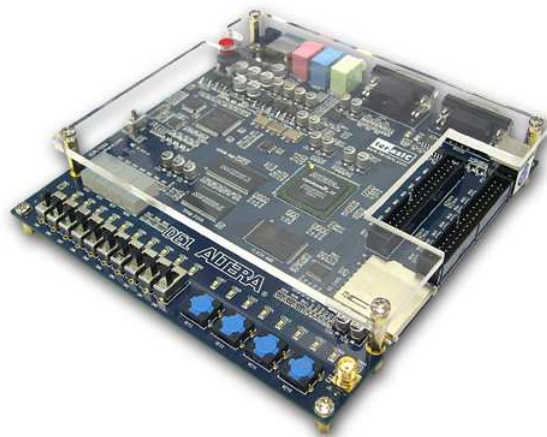




UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP  
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO - FEEC  
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL - DCA

- TUTORIAL -  
ALTERA QUARTUS® II E  
*Kit* DE DESENVOLVIMENTO DE1



Versão 2.1

Campinas, 30 de julho de 2010

## Notas da Versão

**Versão 2.1:** Uma atualização da versão anterior, do primeiro semestre de 2010. Foram incluídas a seção sobre FPGA e sobre a visualização gráfica de NETLIST. Foi também revisada a seção de analisador de tempos.

**Versão 2.0:** Reescrita do texto originalmente proposto no segundo semestre de 2009. Ampliação do conteúdo e adequação das figuras referentes ao ambiente de desenvolvimento QUARTUS® II para a versão 9.1<sup>1</sup>

## Licença de Uso

Este tutorial tem objetivo puramente educativo, sem nenhuma finalidade lucrativa. Sua cópia para finalidades educacionais e pessoais, total ou parcial, é totalmente permitida pelos autores.

---

<sup>1</sup>Na data de publicação deste manual, é disponibilizado em (Altera, 2010f) uma versão já com o pacote de atualizações SP2.

## Prefácio da Versão 2.1

As dúvidas levantadas pelos alunos da disciplina de EA773 – Laboratório de Circuitos Lógicos, ao longo do primeiro semestre de 2010, nos motivaram a inserir uma seção sobre a visualização gráfica do resultado sintetizado pelo aplicativo QUARTUS® II, fazer uma brevíssima introdução aos dispositivos FPGAs e revisar a seção sobre os tempos gerados pelo seu analisador temporal clássico.

Agradecemos a todos os nossos alunos que contribuíram, direta e indiretamente, à melhoria do conteúdo deste breve tutorial e esperamos que ele facilite os novos usuários de QUARTUS® II a explorar toda a sua potencialidade. Continuamos abertos a todas as críticas construtivas para aprimorar este material.

Bom proveito!

Prof. Dra. Wu Shin - Ting  
ting AT dca DOT fee DOT unicamp DOT br

30 de julho de 2010

## Prefácio da Versão 2.0

O início deste texto deu-se a partir da Versão Beta 1.0 a qual foi desenvolvida no segundo semestre de 2009 pelo monitor da disciplina Lucas Martins Guido. Assim sendo, esse texto foi inicialmente pensado para ser a Versão 1.1. Contudo, devido ao grande número de correções, sugestões e adequações para a versão 9.1 do QUARTUS<sup>®</sup> II, achamos melhor classificar essa nova versão do tutorial como 2.0.

A princípio, partimos da organização do texto apresentado anteriormente na Versão Beta 1.0. Entretanto, pensamos em suplantando algumas partes do texto, explorar melhor alguns exemplos apresentados anteriormente e, além disso, apresentar as referências principais para aqueles que utilizarão esse ambiente de desenvolvimento pela primeira vez. Como consequência, a estruturação inicial do texto, proposta anteriormente, teve que ser alterada para adequar-se às modificações realizadas.

Esse breve tutorial foi pensado para os alunos da disciplina EA773 – Laboratório de Circuitos Lógicos os quais utilizarão, ao longo do curso, a placa de desenvolvimento da Terasic Technologies, a *Development and Education Board* – DE1. Gostaríamos de ressaltar que existe um excelente manual na página do fabricante (Altera e Terasic, 2010) o qual sugerimos fortemente que seja utilizado como fonte principal de informações sobre a placa DE1, onde todos os módulos funcionais da placa são apresentados em grande detalhe.

A seção das principais perguntas apresentadas pelos alunos em semestres anteriores foi mantida a menos de modificações no texto para melhor compreensão. Provavelmente no final do semestre, novas informações venham a ser inseridas.

Aos mais experientes, fica a sugestão para aprofundar-se a partir do estudo da página do fabricante do QUARTUS<sup>®</sup> II, onde é possível encontrar muita informação, muito mais detalhadas do que é apresentado aqui.

Agradecemos a todos os nossos alunos que contribuíram com suas perguntas, observações, e soluções criativas ao longo dos cursos que lecionamos. Estamos sempre abertos para críticas construtivas a fim de melhorar o material apresentado.

E, por fim, esperamos que este breve tutorial te ajude a entrar ao fantástico mundo de Projetos Digitais.

Prof. Dra. Wu Shin - Ting  
ting AT dca DOT fee DOT unicamp DOT br

Filipe Ieda Fazanaro  
filipe DOT fazanaro AT gmail DOT com

11 de maio de 2010

## Prefácio da Versão 1.0

Esse tutorial foi criado para finalidades educacionais, inicialmente com o objetivo de auxiliar os alunos de EA773-Laboratório de Circuitos Lógicos, na UNICAMP. O software Quartus II que utilizei para a obtenção das imagens contidas neste tutorial é a versão Web Edition que não requer licença.

Escrevi na forma de um passo-a-passo, que começa desde o download do software e abrange a prototipagem do sistema digital, simulação em software e dicas rápidas. Não me preocupei em usar uma linguagem rebuscada e procurei durante o texto todo colocar a informação de uma forma agradável.

No final desse tutorial você encontrar um FAQ com as perguntas mais frequentes durante as monitorias. Possivelmente suas dúvidas futuras podem aparecer lá.

Meu objetivo é transmitir aos colegas um pouco da experiência que adquiri em alguns semestres lidando com essa placa e software. Dicas para melhorar esse material são muito bem-vindas. Agradeço a todos que colaboraram para a criação deste tutorial, contribuindo com dúvidas, incentivo e apoio. Espero que ajude.

Abraços,

Lucas Martins Guido

06 de setembro de 2009

# Sumário

<b>1</b>	<b>Introdução</b>	<b>7</b>
<b>2</b>	<b>Começando com Alguns Conceitos</b>	<b>7</b>
2.1	Projeto . . . . .	7
2.2	Metodologia de Projeto Hierárquico . . . . .	8
2.3	Nível de Abstração de um Sistema Digital . . . . .	9
2.4	FPGAs . . . . .	10
2.5	VHDL . . . . .	11
<b>3</b>	<b>Instalando o Quartus® II</b>	<b>12</b>
<b>4</b>	<b>Modelando Um Projeto</b>	<b>15</b>
4.1	Modelo Comportamental . . . . .	15
4.2	Modelo Estrutural . . . . .	16
<b>5</b>	<b>Criando um Projeto no Quartus® II</b>	<b>17</b>
<b>6</b>	<b>Criando Arquivos de Projeto</b>	<b>19</b>
<b>7</b>	<b>Capturando o Esquemático</b>	<b>20</b>
7.1	Inserção de Componentes . . . . .	21
7.2	Ligação entre Componentes . . . . .	21
7.3	Nomeação dos Pinos de Entrada e de Saída . . . . .	23
7.4	Nomeação das Linhas das Ligações . . . . .	24
<b>8</b>	<b>Descrevendo em VHDL</b>	<b>25</b>
<b>9</b>	<b>Encapsulando os Componentes</b>	<b>27</b>
<b>10</b>	<b>Compilando</b>	<b>29</b>
<b>11</b>	<b>Visualizando os Resultados da Compilação</b>	<b>31</b>
<b>12</b>	<b>Analisando as Restrições de Tempos do Projeto</b>	<b>33</b>

<b>13 Simulando</b>	<b>34</b>
<b>14 Carregando o Projeto</b>	<b>39</b>
<b>15 FAQs</b>	<b>41</b>
15.1 Por que a onda de saída está deslocada para a direita em relação às entradas . . .	41
15.2 O que são esses picos que ocorrem, às vezes, nas transições? . . . . .	42
15.3 Como aumento o tempo máximo nas simulações? . . . . .	42
15.4 Não compila e aparece que o problema é a hierarquia. . . . .	42
15.5 O nome do meu projeto é Projetão da Moçada e não compila. Por quê? . . . . .	42
15.6 Tenho 200 entradas que ficam em 1 sempre, mais 200 que sempre ficam em 0. Quando coloco no <i>Waveform</i> fica uma bagunça. Já que eu não tenho que mudá-las sempre, tem algo mais prático a ser feito? . . . . .	43
15.7 Fui no <i>TTL HandBook</i> e justamente a página que eu queria não estava lá. O que faço? . . . . .	43
15.8 Sumiram as ferramentas do Waveform. . . . .	43
15.9 No simulador, os pinos não aparecem. . . . .	43
15.10 Simulei anteriormente, inseri e tirei alguns pinos, quero simular novamente. Compilo e quando abro o simulador só tenho pinos antigos. Que fazer? . . . . .	43
15.11 Sumiram as ferramentas da área de trabalho. . . . .	43
15.12 Estou na tela de associação de pinos com endereços, para gravação. O problema é que minha tela não mostra, na parte de baixo, os pinos do meu projeto. . . . .	44
15.13 Estou na tela de associação de pinos com endereços, para gravação. O problema é que minha tela não mostra, ao lado dos pinos do meu projeto, a coluna <i>Location</i> . . . . .	44
15.14 Estou na tela de gravação mas não consigo gravar. . . . .	44
15.15 Usei o <i>clock</i> da placa e todos os meus LEDs ficam acesos. . . . .	44
15.16 Por que ao renomear um pino de entrada com o nome <i>CLOCK_27</i> , o sistema não o associa ao pino do relógio de 27 MHz do <i>kit</i> ? . . . . .	44

# 1 Introdução

Esse tutorial foi elaborado com o objetivo de auxiliar os alunos do curso EA773 – Laboratório de Circuitos Lógicos da Faculdade de Engenharia Elétrica e de Computação (FEEC), UNICAMP, os quais, pelo menos em sua maioria, terão contato pela primeira vez com o ambiente QUARTUS® II, da Altera (Altera, 2010a) bem como com a placa de desenvolvimento DE1 (Altera e Terasic, 2010) a qual é disponibilizada aos alunos durante o curso. Para aqueles que já tiveram a oportunidade de utilizar tanto o *software* quanto a placa (ou que já trabalham com outros dispositivos lógicos programáveis da Altera) e queiram se aprofundar sobre as suas possibilidades, sugere-se que as referências (Altera, 2010a; Altera, 2010f) sejam utilizadas como fonte de informação.

Durante o 2<sup>o</sup> semestre de 2010, o curso tem como base os roteiros experimentais propostos no seguinte endereço eletrônico:

<http://www.dca.fee.unicamp.br/courses/EA773/2s2010/index.html>

Em primeiro lugar, apresentamos alguns conceitos úteis relacionados com o ambiente de desenvolvimento QUARTUS® II. Em seguida, serão apresentados os principais passos que um aluno pode seguir para criar um projeto nesse ambiente de desenvolvimento, desde a sua instalação, passando pelos procedimentos de configuração, criação de um ambiente de projeto, criação dos arquivos de projeto, utilizando tanto esquemático quanto linguagens de descrição de *hardware*, tais como VHDL, compilação, simulação, e análise do desempenho temporal. E, finalmente, são dados os passos necessários para programar um *chip* FPGA disponível no *kit* DE1. Todos esses passos são exemplificados a partir de um projeto de máquina de 4 estados.

## 2 Começando com Alguns Conceitos

Nesta seção é apresentada uma breve introdução aos termos relacionados com o desenvolvimento de um projeto digital.

### 2.1 Projeto

Tanto no QUARTUS® II como em diversos aplicativos de desenvolvimento de circuitos digitais configuráveis, todos os dados referentes ao circuito devem estar contidos em um projeto. Um projeto é uma espécie de “ambiente” em que é possível desenhar um circuito lógico digital, executar compilações, simulações, analisar sinais em tempo de execução (Altera, 2010c) etc. Um projeto inclui o nome do projeto, a pasta, o nome da entidade no topo do nível hierárquico corrente e o dispositivo onde o projeto será carregado. Arquivos que não estejam relacionados ao projeto não serão levados em conta na compilação e muito menos durante a simulação.



Muitos erros ocorrem devido ao descaso com os arquivos do projeto, que devem estar juntos em uma mesma pasta. Com isso em mente e para evitar problemas, vale o seguinte lema:

*UM projeto, UMA pasta.*

O primeiro passo para começar um projeto no QUARTUS<sup>®</sup> II consiste em criar uma pasta para colocar os respectivos arquivos. Faça isso como de costume, escolha um local (que pode ser um *pendrive*, mesmo não sendo aconselhável) no computador e crie uma pasta vazia. Um detalhe bastante importante consiste no fato de que o nome da pasta **não** pode conter espaços. Por essa razão, “Projeto 01” não é um nome válido e, ao invés disso, deve-se optar, por exemplo, por “Projeto.01” ou por “Projeto\_01”.

## 2.2 Metodologia de Projeto Hierárquico

Metodologia de projeto hierárquico consiste essencialmente em dividir projetos em uma estrutura hierárquica de entidades de projeto. Cada uma das entidades possui uma função específica e uma interface de iteração com outras entidades. Assim, um projeto pode ser visto através de diferentes níveis de hierarquia, desde os blocos lógicos elementares até a visão geral no topo da hierarquia. Quando o fluxo de desenvolvimento de um projeto parte dos blocos elementares e, então, integrá-los em entidades de nível mais alto, é dito que se trata da metodologia *bottom-up*. E quando o fluxo de projeto ocorre no sentido inverso, saindo de um nível de concepção geral e vai se detalhando até chegar a um nível realizável com a tecnologia disponível, a metodologia é conhecida como *top-down*.

No ambiente QUARTUS<sup>®</sup> II é possível adotar a metodologia *bottom-up*, *top-down*, ou a combinação das duas para desenvolver um projeto. Neste curso será usada a metodologia *bottom-up* que impõe a seguinte estrutura: diversas entidades auxiliares que devem ser agregadas para constituir uma entidade de projeto principal. Essas entidades são colocadas em uma espécie de árvore, o que forma uma hierarquia. Não respeitar essa hierarquia de entidades resulta em erros durante a compilação do projeto, pois a compilação é executada de forma ascendente, ou seja, a entidade no topo será o último lido no processo de compilação. Pode-se ilustrar estes conceitos com um exemplo.

Imagine que se tenha como objetivo criar um relógio digital com despertador, isto é, quando o horário for igual ao horário selecionado o relógio despertará. Supõe-se ainda que exista uma lista de componentes e ferramentas que poderão ser utilizadas, entre as quais figuram um relógio digital e um comparador. Pode-se então construir a partir do relógio e do comparador um despertador, conforme mostra a Figura 1. A entidade “Relógio COM Despertador” é o topo da hierarquia de projeto.

Diversas vantagens podem ser destacadas quando se emprega a metodologia hierárquica no desenvolvimento de um projeto. Uma delas é que o desenvolvimento do projeto pode ser

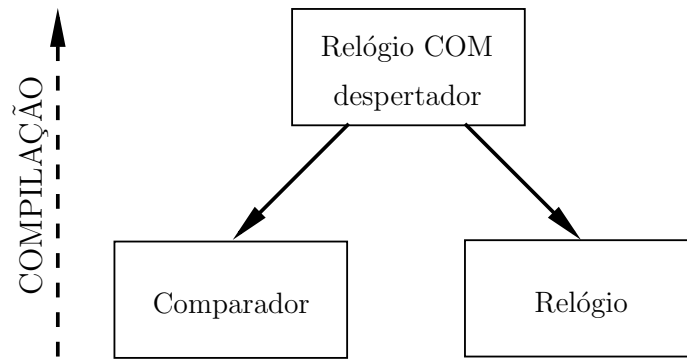


Figura 1: Hierarquia de entidades de projeto.

realizado por partes, ou seja, cada entidade de projeto é compilado, analisado, simulado e testado separadamente. Assim, garante-se que o seu funcionamento esteja de acordo com as especificações do projeto e, eventualmente, erros, modificações e otimizações tornam-se tarefas mais simples de serem executadas. Adicionalmente, ao se analisar o projeto como um todo, fica mais simples procurar por eventuais erros que venham a ocorrer e, conseqüentemente, facilita o trabalho para a sua correção já que os módulos foram testados previamente. Na Seção 9 são discutidos os principais procedimentos para que uma ou mais entidades desenvolvidas em um projeto A sejam empregados em um outro projeto B e/ou C.

