 4x1.

4.2.1 Implementação do MUX 4x1 em Verilog

O código Verilog que descreve o MUX 4x1 é listado a seguir.

```
////////////////////////////////
// Exemplo: Multiplexador 4x1
// Arquivo mux41.v
////////////////////////////////
module mux41(S, E0, E1, E2, E3, C0, C1);

// Port declaration
output S;
input E0, E1, E2, E3, C0, C1;

// Wires
wire and0, and1, and2, and3;

// Assign
assign and0 = E0 & ~C0 & ~C1;
assign and1 = E1 & ~C0 &  C1;
assign and2 = E2 &  C0 & ~C1;
assign and3 = E3 &  C0 &  C1;

assign S = and0 | and1 | and2 | and3;
```

```
endmodule

//////////////////////
// Final do arquivo //
//////////////////////
```

Além da implementação do MUX 4x1 no arquivo **mux41.v**, foi criado um arquivo de teste, chamado **tb_mux4x1.v**, com o objetivo de realizar o *testbench* do projeto. Ele é apresentado a seguir.

```
////////////////////////////////////////////////////
// Arquivo de teste para mux41.v
// Nome do arquivo: tb_mux41.v
////////////////////////////////////////////////////

module tb_mux41();

// Port wire declaration
wire S_test;

// Port reg declaration
reg E0_test;
reg E1_test;
reg E2_test;
reg E3_test;
reg C0_test;
reg C1_test;

// Test variables as parameter of mux41 module
mux41 dut(.S(S_test), .E0(E0_test), .E1(E1_test), .E2(E2_test), .E3(E3_test),
.C0(C0_test), .C1(C1_test));

initial begin
    C0_test = 0; C1_test = 0; E0_test = 1; E1_test = 0; E2_test = 0;
    E3_test = 0; #50;
    C0_test = 0; C1_test = 0; E0_test = 0; E1_test = 0; E2_test = 0;
    E3_test = 0; #50;

    C0_test = 0; C1_test = 1; E0_test = 0; E1_test = 1; E2_test = 0;
    E3_test = 0; #50;
    C0_test = 0; C1_test = 1; E0_test = 0; E1_test = 0; E2_test = 0;
    E3_test = 0; #50;

    C0_test = 1; C1_test = 0; E0_test = 0; E1_test = 0; E2_test = 1;
    E3_test = 0; #50;
    C0_test = 1; C1_test = 0; E0_test = 0; E1_test = 0; E2_test = 0;
    E3_test = 0; #50;

    C0_test = 1; C1_test = 1; E0_test = 0; E1_test = 0; E2_test = 0;
```

```
    E3_test = 1; #50;
    C0_test = 1; C1_test = 1; E0_test = 0; E1_test = 0; E2_test = 0;
    E3_test = 0; #50;
end

endmodule

//////////////////////
// Final do arquivo //
//////////////////////
```

O resultado da simulação do MUX 4x1 é mostrado na Figura 51.