### 2.3 Nível de Abstração de um Sistema Digital

Um projeto em circuitos digitais pode ser analisado a partir de diferentes níveis de abstração, como é ilustrado na Figura 2 (adaptado a partir de (der Spiegel, 2010)) facilitando, assim, a descrição e o desenvolvimento do projeto, independentemente de sua complexidade.

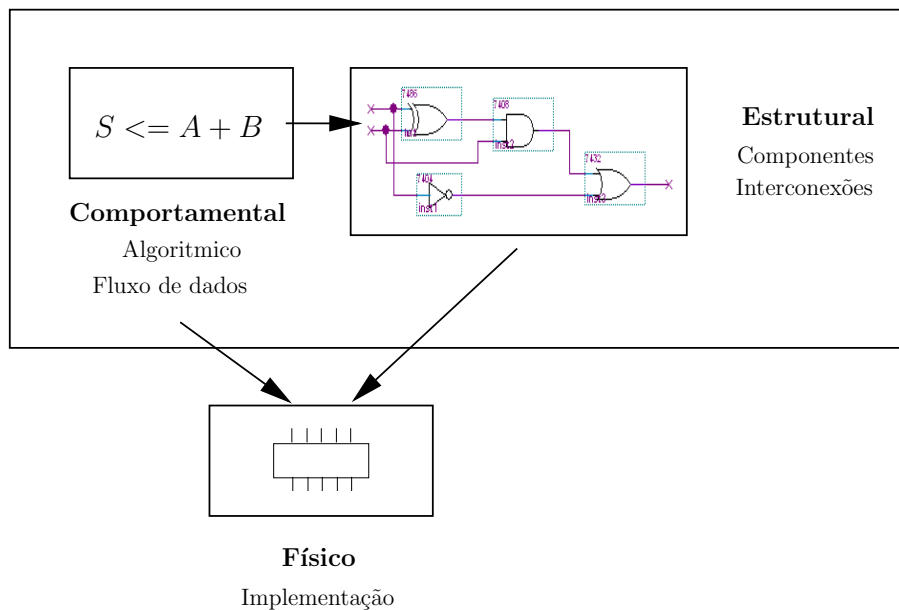


Figura 2: Níveis de abstração de um projeto digital.

O mais alto nível de abstração é o **comportamental** onde se descreve o circuito em

termos do que ele faz, o que ele implementa e como ele se comporta. Em outras palavras, a descrição comportamental mostra a relação entre os sinais de entrada e os de saída tal como uma tabela-verdade, ou a relação entre os estados por meio de uma tabela de transições ou um diagrama de estados. Já no nível **estrutural** tem-se a descrição do sistema em termos dos componentes lógicos (tais como portas lógicas, registradores e *flip-flops*) interconectados, ou seja, as estruturas lógicas que realizam as funções especificadas nos requisitos do projeto. Finalmente, no nível **físico** ocupa-se com a tecnologia onde o projeto será implementado. Esta tecnologia pode variar desde os componentes lógicos de função fixa, como os circuitos integrados da família lógica TTL ou CMOS, até os dispositivos lógicos “programáveis” (PLDs – *Programmable Logic Devices*) de alta densidade.

## 2.4 FPGAs

Diferentemente dos circuitos integrados da família lógica TTL ou CMOS, que tem as suas funções lógicas definidas no ato de fabricação, as matrizes de portas programáveis em campo, mais conhecidos como FPGAs – *Field Programmable Gate Arrays*, caracterizam-se por apresentarem as suas funcionalidades configuradas exclusivamente pelos usuários. Esta classe de dispositivos lógicos foi inventada por Ross Freeman, um dos fundadores de Xilinx Inc. em 1984, e teve o seu lançamento no ano de 1985.

Distinguem-se num FPGA três módulos: blocos de entrada e saída, blocos lógicos configuráveis, e uma matriz de chaves de interconexão. Os blocos lógicos são dispostos de forma bidimensional, as chaves de interconexão são dispostas em formas de trilhas verticais e horizontais entre as linhas e as colunas dos blocos lógicos, e os blocos de entrada e de saída são dispostos na borda do *chip*. Tipicamente, os blocos lógicos compreendem alguns *flip-flops* e lógicas combinacionais, e as interconexões entre eles podem ser configuradas por meio de um arquivo binário, conhecido genericamente como BITSTREAM. Vale observar que para a maioria dos FPGAs este arquivo precisa ser recarregado após cada corte de energia.

Para facilitar a configuração de um dispositivo lógico programável, existem aplicativos vendidos por terceiros capazes de converter as descrições de um projeto em alto nível numa malha de blocos lógicos conectados, também conhecida como NETLIST, e transformar esta malha num “*layout*” incluindo blocos lógicos, blocos de entrada e de saída e caminhos de interconexões existentes nesses dispositivos lógicos. Tais aplicativos são conhecidos como **compiladores lógicos**.

## 2.5 VHDL

A Linguagem de Descrição de *Hardware* VHDL<sup>2</sup> foi desenvolvida em meados da década de 1980 pelo Departamento de Defesa dos Estados Unidos como uma forma concisa de documentar os projetos desenvolvidos sobre circuitos integrados de alta velocidade (VHSIC) e, posteriormente, padronizada pelo IEEE (IEEE-VHDL, 2009). Com o desenvolvimento de componentes eletrônicos reconfiguráveis para construir circuitos digitais, tais como os FPGAs, o padrão VHDL se transformou em uma das principais linguagens de descrição de *hardware* de alto nível para projetar e implementar circuitos digitais (Tocci, Widmer e Moss, 2008).

O VHDL permite, então, que o sistema digital seja descrito tanto em nível estrutural quanto em nível comportamental, focando no percurso dos dados ao longo do **fluxo de dados** e como eles são tratados (**sequencialmente** ou **concorrentemente**) (IEEE-VHDL, 2009; Pedroni, 2008; Perry, 2002; der Spiegel, 2010; Tocci *et al.*, 2008).

Assim como qualquer linguagem de especificação, o VHDL obedece alguns padrões de estruturação. Deve-se atentar ao fato de que o VHDL, como o próprio nome diz, é uma linguagem de *descrição de hardware* e, portanto, deve-se ter em mente que o código irá descrever um circuito digital, o seu comportamento, a maneira pela qual serão tratados os sinais envolvidos. No Código 1 é apresentado um exemplo da descrição de um semi somador de dois *bits*  $x_i$  e  $y_i$  em VHDL, cujo resultado de soma e de transporte é, respectivamente

$$\begin{aligned} p_i &= x_i \oplus y_i \\ g_i &= x_i \wedge y_i \bar{q}_0 \end{aligned} \tag{1}$$

Basicamente, um circuito descrito a partir do padrão VHDL é composto de três partes. Na primeira parte, no cabeçalho, definem-se as bibliotecas que serão empregadas no circuito do semi somador descrito pelo Código 1. Na segunda parte define-se a entidade (**Entity**) que representa o circuito. Nesse ponto, são definidas todas as portas de entrada e de saída e, também, os tipos de sinais que serão empregados. Por exemplo, pode-se definir que uma determinada entrada da entidade receba *bits* de sinais. A sutileza ao definir, por exemplo, uma entrada como binária consiste no fato de que a variável somente aceitará níveis lógicos ALTO ('1') ou BAIXO ('0'). Já ao se definir como sinal lógico (como explicitado no Código 1), as variáveis podem receber, além de '1' e '0', sinais indeterminados (*don't care*).

Finalmente, na terceira parte do código, define-se a arquitetura do circuito, ou seja, o seu funcionamento propriamente dito. As funções **xor** e **and** são definidas a partir das bibliotecas incluídas na primeira parte, logo no início do código (IEEE-VHDL, 2009).

É importante salientar que não é objetivo deste tutorial realizar estudos aprofundados

---

<sup>2</sup>Abreviação do termo, em inglês, *Very High Speed Integrated Circuit* (VHSIC) *Hardware Description Language* (HDL).

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity semi_somador is
6      port
7      (
8          xi  : in std_logic;
9          yi  : in std_logic;
10         pi  : out std_logic;
11         gi  : out std_logic
12     );
13 end semi_somador;
14
15 architecture behavior of semi_somador is
16 begin
17     pi <= xi xor yi;
18     gi <= xi and yi;
19
20 end behavior;

```

Código 1: Descrição de um semi-somador em VHDL.

quanto ao padrão VHDL. Para isso, recomenda-se o estudo atento de livros especializados sobre o assunto tais como o de Pedroni (2008) e o de Perry (2002), o material apresentado em (der Spiegel, 2010) e, principalmente, pela documentação referente à padronização disponibilizada em (IEEE-VHDL, 2009).

### 3 Instalando o Quartus<sup>®</sup> II

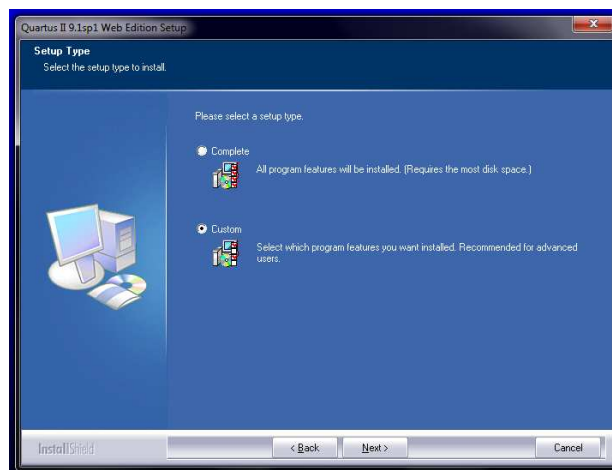
A Altera disponibiliza uma versão gratuita do QUARTUS<sup>®</sup> II, denominada de *Web Edition* em seu site, sem que haja a necessidade de aquisição de uma licença comercial<sup>3</sup>. Basta fazer o *download* e instalar no seu computador. Atualmente o endereço para o *download* é:

[https://www.altera.com/support/software/download/altera\\_design/quartus\\_web/dnl-quartus\\_we.jsp](https://www.altera.com/support/software/download/altera_design/quartus_web/dnl-quartus_we.jsp)

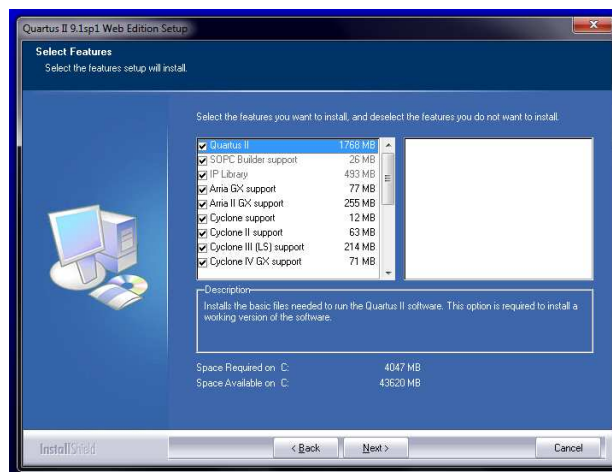
---

<sup>3</sup>Existem algumas diferenças entre a versão gratuita e a versão comercial. Basicamente, essa diferença consiste no tipo de licença a qual libera ou não determinadas funcionalidades do *software*. Para maiores informações, deve-se analisar o seguinte endereço eletrônico: [http://www.altera.com/products/software/products/quartus2web/features/sof-quarweb\\_features.html](http://www.altera.com/products/software/products/quartus2web/features/sof-quarweb_features.html).

O arquivo de instalação da versão 9.1 do QUARTUS® II possui um tamanho aproximado de 1.95 GB<sup>4</sup> e já vem com os pacotes de atualizações (quando existentes) embutidos<sup>5</sup>. Contudo, ao selecionar todos os pacotes disponíveis, após finalizado o procedimento de instalação, o QUARTUS® II chega a ocupar mais de 4.1 GB. Existem alguns detalhes que podem economizar um espaço no HD sem prejudicar as funcionalidades necessárias para o andamento do curso. Para isso, logo no início da instalação, deve-se escolher a opção *Custom*, como pode ser observado na Figura 3(a):



(a) Seleção do modo de instalação.



(b) Seleção dos pacotes a serem instalados.

Figura 3: Primeiras janelas do programa de instalação do QUARTUS® II.

Em seguida, tem-se a possibilidade de escolher quais pacotes podem ser instalados como mostra a Figura 3(b). No caso de omissão por parte do usuário, o *software* de instalação habilita os pacotes para todas as famílias de FPGAs disponibilizados pela Altera. Como o FPGA do *kit* de desenvolvimento disponível aos alunos do curso EA773 é um Dispositivo Lógico Programável (*Programmable Logic Device* - PLD) da família CYCLONE II (Altera, 2010b), não é

<sup>4</sup>Esse é o arquivo de instalação para o Windows. Existe uma versão beta para Linux mas o procedimento de instalação é diferente e não será tratado neste tutorial. Um novo tutorial está em fase de desenvolvimento o qual abrangerá os principais pontos para sistemas operacionais Linux.

<sup>5</sup>Na data deste tutorial, a versão para *download* vem incorporado o pacote de atualizações SP2.

necessário instalar todos os pacotes e, dessa maneira, alguns podem ser desabilitados sem perda de funcionalidade do QUARTUS<sup>®</sup> II. As seguintes opções devem ser levadas em consideração ao longo da instalação<sup>6</sup>:

- QUARTUS II (obrigatório): o *software* de desenvolvimento propriamente dito;
- SOPC BUILDER SUPPORT (obrigatório): essa é uma ferramenta obrigatória no desenvolvimento do *softcore* NIOS II, não explorado ao longo do curso (Altera, 2010e);
- IP LIBRARY (obrigatório): contém toda a propriedade intelectual que implementa determinadas funcionalidades, tais como controladores de memória, periféricos de interfaces (ETHERNET, PCI, PCIx), dentre outras possibilidades (Altera, 2010d);
- CYCLONE II SUPPORT (obrigatório): suporte à família de FPGA que é usada no *kit* de desenvolvimento DE1;
- THIRD-PARTY EDA TOOL INTERFACES (opcional): ferramentas auxiliares de desenvolvimento implementadas por empresas parceiras da Altera;
- TUTORIAL FILES (opcional): arquivos auxiliares de tutoriais disponibilizados no diretório de instalação (arquivos em formato .pdf) e bastante úteis. Recomenda-se que sejam instalados (e estudados).

Existem dois estilos para o ambiente de desenvolvimento: QUARTUS II e MAX+PLUS II, sendo esse último *layout* a base para esse tutorial. A opção pelo estilo do ambiente de desenvolvimento aparece logo após que a instalação do QUARTUS<sup>®</sup> II tenha sido finalizada, assim como é apresentado na Figura 4(a). Caso o usuário queira alterar para o padrão QUARTUS II, basta ir em

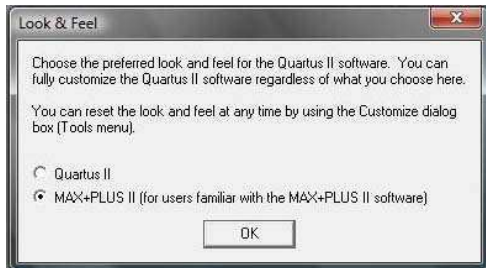
*Tools > Customize*

Aparecerá então a janela apresentada na Figura 4(b) e, assim, pode-se selecionar a opção QUARTUS II. O *software* de desenvolvimento deve obrigatoriamente ser reiniciado para que as alterações sejam efetivadas.

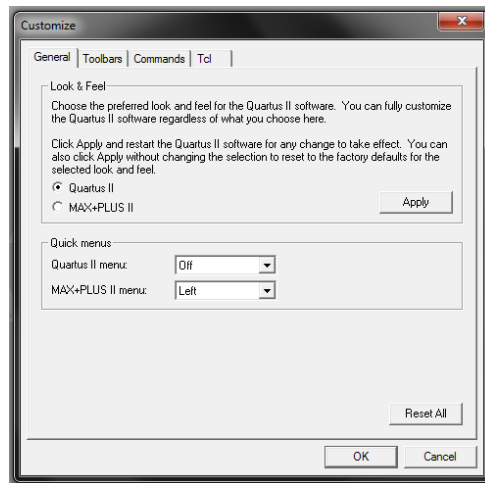
Na primeira vez que o *kit* de desenvolvimento é conectado ao computador, assim como qualquer dispositivo (por exemplo, *pendrive*, cartões de memória, câmeras fotográficas, tocadores de MP3), é necessário que seja instalado o *driver* USB-BLASTER, responsável pela comunicação do *kit* com o computador. Em outras palavras, esse *driver* permite que a configuração do *chip* FPGA (ou em qualquer outro *chip* que utilize esse tipo de controlador) seja carregada nele a fim de que este execute um conjunto de funções específico. Este *driver* é instalado no diretório

---

<sup>6</sup>A título de observação, opções relativas às famílias de FGPA STRATIX, CYCLONE, CYCLONE III, CYCLONE IV, MAX e outros podem ser desconsiderados. Eventualmente, opções relacionadas a outras funcionalidades (tais como tutoriais e arquivos de ajuda) podem ser, caso seja de interesse do usuário, adicionadas durante a instalação.



(a)



(b)

Figura 4: Configuração do ambiente de desenvolvimento: (a) na inicialização do *software* e (b) após a inicialização, em qualquer momento que seja de interesse do usuário.

*<diretório de instalação de Quartus II>/drivers/usb-blaster*

Maiores informações e detalhes sobre como proceder são encontradas a partir de [http://www.altera.com/support/software/drivers/usb-blaster/dri-usb-blaster-xp.html?GSA\\_pos=4&WT.oss\\_r=1&WT.oss=usb-blaster](http://www.altera.com/support/software/drivers/usb-blaster/dri-usb-blaster-xp.html?GSA_pos=4&WT.oss_r=1&WT.oss=usb-blaster).

## 4 Modelando Um Projeto

Antes de criar um projeto no ambiente QUARTUS<sup>®</sup> II, é necessário modelar o problema em uma linguagem processável pelos circuitos digitais. Nesta seção são mostrados os passos de a modelagem de um problema.

Um exemplo prático de projeto a ser empregado ao longo deste tutorial consiste de um circuito lógico digital que implementa uma máquina de estados de Moore, cuja saída depende de uma variável de entrada  $x$  ( $x = "1"$  ou  $x = "0"$ ). Esta máquina é denominada “**Detector de Sequência Zero**”. Basicamente, ela será capaz de detectar uma sequência consecutiva de 3 zeros. Sempre que a sequência é detectada, sinaliza-se com um nível lógico ALTO representado pela variável de saída  $z$ . Maiores informações podem ser adquiridas em (Wu, 2001).

### 4.1 Modelo Comportamental

Uma forma para representar o comportamento do circuito em questão é o diagrama de estados, conforme ilustra Figura 5. A tabela de estados e saídas, apresentada na Tabela 1, é uma representação alternativa para Figura 5.



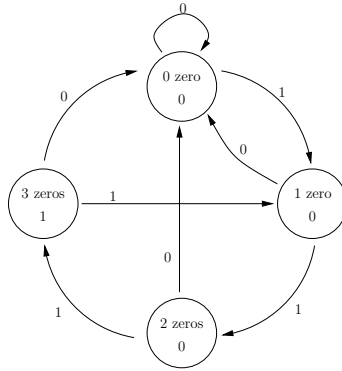


Figura 5: Diagrama de estados do detector de sequência de zeros.

Estado	$x$		$z$
	0	1	
$S_1$ (0 “zero”)	$S_0$	$S_1$	0
$S_2$ (1 “zero”)	$S_0$	$S_2$	0
$S_3$ (2 “zeros”)	$S_0$	$S_3$	0
$S_4$ (3 “zeros”)	$S_0$	$S_1$	1
	Próximo estado		

Tabela 1: Tabela de transições de estados e saídas para o diagrama da Figura 5.

## 4.2 Modelo Estrutural

Para poder implementar o diagrama de estados da Figura 5 com blocos lógicos elementares, como portas lógicas, *latches* e *flip-flops*, é necessário reduzi-lo em funções lógicas definidas sobre variáveis binárias. O primeiro passo consiste, então, em definir quais serão as variáveis de estado. Nesse exemplo, tomando como base o que é apresentado na Tabela 1, são utilizados quatro estados. Conseqüentemente, dois *bits*,  $q_1q_0$ , são suficientes para representá-los.

No segundo passo, deve-se construir a tabela de transições e de saídas para o sistema. Nesse projeto, serão atribuídos (arbitrariamente) os valores 00, 01, 10 e 11 que  $q_1q_0$  podem assumir relativos aos estados  $S_0$ ,  $S_1$ ,  $S_2$  e  $S_3$ , respectivamente. Dessa maneira tem-se Tabela 2.

$q_1q_0$	$x$		$z$
	0	1	
00	00	01	0
01	00	11	0
10	00	10	0
11	00	01	1
	$q_1^*q_0^*$		

Tabela 2: Tabela 1 em códigos binários.

No terceiro passo, definem-se os *flip-flops* a serem utilizados e suas respectivas tabelas de excitações. Para esse projeto, serão empregados 2 *flip-flops* tipo D para armazenar os dois *bits* de estado  $q_1q_0$ . Assim sendo, tem-se a Tabela 3 e a Tabela 4.

$q_1q_0$	$x$	
	0	1
00	0	0
01	0	1
10	0	1
11	0	0

Tabela 3: Tabela de excitação do *flip-flop*  $d_1$  correspondente ao bit  $q_1$ .

$q_1q_0$	$x$	
	0	1
00	0	1
01	0	1
10	0	0
11	0	1

Tabela 4: Tabela de excitação do *flip-flop*  $d_0$  correspondente ao bit  $q_0$ .

No quarto passo, são derivadas as equações de excitação de cada um dos *flips-flops* a partir das Tabelas 3 e 4:

$$\begin{aligned} d_1 &= x \cdot q_0 \\ d_0 &= x \cdot \bar{q}_1 + x \cdot \bar{q}_0 \end{aligned} \quad (2)$$

No quinto passo, deriva-se a equação de saída:

$$z = q_1 \cdot \bar{q}_0. \quad (3)$$

E, assim, é sintetizado o comportamento descrito na Figura 5, ou alternativamente na Tabela 1, em relações lógicas entre os sinais binários de entrada e de estados. Tais funções são facilmente mapeáveis em componentes lógicos e conexões entre as saídas e as entradas destes componentes. Parte-se, então, para criação de um novo projeto no ambiente QUARTUS<sup>®</sup> II.

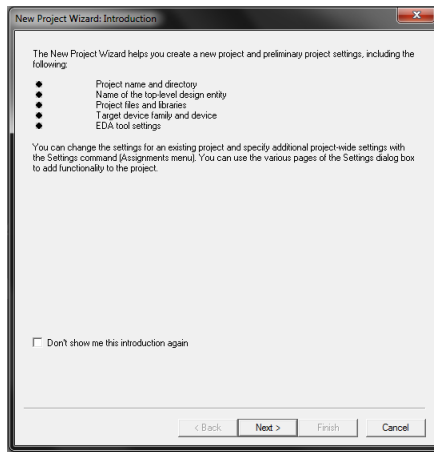
## 5 Criando um Projeto no Quartus<sup>®</sup> II

Para dar início a um projeto no ambiente QUARTUS<sup>®</sup> II, o primeiro passo a ser considerado consiste em criar o arquivo principal de projeto (também denominado *top-level*) já que é a partir dele que a compilação do projeto será feita. Para isso, e tomando como base o FPGA CYCLONE II EP2C20F484C7 presente no *kit* de desenvolvimento DE1, faça o seguinte:

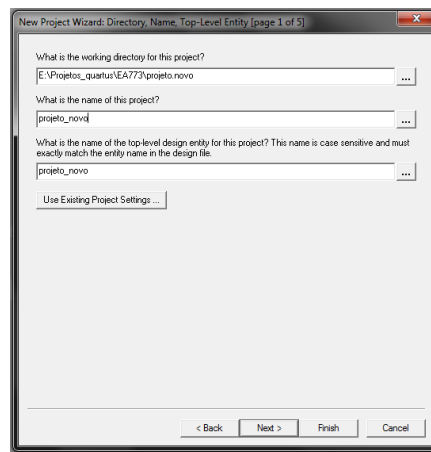
*File > New Project Wizard*

e, assim, a janela apresentada na Figura 6(a) será aberta.

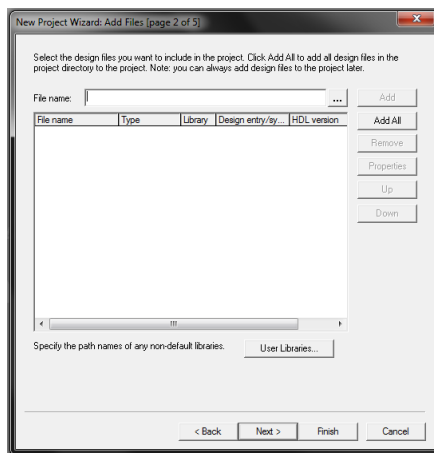
Em seguida, basta repetir os passos apresentados nas Figuras 6(b) a 6(f), tomando o cuidado de escolher o *part number* do dispositivo FPGA corretamente. A título de curiosidade,



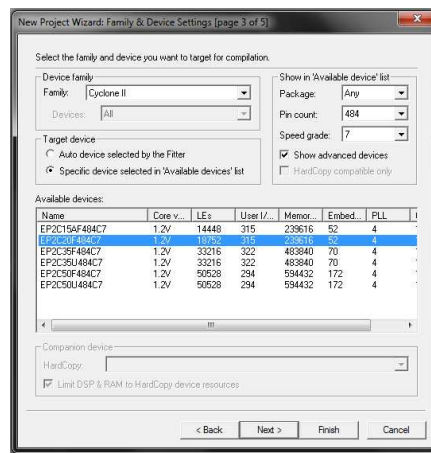
(a)



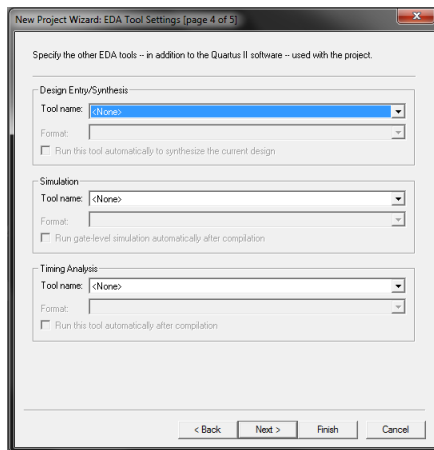
(b)



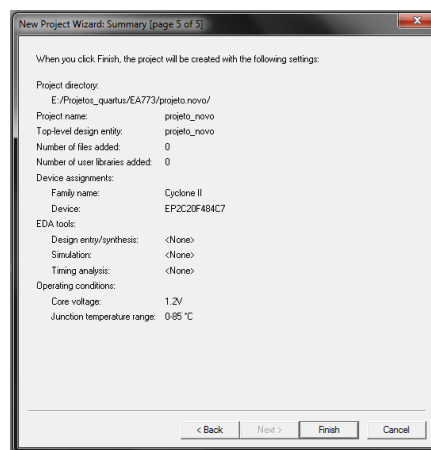
(c)



(d)



(e)



(f)

Figura 6: *Project Wizard* consiste de uma ferramenta de auxílio para a criação de um projeto com uma entidade na raiz (*top-level*): (a) diretrizes de como o projeto será estruturado; (b) definição da pasta na qual o projeto será armazenado, bem como o nome do projeto em si e o nome da entidade raiz; (c) inserção de arquivos desenvolvidos em projetos anteriores; (d) escolha do dispositivo FPGA onde o projeto será programado; (e) escolha de ferramentas de síntese e simulação específicas (de empresas parceiras da Altera); (f) finalização com um resumo das configurações e características do projeto.

vale observar que mesmo desabilitando a instalação de várias famílias de FPGA, no passo correspondente à escolha do dispositivo, todos os modelos de PLDs produzidos pela Altera estão disponíveis ao usuário. É possível escolher outro modelo de FPGA não instalado contudo, deve-se ater ao fato de que quando o projeto for compilado, vários erros serão apresentados devido a falta de suporte para o mesmo.

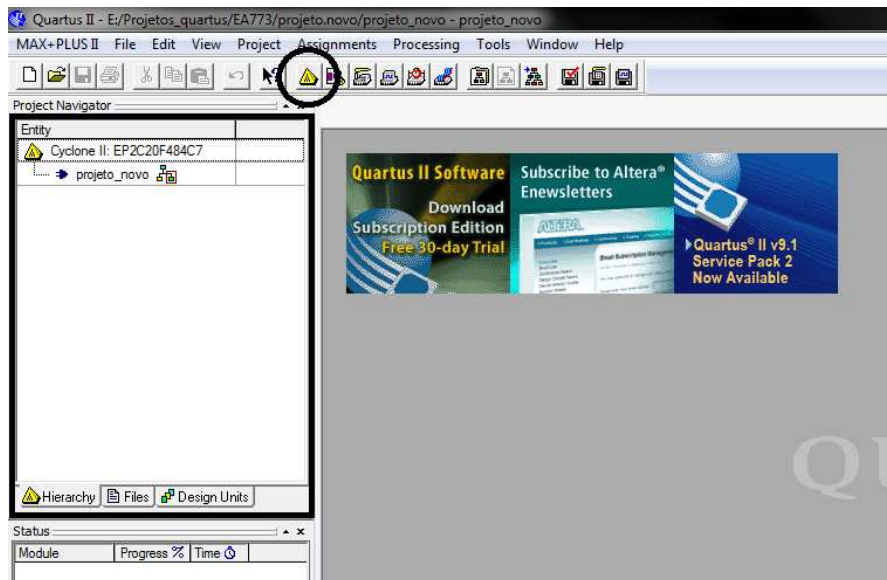


Figura 7: Botão em destaque habilita a hierarquia de projeto. A árvore que representa todos os módulos que compõem o projeto também está destacado.

Finalizado o *Project Wizard*, o QUARTUS® II volta para a sua janela inicial. Caso queira acompanhar ou analisar a hierarquia das entidades que compõem o projeto, clique no botão destacado na Figura 7 e aparecerá uma nova janela mostrando a árvore de hierarquia das entidades do projeto.