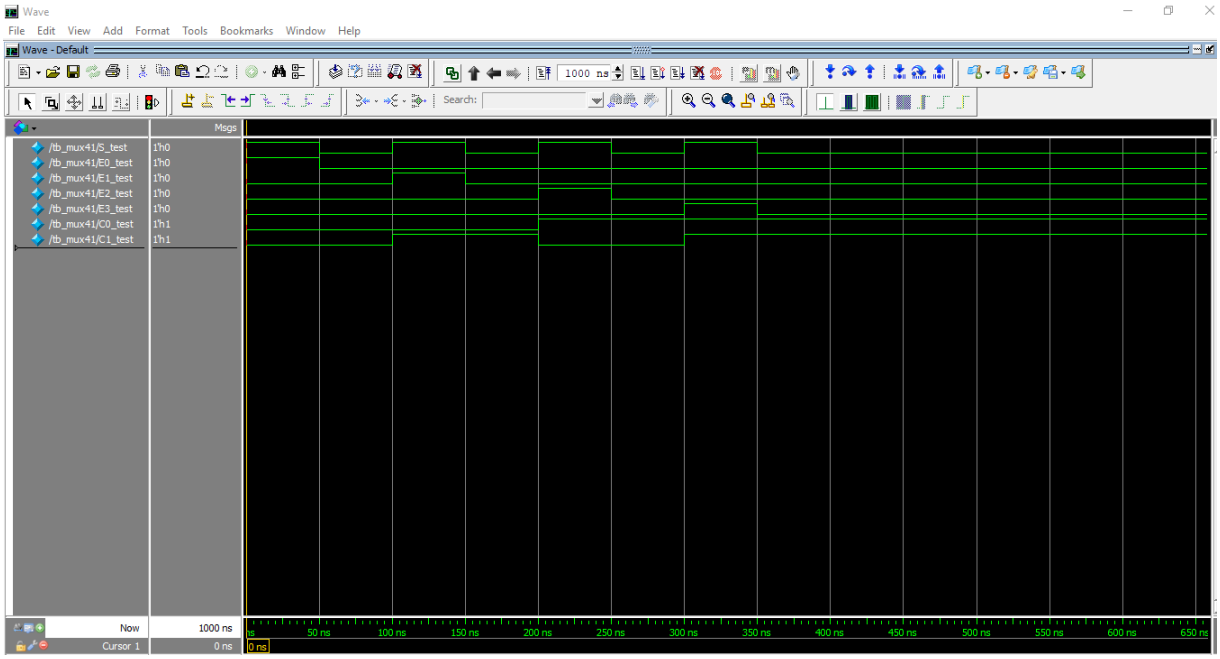

Figura 51: *Testbench* do MUX 4x1, descrito em Verilog.

4.2.2 Implementação do MUX 4x1 em VHDL

Para a descrição do MUX 4x1 em VHDL, foi criado o arquivo *mux41.vhd*, que é listado a seguir.

```
-----------------------------------------------------------------
-- Exemplo: Multiplexador 4x1
-- Arquivo: mux41.vhd
-----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity mux41 is
port
(
  E0: in std_logic;
  E1: in std_logic;
```

```
  E2: in std_logic;
  E3: in std_logic;
  C0: in std_logic;
  C1: in std_logic;
  S: out std_logic
);
end mux41;

architecture mux of mux41 is
begin
    S <= (not(C0) and not(C1) and E0) or --C0C1 == 00, S = E0--
         (not(C0) and    (C1) and E1) or --C0C1 == 01, S = E1--
         (   (C0) and not(C1) and E2) or --C0C1 == 10, S = E2--
         (   (C0) and    (C1) and E3);   --C0C1 == 11, S = E3--
end mux;

----------------------
-- Final do arquivo --
----------------------
```

Para efetuar o *testbench*, foi necessária a criação do arquivo *tb_ mux41.vhd*, apresentado a seguir.

```
-----------------------------------------------------------------
-- Exemplo de testbench para o arquivo mux41.vhdl
-- Arquivo tb_mux41.vhd
-----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity tb_mux41 is
end tb_mux41;

architecture test of tb_mux41 is
    signal S_test: std_logic;
    signal E0_test: std_logic;
    signal E1_test: std_logic;
    signal E2_test: std_logic;
    signal E3_test: std_logic;
    signal C0_test: std_logic;
    signal C1_test: std_logic;

    component mux41
        port
        (
            S: out std_logic;
            E0: in std_logic;
            E1: in std_logic;
            E2: in std_logic;
```

```
        E3: in std_logic;
        C0: in std_logic;
        C1: in std_logic
    );
end component;

begin
    --Unit Under Test--
    uut_mux41: mux41 port map
    (
        S  => S_test,
        E0 => E0_test,
        E1 => E1_test,
        E2 => E2_test,
        E3 => E3_test,
        C0 => C0_test,
        C1 => C1_test
    );

    process begin
        -- E0 --
        E0_test <= '0';
        C0_test <= '0';
        C1_test <= '0';
        wait for 50 ns;
        assert(S_test = '0') report "Erro na entrada E0_test"
        severity error;

        E0_test <= '1';
        C0_test <= '0';
        C1_test <= '0';
        wait for 50 ns;
        assert(S_test = '1') report "Erro na entrada E0_test"
        severity error;

        -- E1 --
        E1_test <= '0';
        C0_test <= '0';
        C1_test <= '1';
        wait for 50 ns;
        assert(S_test = '0') report "Erro na entrada E1_test"
        severity error;

        E1_test <= '1';
        C0_test <= '0';
        C1_test <= '1';
        wait for 50 ns;
        assert(S_test = '1') report "Erro na entrada E1_test"
        severity error;
```

```
            -- E2 --
            E2_test <= '0';
            C0_test <= '1';
            C1_test <= '0';
            wait for 50 ns;
            assert(S_test = '0') report "Erro na entrada E2_test"
            severity error;

            E2_test <= '1';
            C0_test <= '1';
            C1_test <= '0';
            wait for 50 ns;
            assert(S_test = '1') report "Erro na entrada E2_test"
            severity error;

            -- E3 --
            E3_test <= '0';
            C0_test <= '1';
            C1_test <= '1';
            wait for 50 ns;
            assert(S_test = '0') report "Erro na entrada E3_test"
            severity error;

            E3_test <= '1';
            C0_test <= '1';
            C1_test <= '1';
            wait for 50 ns;
            assert(S_test = '1') report "Erro na entrada E3_test"
            severity error;
        end process;
end test;

----------------------
-- Final do arquivo --
----------------------
```