## 6 Criando Arquivos de Projeto

O próximo passo consiste na criação de arquivos de projeto para descrever os circuitos presentes nele. No ambiente QUARTUS® II são disponibilizados dois editores para descrevê-los: Editor (Gráfico) de Bloco e Editor de Texto. Isso é porque neste ambiente a construção do circuito pode ser feito em nível de abstração comportamental via uma linguagem de descrição de *hardware* (*Hardware Description Language* - HDL) tais como Verilog, AHDL e VHDL (Pedroni, 2008; Perry, 2002), ou em nível de abstração estrutural via captura de esquemáticos.

Tendo isso em mente, o seguinte procedimento deve ser executado para abrir uma janela apresentada na Figura 8 a fim de criar um arquivo de esquemático:

*File > Block Diagram/Schematic File*

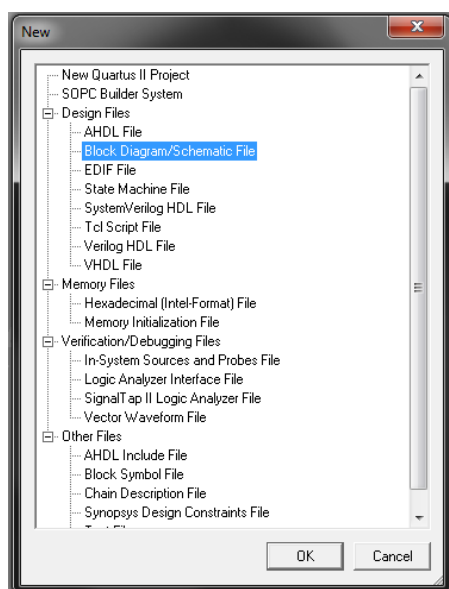


Figura 8: Janela para criação de um arquivo do projeto.

Vale observar que qualquer outro tipo de arquivo de projeto que venha a ser criado e inserido ao projeto, seja outro arquivo de esquemático, seja um arquivo em VHDL ou Verilog, ou até mesmo um arquivo para simulação temporal, pode ser escolhido a partir dessa janela.

Antes de iniciar a edição do arquivo, convém salvar o arquivo com um nome apropriado. Para isso, ative

*File > Save As*

e escolha um nome. De preferência, um nome que resuma o que está sendo projetado. Deve-se prestar atenção ao fato de que abaixo do nome e do tipo do arquivo temos uma opção que deve ser selecionada. Esta opção faz com que o novo arquivo de projeto seja inserido no projeto criado na Seção 5. Lembre-se, também, que em cada pasta deve existir apenas um projeto, mas podem existir inúmeros arquivos de projeto que realizam funções específicas.

## 7 Capturando o Esquemático

Ao longo dessa seção, são apresentados os principais procedimentos a serem empregados para a construção do diagrama eletro-lógico, ou esquemático, dos circuitos eletrônicos digitais com uso do editor gráfico no ambiente QUARTUS® II. O circuito da máquina de estados apresentado na Seção 4 será utilizado como base para os procedimentos apresentados. O arquivo de projeto que foi aberto para construí-lo será salvo como `detectorSequencia.bdf` (Figura 9).



Figura 9: Salvando o arquivo recém criado.

## 7.1 Inserção de Componentes

A construção de um esquemático pode ser feita a partir da utilização de componentes primitivos (tais como portas lógicas e *flip-flops*), de blocos que realizam as funcionalidades de circuitos integrados comerciais (7485, 7404, 74162) e, até mesmo, blocos desenvolvidos em linguagem de descrição de *hardware*.

Para inserir um componente existente, deve-se clicar duas vezes na área de trabalho do editor gráfico (ou fazer *Edit > Insert Symbol*). Em seguida, no campo NAME, deve-se digitar o *part number* do componente ou o seu nome (por exemplo, 7408 ou *and2*). O resultado para essa ação é exemplificado na Figura 10. Vale observar que, caso a opção *Repeat-insert mode* seja selecionada, o usuário pode adicionar quantos componentes se queira e, quando o número desejado tenha sido alcançado, basta clicar com o botão direito do *mouse* e optar por cancelar a ação.

Vale comentar aqui que no ambiente QUARTUS® II dispõe-se de uma funcionalidade que facilita a inserção dos pinos de entrada e de saída. Basta selecionar cada componente que tem pinos de entrada e saída (clicar o botão esquerdo do *mouse* em cima dele), apertar o botão direito do *mouse* e selecionar o item *Generate Pins for Symbol Ports* no *pop-up menu* que vai surgir.

## 7.2 Ligação entre Componentes

Inseridos os componentes necessários ao projeto, deve-se fazer as ligações. Essa é uma tarefa bastante simples de ser realizada. Existem duas ligações básicas que podem ser reali-

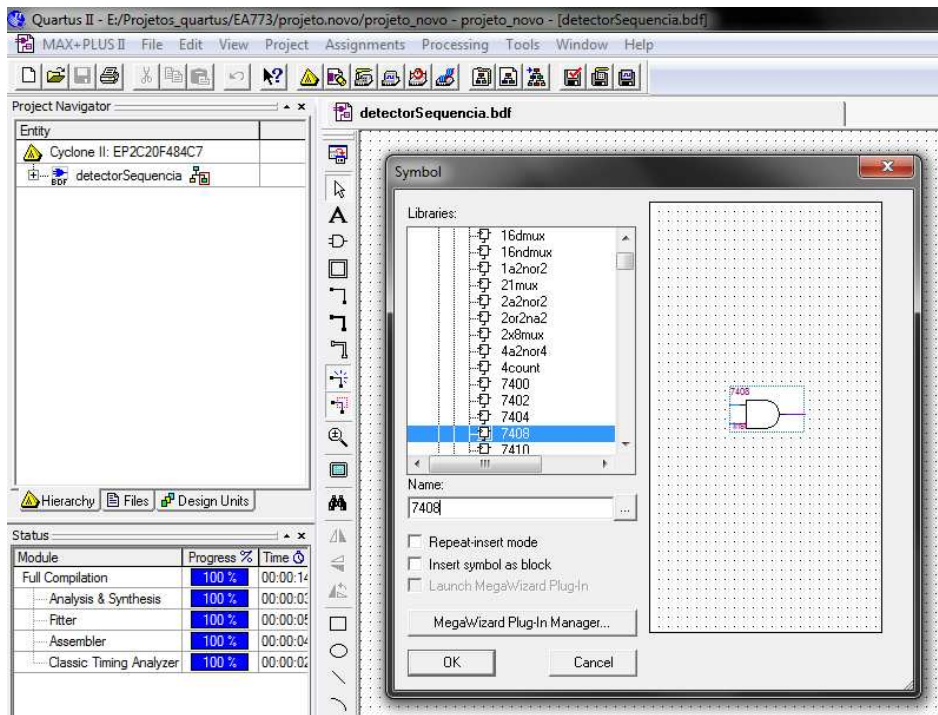


Figura 10: Inserção de um componente 7408.

zadas e duas maneiras distintas de se fazer isso. O primeiro modo de se fazer uma ligação entre componentes consiste em aproximar o *cursor* na porta que se deseja conectar. O *cursor* automaticamente muda o seu formato para o de uma cruz e, assim, o modo de ligação entre os componentes torna-se ativo. Uma segunda maneira consiste em selecionar um dos botões destacados na Figura 11.

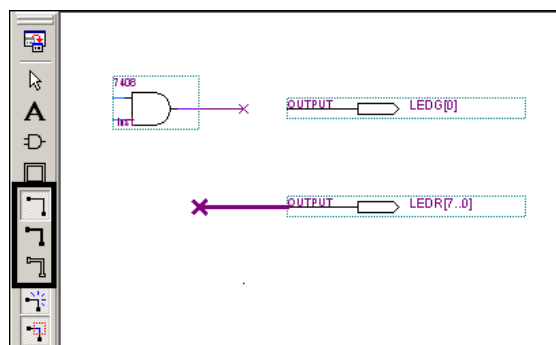


Figura 11: Edição de ligações.

Dentre os tipos mais comuns de ligações que podem ser realizados no QUARTUS® II são as ligações simples e os barramentos. As ligações simples transportam apenas um sinal (ou *bit*) de dados por vez enquanto que os barramentos transportam  $n$  sinais (ou *bits*) de uma vez. Estes sinais podem transportar dados, endereços ou sinais de controle (Tocci *et al.*, 2008).

Na Figura 11 a ligação simples é representada graficamente pela linha mais fina (podendo ser criada ao ativar o botão superior em destaque) e o barramento pela linha mais grossa (no destaque, ativado pelo botão central). Vale observar ainda que a convenção adotada para

indicar o tamanho  $n$  do barramento LEDR é  $\text{LEDR}[n - 1..0]$ .

### 7.3 Nomeação dos Pinos de Entrada e de Saída

Um ponto que vale ser ressaltado consiste na denominação dos pinos de entrada e de saída, que devem ser conectados aos pinos dos periféricos. No *kit* de desenvolvimento DE1, os pinos dos periféricos disponíveis já se encontram conectados a alguns pinos do *chip* FPGA. Portanto, para utilizar tais periféricos nos testes de campo, é necessário associar, ou mapear, apropriadamente os pinos de entrada e de saída do circuito aos pinos do *chip* FPGA. No jargão de laboratório, este procedimento é conhecido como *assignment* dos pinos. Isso pode ser feito manualmente, abrindo uma janela através de

*Assignments > Pins.*

Uma outra alternativa, muito mais simples e altamente recomendada, consiste em utilizar os nomes adotados no manual da placa de desenvolvimento DE1 e importar o arquivo `DE1_pin_assignments.csv` onde se encontram definidas todas as associações. Este arquivo é disponibilizado em (Altera e Terasic, 2010)<sup>7</sup>. Aconselha-se que o arquivo `DE1_pin_assignments.csv` seja copiado para o diretório de projeto. Para importar o *assignment*, deve-se abrir a caixa de diálogo da Figura 12 via

*Assignments > Import Assignments*

Em seguida, deve-se inserir no campo *File Name* o nome do arquivo e confirmar a ação. Esse procedimento deve ser realizado somente uma vez, antes da primeira compilação do projeto. Todas as modificações e compilações futuras não necessitam que o procedimento de importação seja realizada novamente.

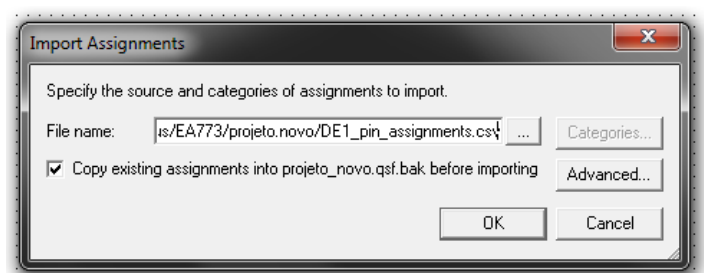


Figura 12: Janela para importação do arquivo com os *assignments* dos pinos para a placa DE1.

A (re)nomeação dos pinos de entrada (*input*) e de saída (*output*) é feita simplesmente clicando-se duas vezes nos respectivos pinos. No projeto em questão, usamos os nomes `SW[0]`, `KEY[0]` e `LEDG[0]`. Na Figura 13 tem-se o circuito lógico finalizado da máquina de estados, com os pinos devidamente associados aos pinos dos periféricos.

<sup>7</sup>O arquivo está também disponível na página do curso de EA773 a partir do 1º semestre de 2010.



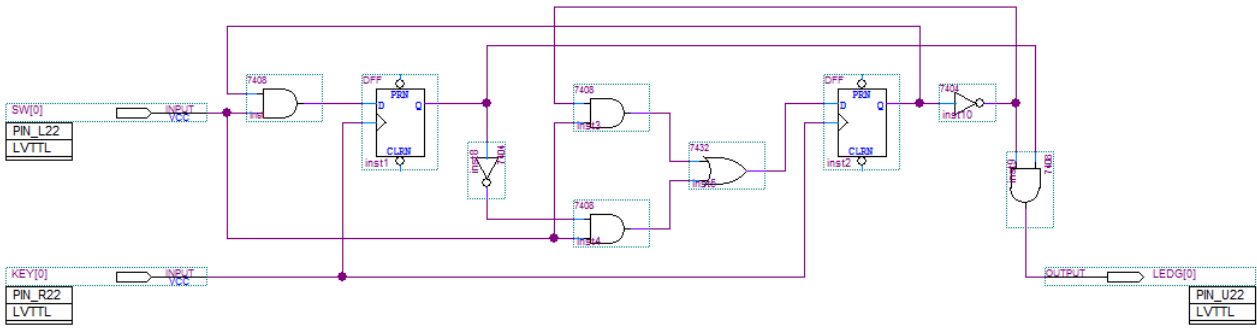


Figura 13: Esquemático da máquina de estados com pinos mapeados nos periféricos.

## 7.4 Nomeação das Linhas das Ligações

O QUARTUS® II possui uma funcionalidade bastante interessante e extremamente útil que consiste na nomeação das ligações entre os diversos componentes e pinos de entrada/saída que compõem um projeto. Uma forma de fazer isso é clicar em cima da linha de ligação que se queira nomear e apertar o botão esquerdo do *mouse* sobre a linha selecionada. Em seguida, escolha o item *Properties* do menu que surgirá e escreva no campo *Name* o nome da linha.

Essa característica funcional do QUARTUS® II permite ao projetista realizar ligações “virtuais”, ou seja, permite que a saída de um determinado componente seja conectada a uma entrada de um outro sem a necessidade da ligação física, somente nomeando-se as ligações. Na Figura 14 tem-se uma aproximação do esquemático que descreve a máquina de estados apresentada na Figura 13. Esse é um exemplo de como a ligação “virtual” pode ser implementada.

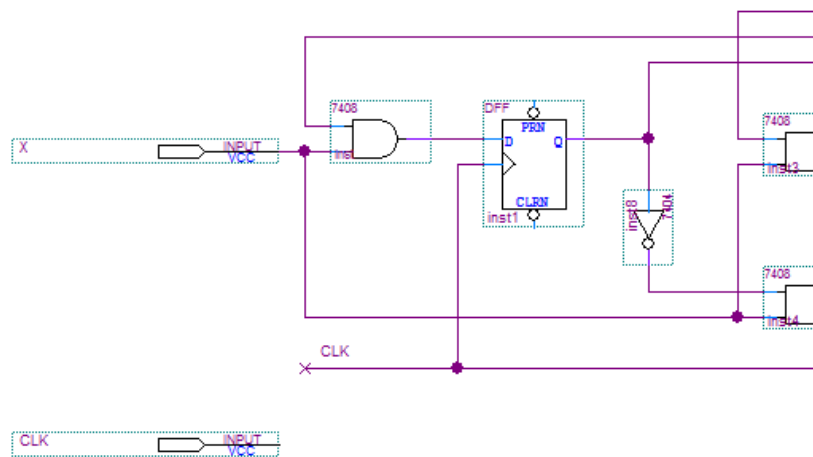


Figura 14: Ligação “virtual”.

Uma outra possibilidade para essa ferramenta está relacionada à análise das formas de ondas e resposta temporal do circuito sendo que os procedimentos necessários para gerar uma simulação temporal serão apresentados na Seção 13. Assim sendo, a título de observação, a nomeação de ligações permite que determinados sinais internos do circuito (por exemplo, um sinal de controle que ative um *latch*) sejam mapeados em pinos de saída e, com isso, possam

ser analisados no arquivo de simulação, com o auxílio dos diagramas temporais. Na Figura 15 é destacado o circuito de detecção de sequência zero onde as saídas dos *flip-flops* estão mapeados em pinos de saída, possibilitando a análise da resposta temporal durante a simulação do circuito.

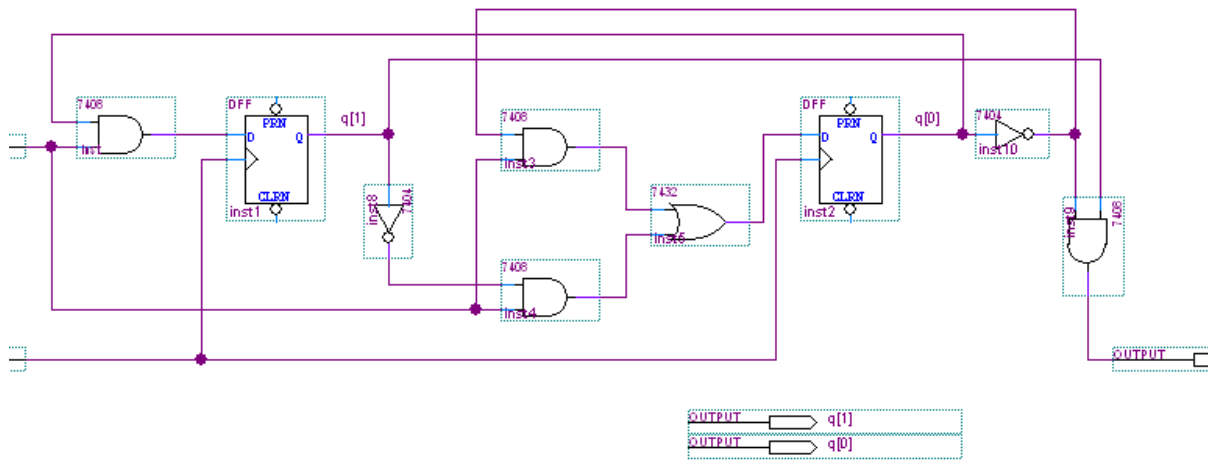


Figura 15: Ligação “virtual” entre os sinais intermediários  $q[1]$  e  $q[0]$  e os pinos de saída.

## 8 Descrevendo em VHDL

Descrever um circuito digital através de uma linguagem gráfica como esquemáticos tornou-se, hoje em dia, bem menos popular que utilizar ferramentas baseadas em linguagens textuais. Usualmente fazer uma descrição textual é menos trabalhoso e menos complexo que elaborar um esquemático. O ambiente QUARTUS<sup>®</sup> II dispõe de compiladores lógicos que convertem essas descrições textuais em arquivos binários transferíveis ao dispositivo lógico programável via a conexão serial do PC (Seção 10).

Para a descrição de circuitos em VHDL dentro do QUARTUS<sup>®</sup> II (ou, também, empregando Verilog ou AHDL), é necessário criar um arquivo específico acessando a opção VHDL File na janela apresentada na Figura 8. Um editor de texto será ativado e o arquivo aberto será salvo com a extensão `.vhd`. Quanto ao nome do arquivo, recomenda-se que seja algo que lembre da funcionalidade do circuito. **Este nome deve ser o mesmo nome da seção ENTITY do componente de VHDL.** No caso, o arquivo será salvo como `detector_funcional.vhd`. O Código 2 é o conteúdo do arquivo, que corresponde à descrição funcional, em VHDL do detector de sequência zero apresentada na Seção 4.1.

É possível implementar o modelo estrutural de um projeto em VHDL. Neste caso, a descrição consiste da declaração dos componentes lógicos utilizados e das ligações entre as portas de entrada e saída destes componentes. É uma versão textual do esquemático, como ilustra o Código 3 salvo como `detector_estrutural.vhd`.

Vale observar que qualquer editor de texto pode ser usado para construir/modificar um código de descrição de *hardware* desde que os cuidados quanto ao nome e extensão do arquivo

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity detector_funcional is
5      port(
6          clk      : in  std_logic;
7          x        : in  std_logic;
8          z        : out std_logic
9      );
10 end entity;
11
12 architecture comportamento of detector_funcional is
13     type estados is (s0, s1, s2, s3);
14     signal state : estados;
15 begin
16     process (clk)
17     begin
18         if(rising_edge(clk)) then
19             case state is
20                 when s0=>
21                     if x = '1' then
22                         state <= s1;
23                     else
24                         state <= s0;
25                     end if;
26                 when s1=>
27                     if x = '1' then
28                         state <= s2;
29                     else
30                         state <= s0;
31                     end if;
32                 when s2=>
33                     if x = '1' then
34                         state <= s3;
35                     else
36                         state <= s0;
37                     end if;
38                 when s3 =>
39                     if x = '1' then
40                         state <= s1;
41                     else
42                         state <= s0;
43                     end if;
44             end case;
45         end if;
46     end process;
47     process (state)
48     begin
49         case state is
50             when s0 =>
51                 z <= '0';
52             when s1 =>
53                 z <= '0';
54             when s2 =>
55                 z <= '0';
56             when s3 =>
57                 z <= '1';
58         end case;
59     end process;
60 end comportamento;

```

Código 2: Implementação do modelo funcional do detector em VHDL.

sejam tomados. A vantagem de utilizar o editor de texto do ambiente QUARTUS® II é que ele dispõe de um conjunto de modelos de programação para uma grande variedade de circuitos combinacionais e sequenciais.

O ambiente Quartus II dispõe de visualizadores gráficos do resultado da síntese (Seção 11). Recomenda-se esta prática porque o procedimento de compilação nem sempre gera os

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity detector_estrutural is
5      port(
6          clk      : in  std_logic;
7          x        : in  std_logic;
8          z        : out std_logic
9      );
10 end;
11
12 architecture comportamento of detector_estrutural is
13     signal d1, d0, q0, q1, t1, t0: std_logic;
14     component dff
15         port(D: in std_logic;
16             CLK: in std_logic;
17             Q: out std_logic);
18     end component;
19     component and2
20         port(IN1: in std_logic;
21             IN2: in std_logic;
22             \OUT\: out std_logic);
23     end component;
24     component or2
25         port(IN1: in std_logic;
26             IN2: in std_logic;
27             \OUT\: out std_logic);
28     end component;
29 begin
30     DFF0: DFF
31         port map (d0, clk, q0);
32     DFF1: DFF
33         port map (d1, clk, q1);
34     U0: AND2
35         port map (x, not q1, t0);
36     U1: AND2
37         port map (x, not q0, t1);
38     U2: OR2
39         port map (t0, t1, d0);
40     U3: AND2
41         port map (x, q0, d1);
42     U4: AND2
43         port map (not q0, q1, z);
44
45 end comportamento;

```

Código 3: Implementação do modelo estrutural do detector em VHDL.

melhores resultados. Modificações podem ser necessárias para direcionar melhor o algoritmo de compilação no processo de síntese.

## 9 Encapsulando os Componentes

Foi apresentado na Seção 2.2 que no ambiente de desenvolvimento de projeto QUARTUS® II os arquivos de projeto são organizados hierarquicamente. É conveniente que as entidades neles descritas sejam encapsuladas em componentes de forma que somente a sua função e a sua interface de entrada e saída sejam vistas pelas entidades de nível mais alto ou até mesmo em outros projetos<sup>8</sup>.

Símbolos de componentes podem ser criados de forma muito simples no ambiente de

---

<sup>8</sup>Esse procedimento é análogo à criação de funções, seja, por exemplo, usando linguagem C ou *scripts* do MATLAB.

desenvolvimento QUARTUS® II a partir de códigos em VHDL ou a partir de arquivos de esquemáticos. Primeiramente, deve-se tomar o cuidado de tornar o arquivo de projeto na entidade de maior hierarquia (*Top-Level Entity*) e de tornar a janela de edição o foco. Em seguida, deve-se selecionar

*File > Create/Update > Create Symbols File from Current File*

e, logo em seguida, é aberta uma janela para salvar o arquivo/símbolo correspondente (extensão `.bsf`). No caso do projeto da máquina detectora, foi criado um arquivo de símbolo para o circuito da Figura 15, sendo salvo como `detectorSequencia.bsf`.

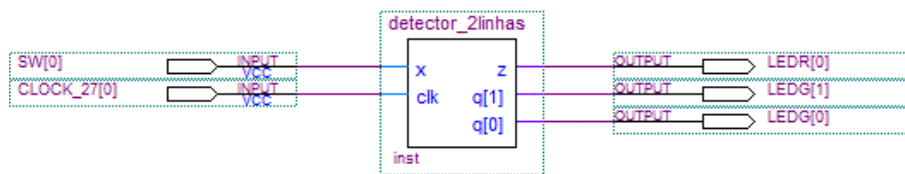


Figura 16: Inserção de um componente criado em um arquivo de projeto.

Para inserir o componente criado em um esquemático, basta escolher o nome do componente nos diretórios após clicar o botão ao lado do campo *Name* na Figura 10. Na Figura 16 tem-se a ilustração de um esquemático com o componente `detectorSequencia.bsf`. Ressalta-se que o nome atribuído a cada pino de entrada e de saída no arquivo de projeto é preservado no símbolo criado. Se as portas de saída `q[1]` e `q[0]` forem agrupadas em um vetor `q[1..0]`, tem-se um circuito como o apresentado na Figura 17, a partir do qual é criado um símbolo com uma porta `q[1..0]`, ao invés de 2 portas (`q[1]` e `q[0]`), como mostra o circuito da Figura 18. Dessa maneira, sugere-se que o usuário empregue nomes que resumam a funcionalidade do sinal. Vale comentar que os cuidados quanto a espaços nos nomes devem ser adotados aqui (por exemplo, uma entrada do tipo `entrada 01` deve ser evitada; prefira `entrada01` ou algo do gênero).

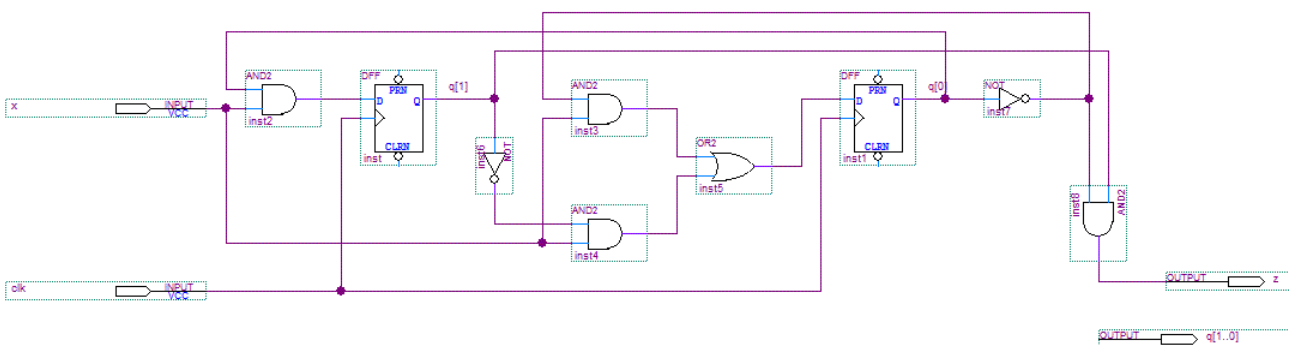


Figura 17: Agrupamento de mais de uma linha em uma porta de saída.

Para inserir um componente em um arquivo de projeto em VHDL, basta declarar a interface do componente. Esta interface pode ser gerada no ambiente QUARTUS® II após setar

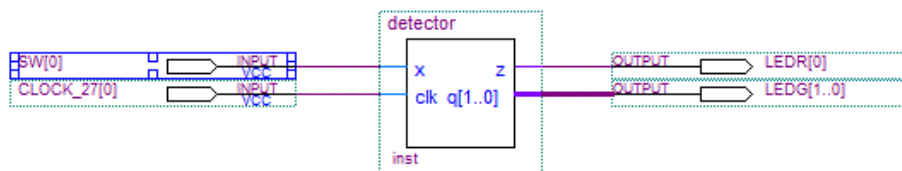


Figura 18: Uso de barramento na porta com mais de uma linha.

o componente como o de nível mais alto e criar o código de declaração do componente por meio de

*File > Create/Update > Create VHDL Component Declaration Files for Current File*

O Código 3 exemplifica a declaração de três componentes: `dff`, `and2` e `or2`.

Pensando em projetos de grande porte, é mais eficaz que os componentes sejam testados e simulados separadamente, pois garante-se que cada um esteja funcionando corretamente. Assim, se ocorrer problemas na integração, a depuração do circuito fica mais simples já que os blocos foram testados previamente. Outra vantagem de compilar e testar os componentes individualmente é a possibilidade de utilização destes componentes em outros projetos. Os procedimentos de compilação e simulação são detalhados, respectivamente, na Seção 10 e na Seção 13.

## 10 Compilando

A compilação é um processo necessário visto que vai analisar se existe algum erro relacionado ao projeto, tal como curto-circuitos e sinais duplicados, além de otimizá-lo e sintetizá-lo em elementos lógicos disponíveis no FPGA selecionado. Por exemplo, uma das otimizações realizadas consiste em que a área ocupada pelo circuito seja menor, sem comprometer a velocidade de operação do circuito. Nesta etapa são também pré-avaliados os requisitos do projeto para verificar se o circuito sintetizado irá atendê-los. Um outro ponto que deve ser levado em consideração consiste no fato de que é ao final dessa etapa que serão gerados os arquivos binários necessários tanto para a simulação do todo o, ou parte do, projeto (Seção 13) quanto para a configuração do FPGA do *kit* de desenvolvimento (Seção 14).

Inicialmente, deve-se lembrar que o QUARTUS® II trabalha com o conceito de hierarquia, como discutido na Seção 2.2. Assim sendo, é fundamental que o arquivo de projeto, tanto em VHDL quanto em esquemático, que se pretende compilar seja o de maior hierarquia. Para garantir que o arquivo de projeto aberto seja o de maior nível hierárquico na compilação, recomenda-se que se coloque o editor do arquivo em foco e selecione

*Project > Set as Top-Level Entity*

ou simplesmente use a tecla de atalho CTRL+SHIFT+J. No exemplo, o arquivo `detectorSequencia.bdf` deve ser setado como o topo da árvore. Na Figura 19 é destacada a mensagem de que a operação foi executada com sucesso.

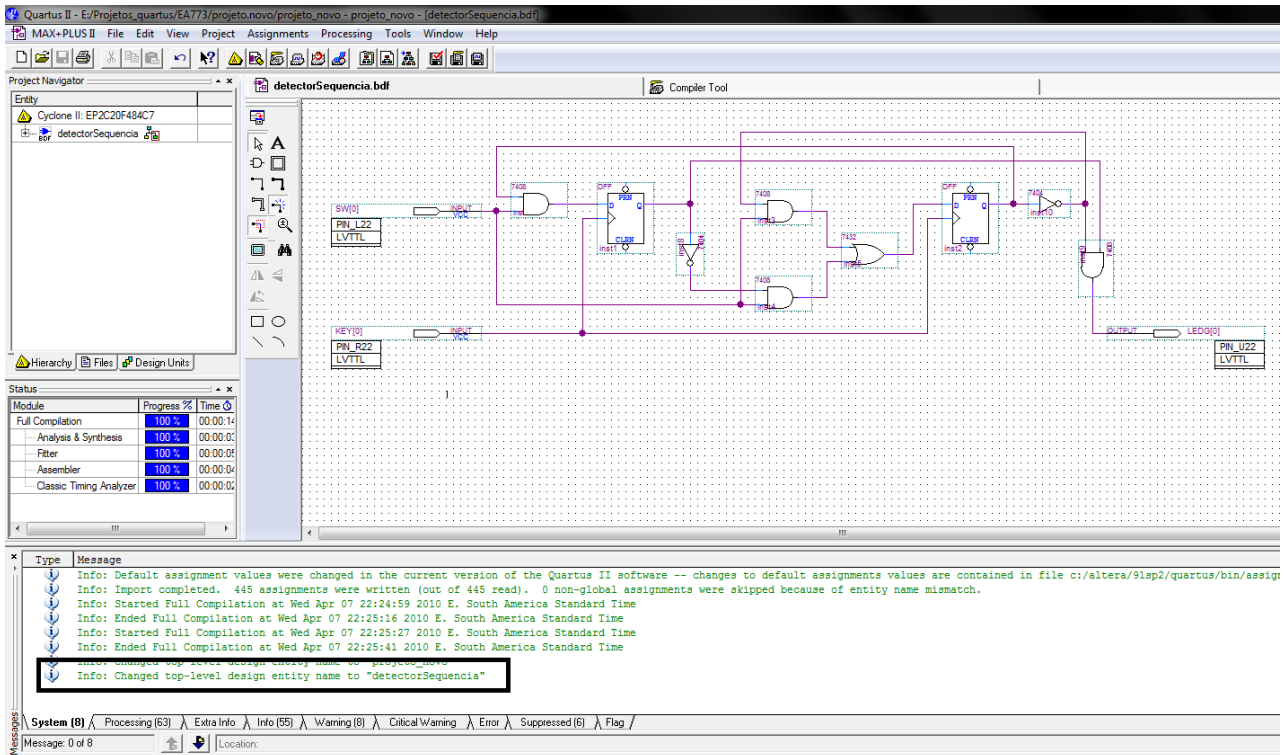


Figura 19: Mensagem da atual subarvore com `detectorSequencia.bdf` no topo.

Agora que o arquivo `detectorSequencia.bdf` está habilitado como o *top-level*, deve-se evocar

*MAX+Plus II > Compiler*

(ou tecla de atalho CTRL+L) para compilá-lo. A Figura 20 apresenta a janela que mostra o progresso da compilação.

Observa-se que as quatro etapas do fluxo completo de compilação são mostradas na janela: *Analysis & Synthesis*, *Fitter*, *Assembler* e *Classic Timing Analysis*. Ao concluir este fluxo, pode-se analisar o relatório sobre o processo clicando o botão **Report**. É possível selecionar o tópico do relatório escolhendo um dos itens na coluna esquerda da janela que surgirá. Por exemplo, para o circuito de detector de sequência zero, ao selecionar

*Flow Summary*

a janela ficou como a apresentada na Figura 21. Ela mostra que o circuito foi sintetizado, com sucesso, para o dispositivo EP2C20F484C7 da família Cyclone II com uso de 3 dos 18.752 elementos lógicos.

E ao selecionar

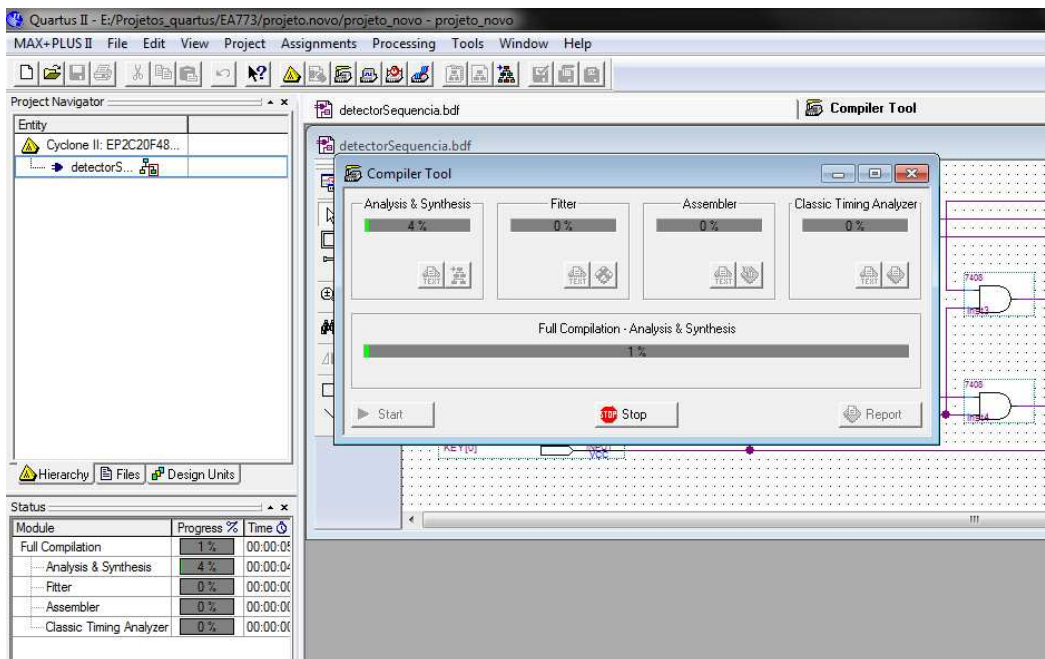


Figura 20: Janela de compilação.