O resultado da simulação do MUX 4x1 é mostrado na Figura 52. Nela, algumas entradas, dependendo do período em que se encontram, estão com a linha marcada de vermelho. Isso ocorre pelo fato dessas entradas serem tratadas como "*don't care*", dentro do bloco *process*, nos seus devidos períodos e etapas de teste.

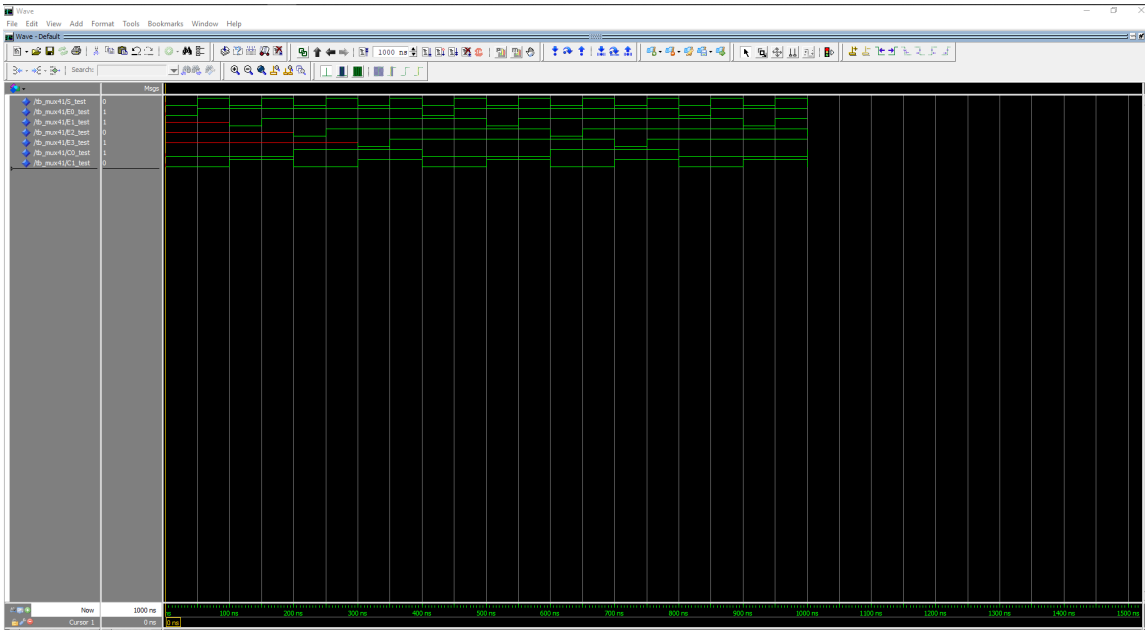

Figura 52: *Testbench* do multiplexador 4x1 em VHDL.

4.3 Meio somador ou *Half Adder* ou HA

Um meio somador (ou *Half Adder* ou HA) é um circuito combinacional capaz de expressar a soma de 2 *bits* de entrada por meio de 2 *bits* de saída. As Figuras 53 e 54, apresentam, respectivamente, o esquemático de uma possível implementação e a tabela de operação desse bloco funcional.