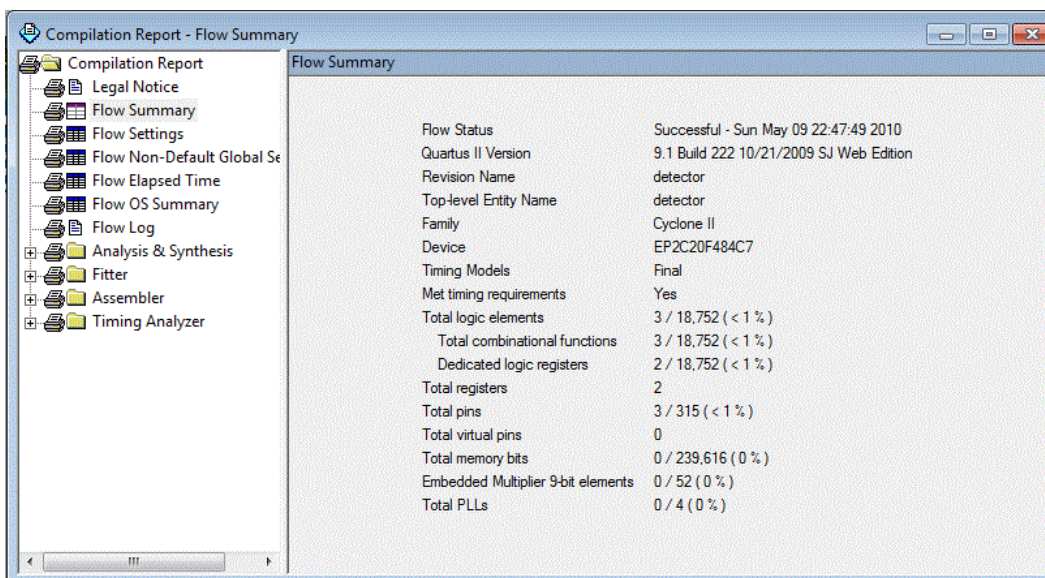


Figura 21: Resumo do fluxo de compilação.

*Timing Analyzer > Summary*

obtem-se o resumo de desempenho temporal apresentado na Figura 22.

## 11 Visualizando os Resultados da Compilação

Para facilitar a análise e a depuração de um projeto, o QUARTUS® II dispõe de alguns visualizadores gráficos de NETLIST sintetizado: RTL VIEWER, STATE MACHINE VIEWER e TECHNOLOGY MAP VIEWER. O primeiro mostra o resultado da síntese a nível de registros de



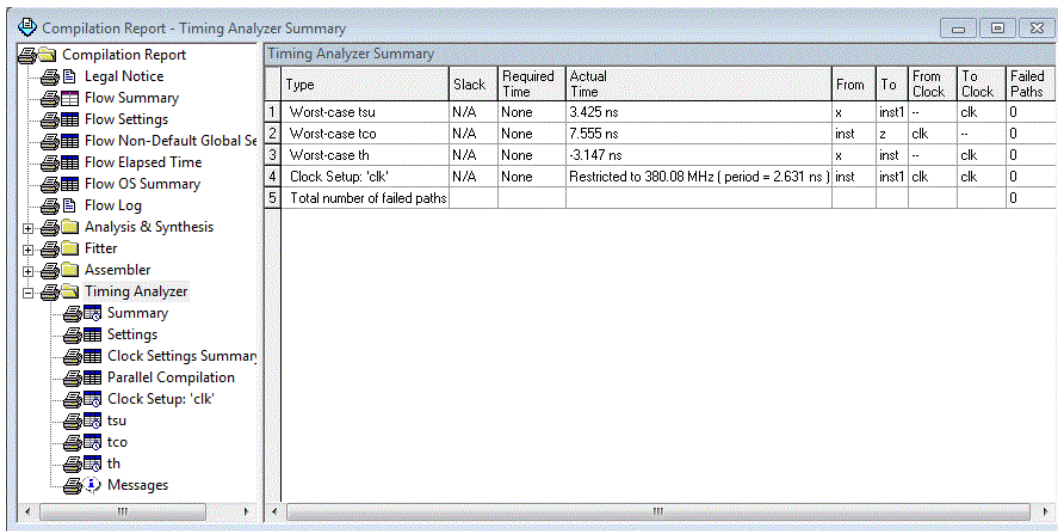


Figura 22: Sumário sobre o desempenho temporal do projeto.

transferências (*Register Transfer Level – RTL*), que consiste basicamente em uma representação por registradores interligados por lógica combinacional, e o segundo, os estados e as transições entre os estados do projeto. O último visualizador, por sua vez, apresenta a hierarquia de unidades atômicas (células e portas de entrada/saída) utilizadas na síntese do projeto. Figura 23 ilustra as três formas de visualização da síntese do Código 2, correspondente à especificação do projeto de detector de sequência zero (seção 4).

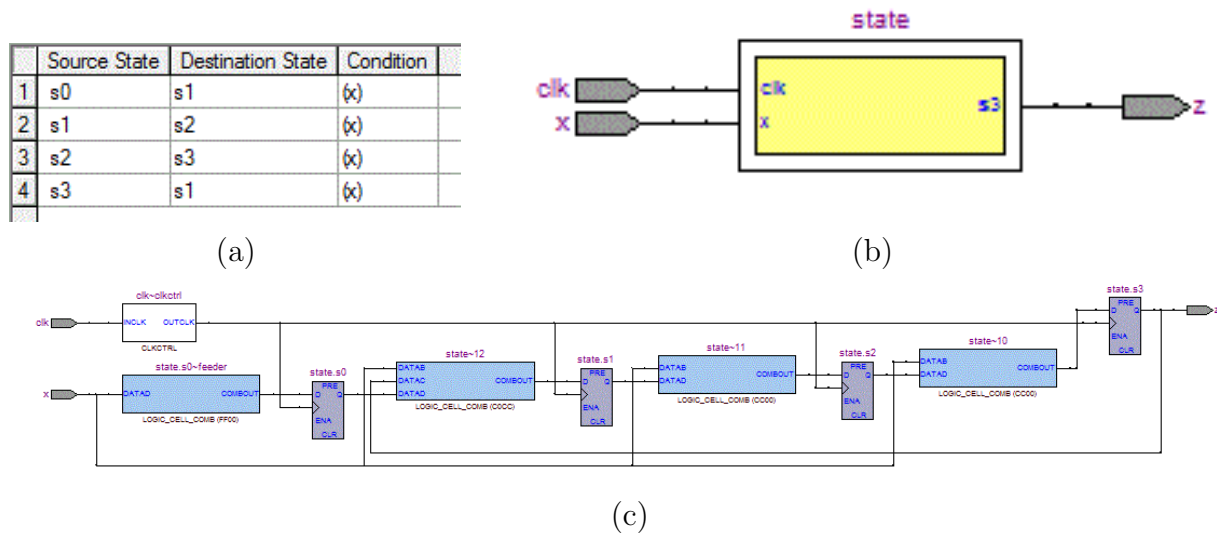


Figura 23: Distintas formas de visualização gráfica de NETLIST: (a) STATE MACHINE VIEWER, (b) RTL VIEWER, e (c) TECHNOLOGY MAP VIEWER.

A sequência de comandos utilizados para habilitar estes visualizadores é:

*Tools > Netlist Viewers*

## 12 Analisando as Restrições de Tempos do Projeto

No ambiente QUARTUS® II encontram-se dois analisadores de tempos estáticos: *Classic Timing Analyzer* e *TimeQuest Timing Analyzer*. Embora a Altera recomende o uso do segundo analisador, o primeiro analisador é suficiente para o propósito da disciplina EA773. Para utilizar o primeiro analisador, é necessário configurar o ambiente para tal. Abre-se a caixa de diálogo de configuração através de

*Assignments > Settings*

e seleciona-se *Timing Analysis Settings* na lista *Category* para ativar *Use Classic Timing Analyzer during compilation*. Nesta mesma caixa de diálogo é possível especificar as restrições temporais do seu projeto que são utilizadas como referência no cômputo dos tempos. Os tempos com valores positivos indicam margens positivas em relação à restrição imposta (restrição satisfeita) e os valores negativos correspondem às margens negativas (restrição não atendida).

O analisador classifica, em primeiro lugar, todos os possíveis caminhos de sinais em: os do sinal de relógio (*CLOCK*), os de dados, e os de sinais de controle assíncronos como os PRN e CLRN nos *flip-flops*. Em seguida, são estimados os seguintes tempos de percurso dos sinais entre os pinos de entrada e saída do circuito e os registradores internos:

- $t_{SU}$ : corresponde ao intervalo de tempo que o sinal de dado deve chegar e estabilizar antes da ocorrência de uma transição do sinal de relógio;
- $t_H$ : corresponde ao intervalo de tempo que o sinal de dado precisa se manter estável após a ocorrência de uma transição do sinal de relógio.
- $t_{CO}$ : corresponde ao (mínimo e máximo) intervalo de tempo necessário para obter uma saída válida após a ocorrência de uma transição do sinal de relógio.

Os tempos de propagação dos sinais através da parte combinacional do circuito também são determinados:

- $t_{PD}$ : corresponde ao (mínimo e máximo) intervalo de tempo para um sinal propagar de um pino de entrada do circuito até um pino de saída via os elementos combinacionais.

A frequência máxima de operação do sinal de relógio (*Registered Performance*), sem que as restrições temporais de todos os sinais do circuito sejam violadas, é também computada.

Vale ainda comentar que o analisador de tempos gera advertências quando ocorre o fenômeno *clock skew*, que consiste na diferença entre os instantes que um mesmo sinal de *clock* chega em dois registradores distintos. Isso nem sempre implica em violações das restrições

temporais do projeto já que às vezes é necessário introduzir propositalmente os atrasos para compatibilizá-los com os atrasos dos sinais de dados ao longo do seu percurso.

Para acessar todos os dados temporais gerados durante a compilação, deve-se selecionar

*MAX+PLUS II > Timing Analyzer*

e, assim, aparecerá uma pasta com 6 abas, como é ilustrado na Figura 24. Caso seja de interesse ter o conhecimento dos detalhes de um dos tempos listados anteriormente, basta clicar na aba correspondente.

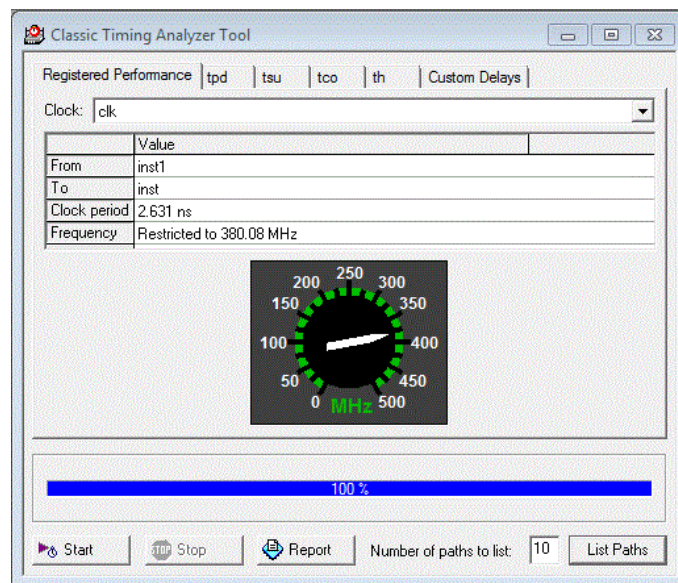


Figura 24: Interface do Analisador de Tempo Clássico.

## 13 Simulando

O próximo passo consiste em realizar simulações que representem o comportamento dos periféricos, o mais próximo da realidade possível. As simulações são utilizadas também para verificar o funcionamento do circuito de maior hierarquia pela análise das formas de onda, verificando a integridade dos sinais e, assim, comprovando o correto funcionamento do sistema antes de carregá-lo no dispositivo FPGA do *kit* DE1. Tendo isso em mente, para realizar a simulação, é preciso criar um arquivo específico, denominado **Vector Waveform File**, simplesmente fazendo *File > New* e selecionando a opção correspondente, como é ilustrado na Figura 25. Outra forma alternativa seria

*MAX+Plus II > Waveform Editor*

Feito isso, o ambiente de trabalho assumirá uma forma semelhante à apresentada na Figura 26.

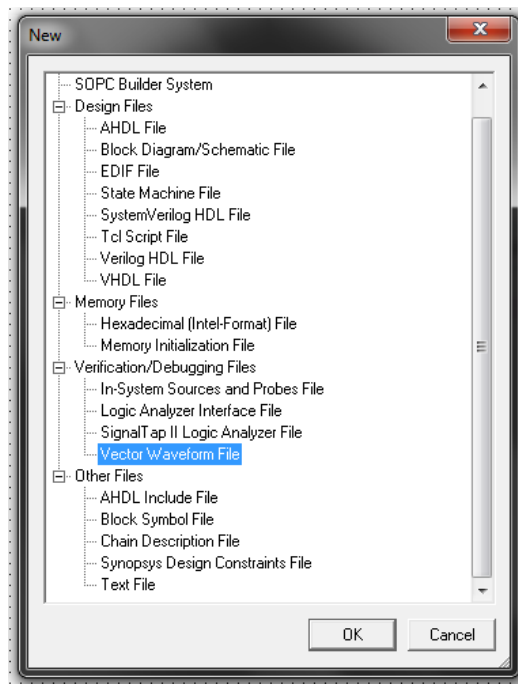


Figura 25: Criando o vetor de análise de formas de ondas.

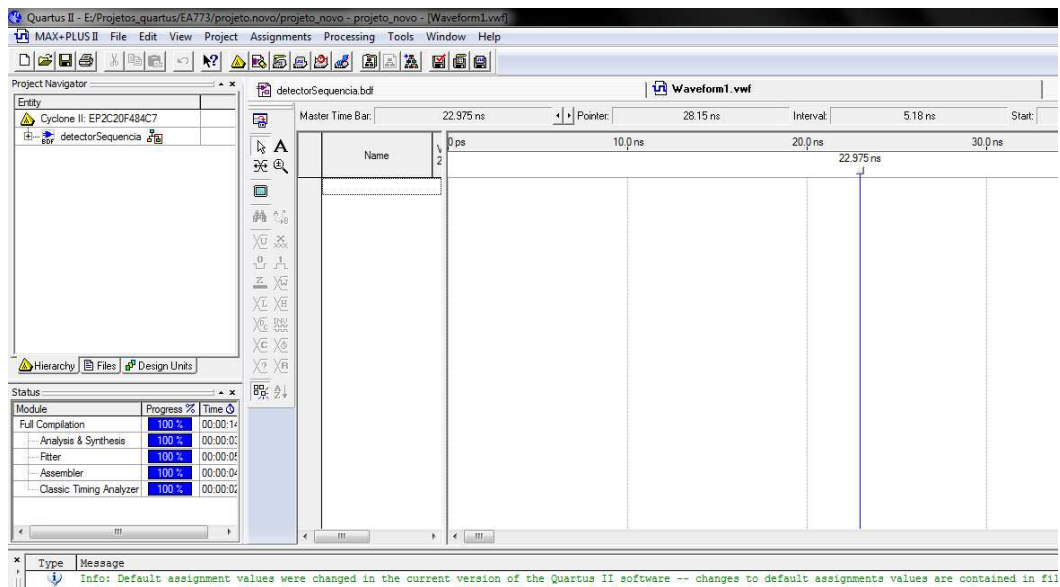


Figura 26: Arquivo de formas de ondas criado.

Criado o arquivo de forma de onda, é necessário acrescentar todos os sinais que serão analisados. Para isso, deve-se fazer

*Edit > Insert > Insert Node or Bus*

e a janela ilustrada na Figura 27 deve aparecer. Em seguida deve-se clicar em *Node Finder* e uma nova janela, como mostrado na Figura 28, irá aparecer. Nesse ponto, deve-se prestar atenção se o arquivo *detectorSequencia.bdf* está selecionado em *Look in* e se o filtro está selecionado como *Pins:all* (nessa opção, todos os pinos de entrada e de saída serão apresentados

e poderão ser inseridos no arquivo de forma de ondas). Feito isso, seleciona-se os pinos desejados e clica-se no botão em destaque da Figura 28. Opcionalmente, podemos clicar no segundo botão para selecionar todos os sinais simultaneamente.

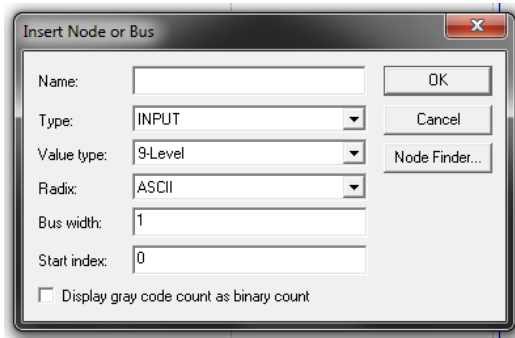


Figura 27: Caixa de diálogo para inserção de sinais.

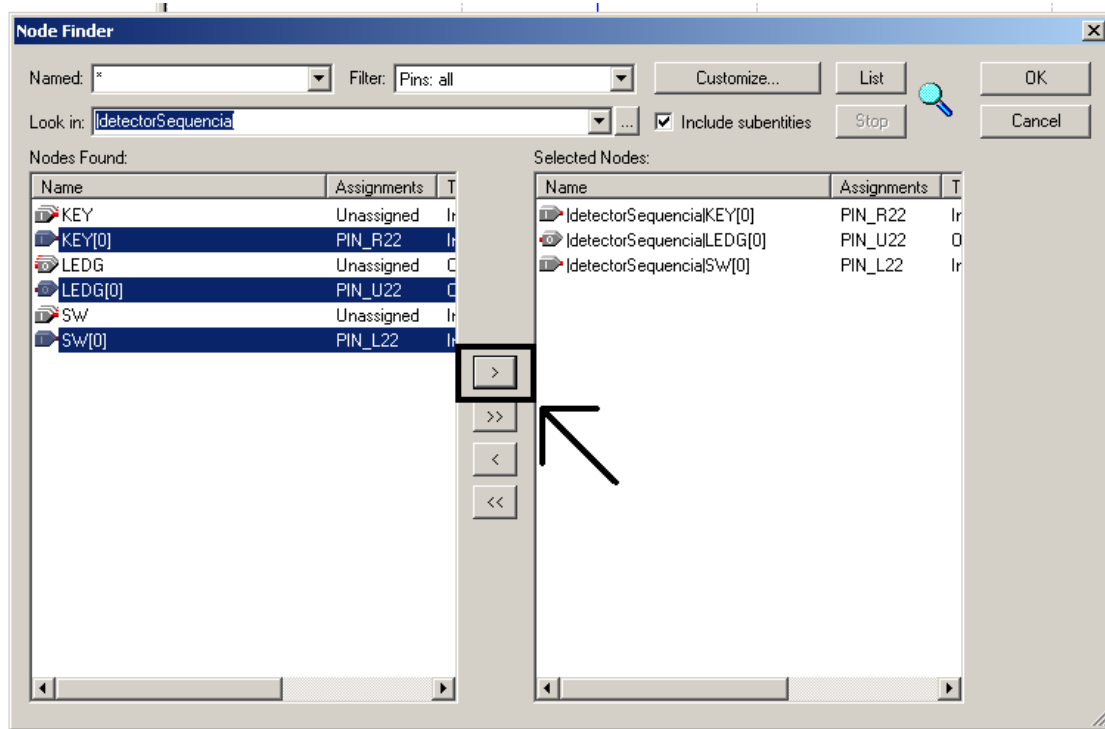


Figura 28: Caixa de seleção de sinais acessíveis.

É necessário observar que a versão do circuito empregada ao longo desta seção é aquele mostrado na Figura 19. Portanto, a janela apresentada anteriormente na Figura 26 conterá os pinos SW[0], KEY[0] e LEDG[0], assim como é apresentado na Figura 29. Edita-se as formas de onda da maneira que for conveniente ao projeto, ou seja, editar as formas de onda de modo que todas as possíveis situações (ou pelo menos, a maioria) de funcionamento do circuito sejam contempladas e que, além disso, a relação temporal entre os sinais obedeça as restrições temporais consideradas nas especificações do projeto. Para editar as formas de onda, é necessário utilizar os botões destacados na Figura 29 tal como o botão que está sinalizado. Esse botão permite a criação de sinais periódicos (sinais de relógio/sinais de clock). Por exemplo, o pino KEY[0]

foi selecionado, o botão em destaque foi pressionado e editou-se o sinal de `clock` de modo que o período do sinal fosse igual a 50 ns. Informações mais detalhadas podem ser encontradas no documento *Quartus II Classic Timing Analyzer*<sup>9</sup>.

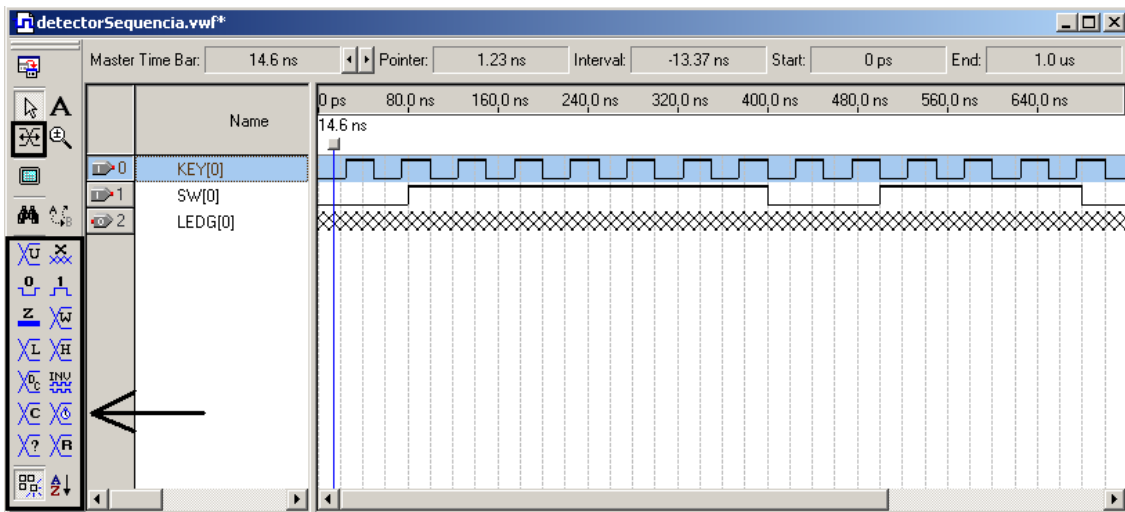


Figura 29: Elementos para edição da forma de onda de cada sinal.

Para a máquina de estados em questão, os sinais devem ser editados de forma que a sequência de 3 zeros seja detectada ( $LEDG[0] = 1$ ), comprovando o correto funcionamento do circuito. Após feitas todas as configurações nos sinais, deve-se salvar o arquivo. Por definição, o QUARTUS<sup>®</sup> II salva o arquivo com as formas de onda com a extensão `.vwf` e com o mesmo nome do *top-level* em estudo (no caso, `detectorSequencia`).