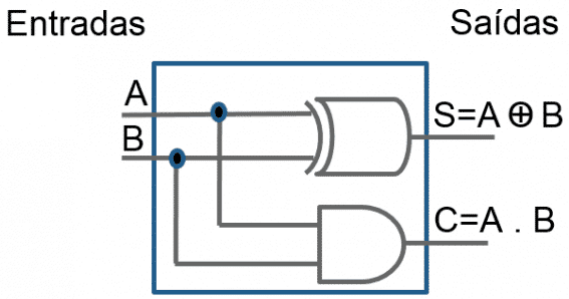

Figura 53: Esquemático de uma possível implementação para o *Half Adder*.

Tabela Verdade Meio Somador			
Entradas		Saídas	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Figura 54: Tabela verdade do *Half Adder*.

4.3.1 Implementação do HA em Verilog

A descrição do HA em Verilog é listada a seguir.

```
//////////////////////////////////////////////////////////////////
// File Downloaded from http://www.nandland.com
//////////////////////////////////////////////////////////////////
module half_adder
  (
   i_bit1,
   i_bit2,
   o_sum,
   o_carry
   );

  input  i_bit1;
  input  i_bit2;
  output o_sum;
  output o_carry;

  assign o_sum   = i_bit1 ^ i_bit2;  // bitwise xor
  assign o_carry = i_bit1 & i_bit2;  // bitwise and

endmodule // half_adder

//////////////////////
// Final do arquivo //
//////////////////////
```

O conteúdo do arquivo de *testbench* do HA é apresentado a seguir.

```
//////////////////////////////////////////////////////////////////
// File Downloaded from http://www.nandland.com
//////////////////////////////////////////////////////////////////
`include "VERILOG_half_adder.v"

module half_adder_tb;

  reg r_BIT1 = 0;
  reg r_BIT2 = 0;
  wire w_SUM;
  wire w_CARRY;

  half_adder half_adder_inst
    (
     .i_bit1(r_BIT1),
     .i_bit2(r_BIT2),
     .o_sum(w_SUM),
     .o_carry(w_CARRY)
     );
```

```
  initial
    begin
      r_BIT1 = 1'b0;
      r_BIT2 = 1'b0;
      #10;
      r_BIT1 = 1'b0;
      r_BIT2 = 1'b1;
      #10;
      r_BIT1 = 1'b1;
      r_BIT2 = 1'b0;
      #10;
      r_BIT1 = 1'b1;
      r_BIT2 = 1'b1;
      #10;
    end

endmodule // half_adder_tb

//////////////////////
// Final do arquivo //
//////////////////////
```

4.3.2 Implementação do HA em VHDL

A descrição do HA em VHDL é listada a seguir.

```
------------------------------------------------------------
-- File Downloaded from http://www.nandland.com
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity half_adder is
  port (
    i_bit1  : in std_logic;
    i_bit2  : in std_logic;
    --
    o_sum   : out std_logic;
    o_carry : out std_logic
    );
end half_adder;

architecture rtl of half_adder is
begin
  o_sum   <= i_bit1 xor i_bit2;
  o_carry <= i_bit1 and i_bit2;
end rtl;
```

```
----------------------
-- Final do arquivo --
----------------------
```

O conteúdo do arquivo de *testbench* do HA é apresentado a seguir.

```
------------------------------------------------------------
-- File Downloaded from http://www.nandland.com
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity half_adder_tb is
end half_adder_tb;

architecture behave of half_adder_tb is
  signal r_BIT1  : std_logic := '0';
  signal r_BIT2  : std_logic := '0';
  signal w_SUM   : std_logic;
  signal w_CARRY : std_logic;
begin

  UUT : entity work.half_adder  -- uses default binding
    port map (
      i_bit1  => r_BIT1,
      i_bit2  => r_BIT2,
      o_sum   => w_SUM,
      o_carry => w_CARRY
      );

  process is
  begin
    r_BIT1 <= '0';
    r_BIT2 <= '0';
    wait for 10 ns;
    r_BIT1 <= '0';
    r_BIT2 <= '1';
    wait for 10 ns;
    r_BIT1 <= '1';
    r_BIT2 <= '0';
    wait for 10 ns;
    r_BIT1 <= '1';
    r_BIT2 <= '1';
    wait for 10 ns;
  end process;
end behave;

----------------------
```

```
-- Final do arquivo --
----------------------
```

4.3.3 Simulação

As descrições em Verilog e em VHDL foram simuladas, produzindo o mesmo resultado. A Figura 55 mostra o resultado da simulação, confirmando o que foi especificado na tabela verdade da Figura 54.

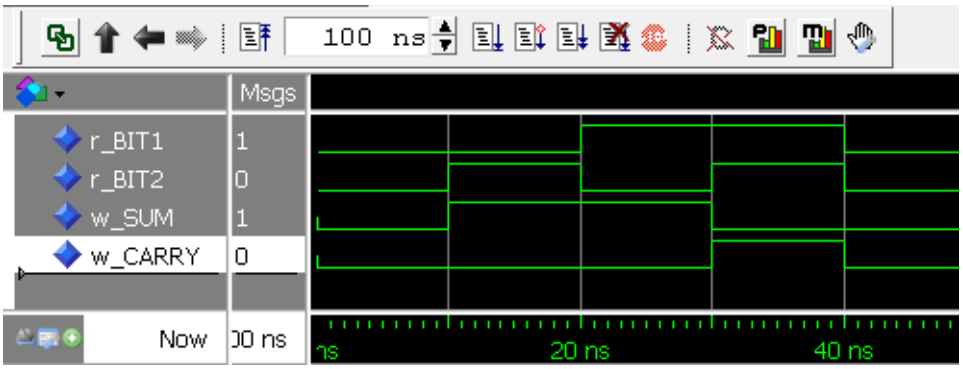

Figura 55: Formas de onda da simulação do HA.

4.4 Somador completo ou *Full Adder* ou FA

Um somador completo (*Full Adder* ou FA) é um circuito combinacional capaz de expressar a soma de 3 *bits* de entrada por meio de 2 *bits* de saída.

A Figura 56 apresenta o esquemático de uma possível implementação

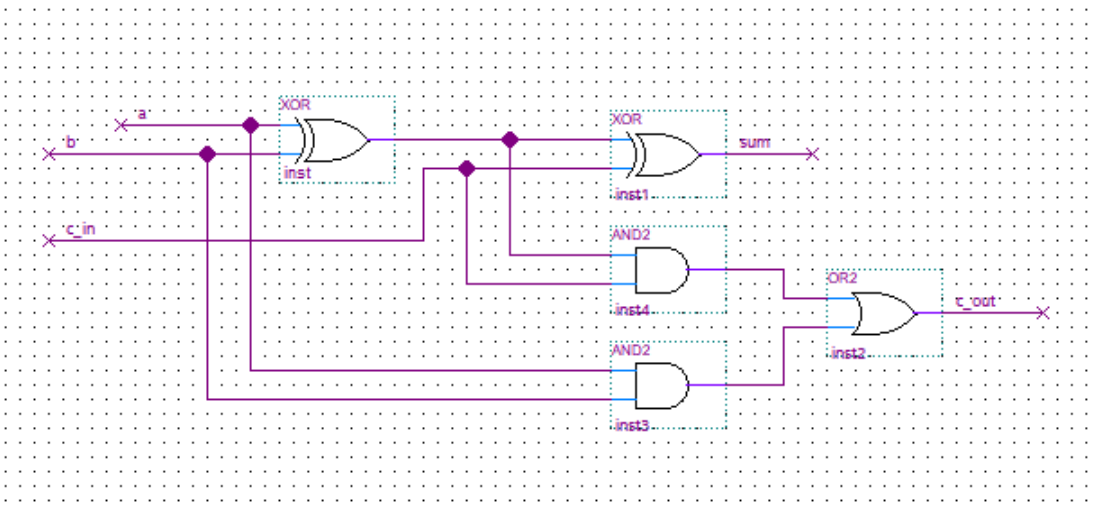

Figura 56: Esquemático de uma possível implementação para o *Full Adder*. Diagrama elaborado no Quartus Prime 18.1.