Para executar a simulação propriamente dita, deve-se fazer

`MAX+PLUS II > Simulator.`

Será aberta uma nova janela, onde é necessário indicar no campo *Simulation input* qual o arquivo de formas de onda será analisado (Figura 30). Isso porque em projetos maiores, cada arquivo de projeto pode ser construído e simulado separadamente com uma grande variedade de formas de onda. Especificamente para o exemplo do detector de sequência, o arquivo `detectorSequencia.vwf` deve ser adicionado clicando-se no botão destacado na Figura 30. Uma outra opção que deve ser marcada é aquela destacada em uma caixa. Essa opção permite que as formas de onda sejam atualizadas sempre que novas simulações forem geradas. Feito isso, basta então selecionar a opção *Timing* no campo *Simulation mode*, e executar a simulação propriamente dita. Para a máquina de estados em estudo, um comportamento possível é o apresentado na Figura 31. No ambiente QUARTUS<sup>®</sup> II é possível escolher o formato em que os valores dos sinais são apresentados. Para isso, selecione a linha do sinal no editor de formas de onda, aperte o botão direito do *mouse*, selecione o item *Properties* do *pop-up menu* que aparecerá, e finalmente selecione o formato desejado no campo *Radix*.

<sup>9</sup>Disponível em <http://www.altera.com/support/software/timing/sof-qts-timing.html>.

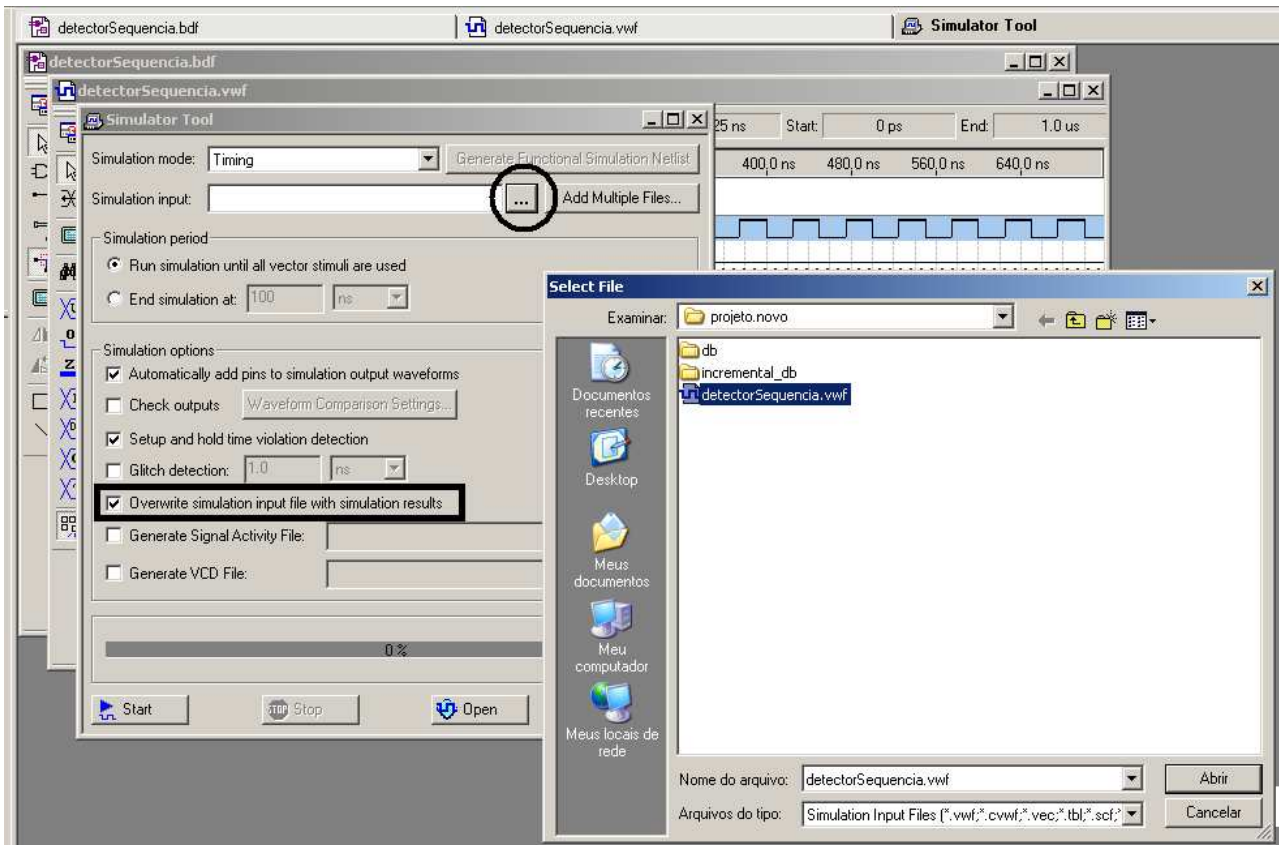


Figura 30: Configuração dos parâmetros para simulação.

A título de observação, para a execução do outro modo de simulação, a *Functional*, primeiramente deve-se clicar em **Generate Functional Simulation Netlist** antes de executar a simulação. Esse opção de simulação ignora os atrasos na resposta de cada componente. Em outras palavras, esse modelo de simulação intenciona a verificação da funcionalidade lógica do circuito. Contudo, durante procedimentos de depuração, o modo *Timing* é o mais recomendada já que leva em consideração as características de temporização (e atrasos) do circuito (Nicolato, 2002).

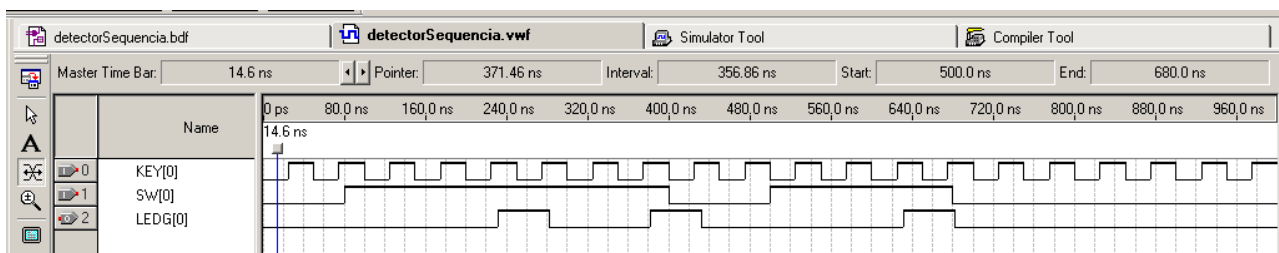


Figura 31: Formas de onda dos sinais que se resultaram de uma simulação.

Uma última observação vale ser realizada. Diante das ferramentas de análise e de simulação disponibilizadas pelo QUARTUS® II, pode-se levar em consideração que uma boa técnica para projetos consiste em, primeiramente, realizar uma simulação funcional para determinar o correto funcionamento do circuito, seguida de sua verificação temporal e, finalmente, verificar a sua funcionalidade completa testando-o no sistema físico (no caso, a placa DE1), junto com

outros dispositivos físicos (por exemplo, motores de corrente contínua a serem controlados pela placa DE1) e as exigências ambientais de aplicação (Nicolato, 2002).

De posse das respostas obtidas e apresentadas na Figura 31, pode-se comprovar que o circuito está funcionando como o esperado. Os *flip-flops* tipo D são atualizado a cada borda de subida do sinal de relógio (representado pelo pino de entrada KEY[0]), desde que o sinal  $x$  (representado pelo pino SW[0]) permaneça em nível lógico ALTO. Conclui-se então que o circuito está pronto para ser carregado no FPGA do *kit* DE1.

## 14 Carregando o Projeto

A fase final do projeto é fazer os “testes de campo”, carregando o projeto no dispositivo FPGA selecionado e testando-o com os dispositivos físicos reais. Ao longo do curso EA773 é utilizado o *kit* de desenvolvimento DE1 (Altera e Terasic, 2010). Essa placa possui diversos periféricos, tais como LEDs, chaves, botões, pinos de propósito geral e outros componentes mais complexos tais como memórias, interfaces de áudio e de vídeo bem como comunicação serial (RS232), conforme é ilustrado na Figura 32.

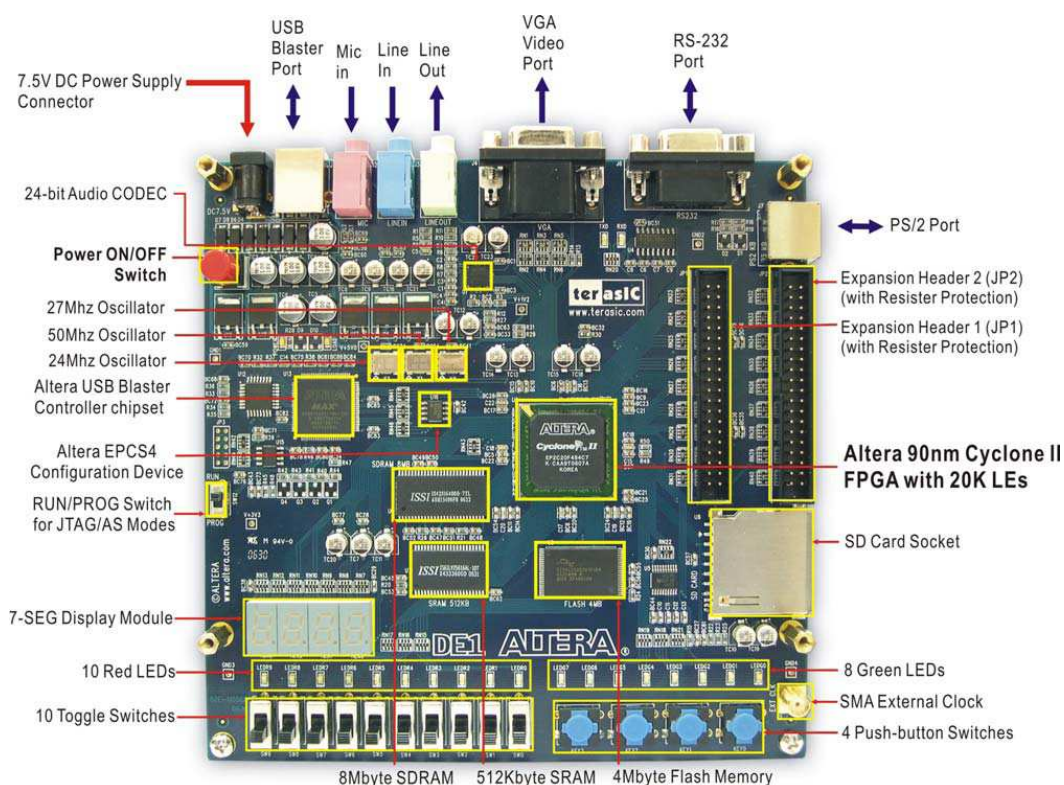


Figura 32: Placa de desenvolvimento DE1.

Como foge ao escopo deste texto, sugere-se uma leitura bastante atenta do seu manual, disponibilizado em (Altera e Terasic, 2010) e também na página do curso EA773. Deve-se prestar especial atenção, principalmente, como alguns periféricos são acionados. Por exemplo, os LEDs são ativados quando nível lógico alto é colocado no pino correspondente do FPGA



ao qual estão ligados. Já os botões são **ativo baixo**.

A transferência do arquivo binário gerado pelo compilador lógico para o FPGA é uma tarefa bastante simples mas que, como todas os outros procedimentos realizados até aqui, demanda de alguns cuidados. Primeiramente, o arquivo binário que será carregado no FPGA é o `projeto_novo.sof`. Esse arquivo é criado a partir do arquivo de maior hierarquia de um projeto, sendo definido em um dos primeiros passos do *Project Wizard* apresentado na Seção 2. Dessa maneira, deve-se abrir o arquivo `projeto_novo.bdf`, setá-lo como o de maior hierarquia, e inserir o componente criado na Seção 9, bem como pinos de entrada e de saída necessários, renomeá-los respeitando a padronização de nomenclatura apresentada na Seção 7.3, importar o *assignment* dos pinos e compilar o projeto. Nesse ponto, deve-se ter um circuito semelhante ao ilustrado na Figura 33.

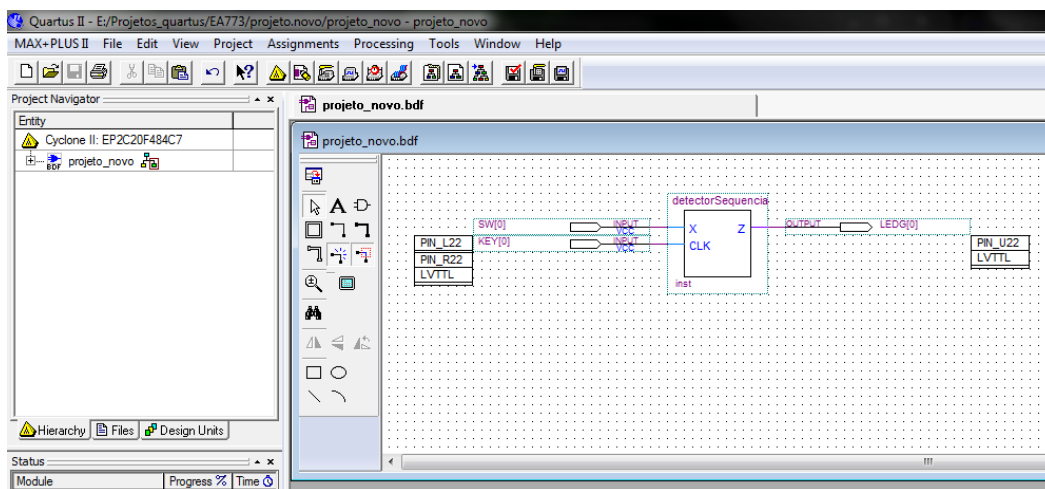
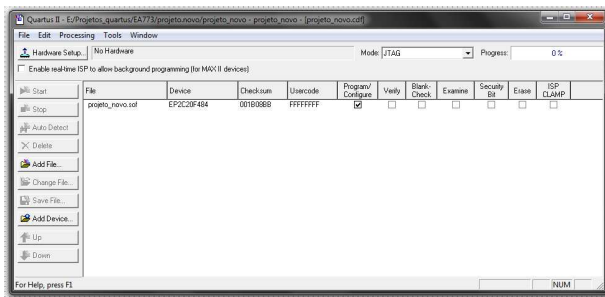


Figura 33: Circuito pronto para ser carregado no FPGA.

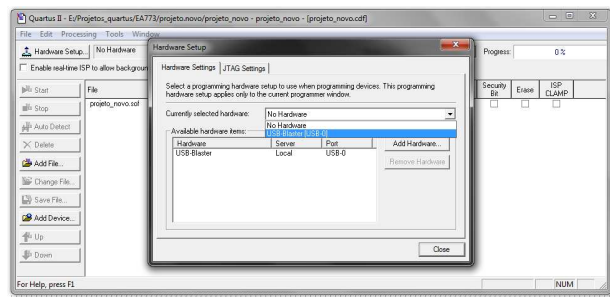
Em seguida, deve-se fazer

*MAX+PLUS II > Programmer.*

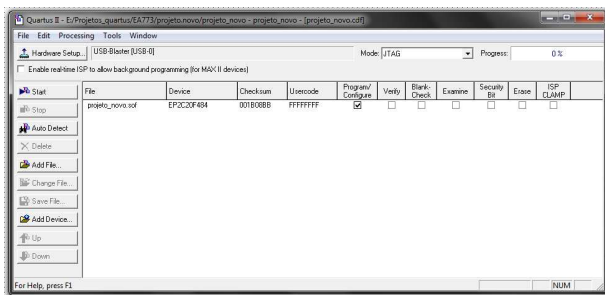
A janela que está ilustrada na Figura 34(a) aparecerá. Deve-se tomar o cuidado de que a opção *USB-Blaster* esteja habilitada, como na Figura 34(c). Caso não esteja, basta clicar em *Hardware Setup* e seguir os procedimentos ilustrados na Figura 34(b). Finalmente, basta acionar a gravação e, caso tenha sido bem sucedida, uma mensagem irá aparecer assim como é ilustrada na Figura 34(d). Em seguida, basta mudar a chave `SW[0]` para ALTO e pressionar o botão `KEY[0]` repetidas vezes. Quando a contagem for igual a 3, ou seja, quando a sequência de zeros for detectada, o `LEDG[0]` acenderá, indicando o sucesso e comprovando, na prática, o correto funcionamento da máquina de estados.



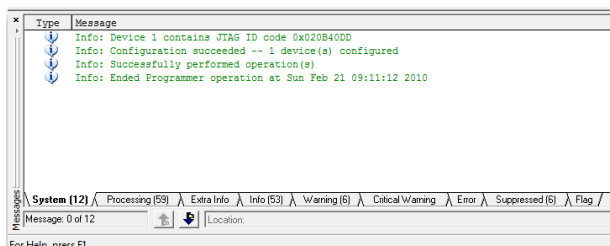
(a)



(b)



(c)



(d)

Figura 34: Procedimentos para gravação do circuito no FPGA: (a) tela inicial para a gravação do binário na placa; (b) janela para habilitar o USB-Blaster; (c) janela de gravação com o USB-Blaster habilitado; (d) caixa de mensagem da conclusão de gravação bem sucedida.

## 15 FAQs

*Esta seção contém as perguntas mais frequentes realizadas pelos alunos durante o curso quando a versão beta deste manual foi apresentada. Tais questões foram transcritas para essa nova versão e serão atualizadas conforme o andamento do curso neste e nos próximos semestres.*

### 15.1 Por que a onda de saída está deslocada para a direita em relação às entradas

Perceba que a saída  $z$  é modificada um pouco depois da mudança das entradas. Pensando de forma ideal essa mudança na saída deveria ocorrer no mesmo instante em que as entradas fosse modificadas. Deve-se atentar para o fato de que toda porta-lógica possui um tempo de propagação, um atraso. Essa característica representa o tempo necessário para que o sinal de entrada seja interpretado pelo circuito lógico e obtenha-se a resposta na saída. Os motivos pelos quais isso ocorre fogem do interesse desse tutorial, mas podem ser entendidos estudando-se a eletrônica interna de cada porta lógica, principalmente em relação a tecnologia dos transistores envolvidos. Além disso, os *datasheets* dos componentes usualmente trazem gráficos que representam esses tempos e os procedimentos que devem ser adotados para evitar maiores problemas.

## 15.2 O que são esses picos que ocorrem, às vezes, nas transições?

Tome o cuidado de ampliar o pico (“*zoom in*”). Repare no eixo do tempo para calcular a sua largura. Ele mede um pouco menos de 0.280 ns, ou seja, 280 ps (pico segundos). Isso ocorre, como explicado anteriormente, devido à eletrônica interna das portas. Esse tipo de fenômeno sempre ocorre nas transições. Isso é mais um motivo para evitar trabalhar com o circuito muito próximo do tempo de atraso. Quanto maior o tempo que o dado fica na saída, menos significativo é o ruído e menor a chance dele ser lido como um dado.

## 15.3 Como aumento o tempo máximo nas simulações?

Várias vezes foi citado que o tempo máximo não pode ser extrapolado, porém muitas vezes ele é pouco para as nossas necessidades. Surge então a necessidade de aumentá-lo. Para isso, abra o arquivo correspondente com as formas de onda a serem analisadas e faça *Edit > End Time* e escolha o tempo máximo da simulação o qual seja mais conveniente para o circuito em análise.

## 15.4 Não compila e aparece que o problema é a hierarquia.

Primeiro tome o cuidado de que não existem dois projetos na mesma pasta. Em seguida, mude o arquivo principal na hierarquia, selecionando outra folha como folha principal. Um procedimento bastante eficiente é aquele discutido anteriormente na Seção 10, onde é explicado o procedimento para a compilação: abra na janela o arquivo que será usado como principal (no exemplo o Relógio COM Despertador) e, em seguida, faça *Project > Set as Top-Level Entity*.

## 15.5 O nome do meu projeto é Projetão da Moçada e não compila. Por quê?

Evite espaços e caracteres latinos (tais como “ç” e “ã”) tanto no nome das pastas onde os projetos serão gravados quanto no nome dos arquivos e pinos de entrada e de saída. Existem programas que aceitam, outros não. A título de observação, o QUARTUS® II aceita letras maiúsculas.

## 15.6 Tenho 200 entradas que ficam em 1 sempre, mais 200 que sempre ficam em 0. Quando coloco no *Waveform* fica uma banguça. Já que eu não tenho que mudá-las sempre, tem algo mais prático a ser feito?

Abra a janela de seleção de componentes (duplo clique na área de trabalho), digite VCC ou GND. No primeiro caso, o bloco correspondente representa nível lógico ALTO e, no segundo, nível lógico BAIXO. Esses blocos são interessantes de serem usados quando um determinado sinal de algum bloco deve ser desabilitado permanentemente. Por exemplo, um sinal CLRN representa uma porta de CLEAR que é acionada em nível BAIXO. Ligá-la ao VCC garante que ela nunca será acionada.

## 15.7 Fui no *TTL Handbook* e justamente a página que eu queria não estava lá. O que faço?

Acesse o site <http://www.alldatasheet.com/> e procure pelo *datasheet* do componente que você quer estudar. Evite imprimir. Salve em um *pendrive* e leve com você para o laboratório. Copie no computador e use quando for necessário.

## 15.8 Sumiram as ferramentas do *Waveform*.

Faça *Tools > Customize Waveform Editor*.

## 15.9 No simulador, os pinos não aparecem.

Compile o arquivo e, em seguida, insira os pinos. Use o procedimento apresentado na Seção 13.

## 15.10 Simulei anteriormente, inseri e tirei alguns pinos, quero simular novamente. Compile e quando abro o simulador só tenho pinos antigos. Que fazer?

Apague os pinos que você apagou no esquemático e insira os novos. Analise o procedimento apresentado Seção 13.

## 15.11 Sumiram as ferramentas da área de trabalho.

Faça *Tools > Customize Block Editor*.

### **15.12 Estou na tela de associação de pinos com endereços, para gravação. O problema é que minha tela não mostra, na parte de baixo, os pinos do meu projeto.**

Compile o projeto e abra a tela de associação novamente. Caso o local onde os pinos deveriam aparecer simplesmente não está lá, ainda nessa tela siga o caminho: *View > All Pins List*. Como dito na Seção 7 é muito mais simples se a padronização de nomenclatura dos pinos apresentada no manual da placa seja mantida. Assim, pode-se importar o *assignment* dos pinos.

### **15.13 Estou na tela de associação de pinos com endereços, para gravação. O problema é que minha tela não mostra, ao lado dos pinos do meu projeto, a coluna *Location*.**

Clique com o botão direito nessa “tabela” e vá para a opção *Customize Columns*. Insira *Location* na sua tabela. Novamente, como dito na Seção 7 é muito mais simples se a padronização de nomenclatura dos pinos apresentada no manual da placa seja mantida. Assim, pode-se importar o *assignment* dos pinos.

### **15.14 Estou na tela de gravação mas não consigo gravar.**

Releia a Seção 14 e analise com atenção a Figura34.

### **15.15 Usei o *clock* da placa e todos os meus LEDs ficam acesos.**

As frequências dos osciladores existentes na placa DE1 e disponíveis aos usuários são de 50 MHz e 27 MHz. O olho humano não identifica oscilações maiores do que 20 ou 30 Hz. O ideal é dividir a frequência dos osciladores utilizando o componente da biblioteca padrão do QUARTUS® II denominado *freqdiv*. Em algumas situações, pode-se construir um divisor de frequência com *flip-flops* tipo T ou contadores binários.

Aconselha-se que a simulação temporal (análise das formas de onda) sempre sejam geradas. Um divisor de frequência mal projetado pode invalidar o correto funcionamento do circuito.

### **15.16 Por que ao renomear um pino de entrada com o nome *CLOCK\_27*, o sistema não o associa ao pino do relógio de 27 MHz do *kit*?**

O *kit* DE1 possui dois relógios de 27 MHz. Eles são referenciados pelos nomes *CLOCK\_27[0]* e *CLOCK\_27[1]*.

## Referências

- Altera (2010a), ‘Altera Corporation’, Página principal: <http://www.altera.com>.
- Altera (2010b), ‘Altera Corporation - Cyclone II FPGAs at Cost That Rivals ASICs’, Página principal de documentação: <http://www.altera.com/products/devices/cyclone2/cy2-index.jsp>.
- Altera (2010c), ‘Altera Corporation - Debugging with the SignalTap II Logic Analyzer’, Página de documentação: [http://www.altera.com/literature/hb/qts/qts\\_qii53009.pdf?GSA\\_pos=6&WT.oss\\_r=1&WT.oss=SignalTap%20II](http://www.altera.com/literature/hb/qts/qts_qii53009.pdf?GSA_pos=6&WT.oss_r=1&WT.oss=SignalTap%20II).
- Altera (2010d), ‘Altera Corporation - Intellectual Property & Reference Designs’, Página central de documentação: <http://www.altera.com/products/ip/ipm-index.html>.
- Altera (2010e), ‘Altera Corporation - Nios II Embedded Design Suite Support’, Página central de documentação: [http://www.altera.com/support/ip/processors/nios2/ips-nios2\\_support.html](http://www.altera.com/support/ip/processors/nios2/ips-nios2_support.html).
- Altera (2010f), ‘Altera Corporation - Quartus II Design Software’, Página de documentação: <http://www.altera.com/support/software/sof-quartus.html>.
- Altera e Terasic (2010), ‘Altera Corporation and Terasic Technologies’, Página principal de documentação para o kit DE1: <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=83&PartNo=4>.
- der Spiegel, J. V. (2010), ‘VHDL Tutorial’, Página eletrônica: [http://www.seas.upenn.edu/~ese201/vhdl/vhdl\\_primer.html](http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html). Department of Electrical and Systems Engineering, University of Pennsylvania.
- IEEE-VHDL (2009), ‘IEEE Standard VHDL Language Reference Manual’, *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)* pp. c1 – 626.
- Nicolato, F. (2002), Arquitetura de Hardware para a Extração em Tempo Real de Características de Múltiplos Objetos em Imagens de Vídeo: Classificação de Cores e Localização de Centróides, Dissertação de Mestrado, FEEC - UNICAMP.
- Pedroni, V. A. (2008), *Digital Electronics and Design with VHDL*, Elsevier - Morgan Kaufmann Publishers.
- Perry, D. (2002), *VHDL: Programming by Example*, 4a. edição, McGraw-Hill Professional.
- Tocci, R. J., Widmer, N. S. e Moss, G. L. (2008), *Sistemas Digitais: Princípios e Aplicações*, Person - Prentice Hall.

Wu, S.-T. (2001), Notas de aula do curso “EA772 - Circuitos Lógicos”. Disponível em <http://www.dca.fee.unicamp.br/courses/EA772/2s2001/>., Relatório Técnico, Faculdade de engenharia Elétrica e de Computação - UNICAMP.