A descrição do FA em VHDL é listada a seguir.

```
-----------------------------------------------------------------
-- Exemplo basico de somador completo (full-adder)
-- Arquivo full_adder.vhdl
```

```
------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY full_adder IS
    -- Declaracao das entradas
    PORT( a, b, c_in : IN std_logic;
          c_out, sum : OUT std_logic );

END full_adder;

ARCHITECTURE ckt OF full_adder IS
-- Sinais intermediarios
SIGNAL a_xor_b, and_1, and_2: std_logic;

-- Estrutura do somador
    BEGIN
        a_xor_b <= a XOR b;
        and_1   <= a_xor_b AND c_in;
        and_2   <= a AND b;

        sum     <= a_xor_b XOR c_in;
        c_out   <= and_1 OR and_2;

END ARCHITECTURE ckt;

----------------------
-- Final do arquivo --
----------------------
```

O conteúdo do arquivo de *testbench* do FA é apresentado a seguir.

```
------------------------------------------------------------------
-- Exemplo simples de testbench para o arquivo full_adder.vhdl
-- Arquivo full_adder_testbench.vhdl
------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY full_adder_testbench IS
END full_adder_testbench;

ARCHITECTURE tb_ckt OF full_adder_testbench IS
    SIGNAL test_a, test_b, test_c_in: std_logic; -- corresponde as entradas
    SIGNAL test_c_out, test_sum: std_logic; -- corresponde as saidas

BEGIN
    -- instanciando o circuito em teste
    uut: ENTITY work.full_adder(ckt) -- instancia do modulo
    PORT MAP(a=>test_a, b=>test_b, c_in=>test_c_in, c_out=>test_c_out,
    sum=>test_sum);
```

```
    PROCESS BEGIN
        -- teste 1
        test_a <= '0';
        test_b <= '0';
        test_c_in <= '1';
        WAIT FOR 200 ns;
        -- teste 2
        test_a <= '0';
        test_b <= '1';
        test_c_in <= '1';
        WAIT FOR 200 ns;
        -- teste 3
        test_a <= '1';
        test_b <= '1';
        test_c_in <= '0';
        WAIT FOR 200 ns;
        -- teste 4
        test_a <= '1';
        test_b <= '0';
        test_c_in <= '1';
        WAIT FOR 200 ns;
        -- teste 5
        test_a <= '1';
        test_b <= '0';
        test_c_in <= '0';
        WAIT FOR 200 ns;
        -- teste 6
        test_a <= '0';
        test_b <= '0';
        test_c_in <= '0';
        WAIT FOR 200 ns;
        -- teste 7
        test_a <= '1';
        test_b <= '1';
        test_c_in <= '1';
        WAIT FOR 200 ns;
        -- teste 8
        test_a <= '0';
        test_b <= '1';
        test_c_in <= '0';
        WAIT FOR 200 ns;
    END PROCESS;
END tb_ckt;

----------------------
-- Final do arquivo --
----------------------
```

A Figura 57 mostra o resultado da simulação do FA.

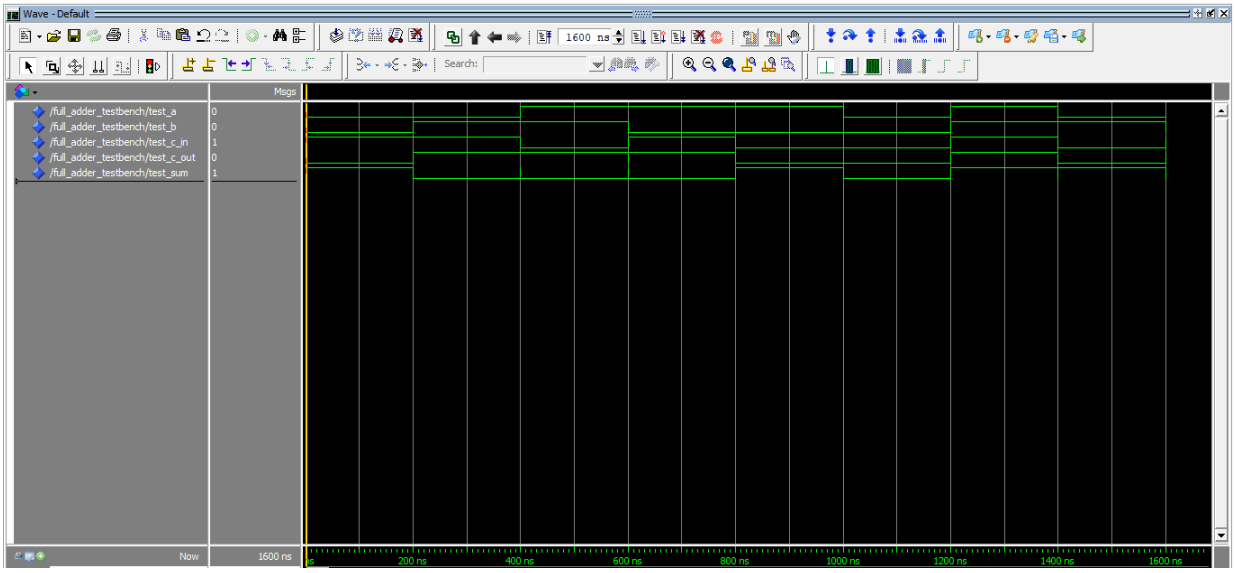

Figura 57: Formas de onda da simulação do FA.

Referências

[Alta] Altera. Max+plus II. "https://www.intel.com/content/www/us/en/programmable/support/support-resources/download/legacy/maxplux2/mp2-index.html". Acesso em: 20/04/2020.

[Altb] Altera. Quartus ii. https://www.intel.com/content/www/us/en/programmable/downloas/download-center.html. Acesso em: 20/04/2020.

[BMS+] Gabriel Bueno, João Pedro Matheus, Lucca Sabbatini, Vinícius Corrêa Figueira, and Alexandre Santos de la Vega. Configurando o *kit* FPGA DE10-Lite. http://www.telecom.uff.br/pet/petws/downloads/tutoriais/fpga/Tutorial_manual_DE10_fpga_2019_12_20.pdf. Acesso em: 20/04/2020.

[Cho] Sumouli Choudhury. *Full Adder in Digital Logic*. "https://www.geeksforgeeks.org/full-adder-in-digital-logic/". Acesso em: 20/04/2020.

[Chu11a] Pong P. Chu. *FPGA prototyping by Verilog examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[Chu11b] Pong P. Chu. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[Emb] Embarcados. Tutorial de Verilog: meio somador (*half adder*). "https://www.embarcados.com.br/tutorial-de-verilog-meio-somador-half-adder/". Acesso em: 20/04/2020.

[Gra15] Mentor Graphics. *ModelSim Tutorial*, 10.4 edition, 2015.

[Int] Intel. Altera. "https://www.intel.com/content/www/us/en/productus/programmable.html". Acesso em: 20/04/2020.

[Lim15] Thiago Lima. Tutorial de verilog: Somador completo (*full adder*). "https://www.embarcados.com.br/tutorial-de-verilog-somador-completo/", 2015. Acesso em: 20/04/2020.

[Men] Mentor Graphics. Passos para instalação do ModelSim. "https://www.mentor.com/company/higher_ed/modelsim-student-edition". Acesso em: 20/04/2020.

[Nan] Nand Land. *Half Adder Module in VHDL and Verilog*. "https://www.nandland.com/vhdl/modules/module-half-adder.html". Acesso em: 20/04/2020.

[PETa] PET. http://portal.mec.gov.br/index.php?option=com_content&view=article&id=12223&ativo=481&Itemid=480. Acesso em: 20/04/2020.

[PETb] Grupo PET-Tele. http://www.telecom.uff.br/pet. Acesso em: 20/04/2020.

[Rea11] Blaine C. Readler. *Verilog by example: a concise introduction for FPGA design*. Full Arc Press, 2011.

[Rou16] Route2basics. Verilog code of 4x1 Multiplexer. "https://www.youtube.com/watch?v=L9dPt08nMMs", 2016. Acesso em: 20/04/2020.

[Rus] Russel. *Full Adder Module in VHDL and Verilog*. "https://www.nandland.com/vhdl/modules/module-full-adder.html". Acesso em: 20/04/2020.

[stu] FPGA 4 student. *Verilog code for full adder*. "https://www.fpga4student.com/2017/02/verilog-code-for-full-adder.html". Acesso em: 20/04/2020.

[Ter18] Terasic. *DE10-Lite User Manual*, 2018.

[VHD14a] Power VHDL. Multiplexador 4x1 vhdl. "https://www.youtube.com/watch?v=I6-3w9HAY70", 2014. Acesso em: 20/04/2020.

[VHD14b] Power VHDL. Primeiro *testbench*. "https://www.youtube.com/watch?v=iCh9Yn_ZmE4", 2014. Acesso em: 20/04/2020.

[Wik] Wikipedia. Hardware description language. "https://en.wikipedia.org/wiki/Hardware_description_language". Acesso em: 20/04/2020.