unesp UNIVERSIDADE ESTADUAL PAULISTA

Campus de Guaratinguetá

Colégio Técnico Industrial de Guaratinguetá "Professor Carlos Augusto Patrício Amorim"

Dispositivos Lógicos Programáveis

Autores

Caio Augusto de Oliveira

Jéssica Azevedo de Aguiar

Mateus Galvão Said Fontanini

Revisão

Prof. Marcelo Wendling

Sumário

1. Introdução

A crescente evolução do mercado de circuitos eletrônicos é visível. Um exemplo para esta afirmação é o mercado mundial, influenciado drasticamente pela produção de componentes eletrônicos nos países chamados de tigres asiáticos e no vale do silício. Pode-se citar alguns pontos na história da eletrônica que foram marcantes para sua evolução: a invenção da válvula, a criação dos transistores, a evolução para circuitos integrados e a construção de microprocessadores. Porém, muitos autores citam que a verdadeira virada no mercado eletrônico do século XX foi a criação dos circuitos de aplicação específica e o aperfeiçoamento do hardware reconfigurável dos Dispositivos Lógicos Reprogramáveis (Programmable Logic Device - PLD).

Para muitos projetos de hardware, o custo da prototipação impedia a construção de sistemas confiáveis, fazendo com que máquinas fossem desenvolvidas e distribuídas com alguns defeitos, sendo estes descobertos de forma drástica por usuários, resultando em um aumento no custo de utilização. A promessa da computação reconfigurável é um "hardware virtual", podendo emular vários sistemas complexos de forma transparente ao usuário.

O potencial de crescimento de atividades relacionadas ao PLD num futuro próximo é muito grande. A automatização dos processos industriais é cada vez mais intensa, e ao longo dos próximos anos, simplificará e acelerará ainda mais todo o ciclo de desenvolvimento de projetos, em velocidade impressionante.

2. Lógica Programável

Os circuitos integrados digitais implementados em pastilha de silício podem ser classificados como circuitos digitais padrão ou circuitos digitais de aplicações específicas ASICs (*Aplication Specific Integrated Circuits*).

Os circuitos padrões são constituídos por portas lógicas (*AND,OR,NOT* e *Flip – Flops*) e necessitam de vários componentes externos para a realização de uma função específica. Os circuitos integrados ASICs são aqueles que precisam de um processo de fabricação especial, que requer máscaras específicas para cada projeto. Outra característica dos circuitos integrados ASICs é o tempo de desenvolvimento longo e os custos extremamente altos. Geralmente não necessitam de muitos componentes externos para a realização de uma função específica, pois sua alta densidade os torna aptos para a implementação de vários tipos de aplicação. Em ambos os casos, os circuitos integrados digitais possuem suas funções internas predefinidas, implementadas na sua construção no processo de fabricação.

O desenvolvimento de projeto de circuitos digitais tem evoluído rapidamente nas últimas décadas. A utilização da ferramenta de *software* denominada EDA (*Eletronic Design Automation*) e o aperfeiçoamento dos PLDs (*Programmable Logic Devices*) têm simplificado e acelerado todo o ciclo de projeto.

Os PLD são circuitos integrados que podem ser configurados pelo próprio usuário. Não apresentam uma função lógica definida, até que sejam configurados. Possuem, como principal característica, a capacidade de programação das funções lógicas pelo usuário, eliminando-a do processo de fabricação do circuito integrado, o que facilita, assim, as prováveis mudanças de projeto. Em comparação com outras tecnologias de circuitos integrados digitais, os dispositivos de lógica programável apresentam um ciclo de projeto menor e custos reduzidos.

Os controladores lógicos programáveis disponíveis no mercado brasileiro utilizam na sua arquitetura microcontroladores e circuitos integrados de aplicações específicas ASICs.

Esses microcontroladores, para executar seus programas de controle, necessitam realizar ciclos de busca e execução de instrução. O ciclo de busca da instrução não está diretamente relacionado com o processo no qual o controlador lógico programável está

inserido, mas é a condição determinante para o microcontrolador executar o programa que está carregado na memória.

Essa necessidade de busca da instrução demanda tempo do microcomputador, que poderia estar sendo utilizado na execução de tarefas pertinentes ao processo.

Os microcontroladores são componentes extremamente flexíveis devido a sua programabilidade. Essa programação permite sua aplicação em diversos tipos de controles industriais.

A execução de um algoritmo de controle depende de seu software, armazenado em memória, que será executado numa arquitetura baseada em microprocessador, com ciclo de buscas e execução das instruções.

Numa arquitetura baseada em dispositivo lógico programável, por exemplo, FPGA (*Field Programmable Gate Array*),um algoritmo de controle é implementado por hardware, sem a necessidade de ciclos de busca e execução de instruções.

O problema básico a ser resolvido é a implementação de uma arquitetura eficiente, baseada em lógica programável estruturada, para execução desse algoritmo de controle, em vez de compilá-lo para sua execução em microprocessador.

A tarefa que faz a tradução de um algoritmo para uma arquitetura de hardware eficiente é denominada síntese. A síntese cria uma arquitetura com células lógicas que executam as operações do algoritmo, sem a necessidade de se gerar e decodificar instruções.

Uma das grandes vantagens da utilização de dispositivos lógicos programáveis nessa nova arquitetura proposta é a possibilidade de se definir vários blocos de hardware, que operam em paralelo, aumentando muito a capacidade computacional do sistema.

Os dispositivos lógicos programáveis (PLDs) foram os dispositivos eletrônicos que possibilitaram a implementação de Lógica Programável Estruturada. Os PLDs podem ser classificados em função de portas lógicas que comportam, como descrito a seguir:

(1) SPLDs (*Simple Programmable Logic Devices*): São dispositivos simples e de baixa capacidade, que tipicamente contêm menos de 600 portas lógicas, fabricados com tecnologia CMOS.

(2) HCPLDs (*High Complex Programmable Logic Devices*): São dispositivos de alta capacidade, que tipicamente contêm mais de 600 portas lógicas; os mais modernos podem atingir cerca de 250.000 portas e englobam os dispositivos CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Arrays*), todos fabricados com tecnologia CMOS.

SPLDs

Os PLAs (*Programmables Logic Arrays*) foram os primeiros Dispositivos Lógicos Programáveis Simples (SPLDS) criados especificamente para a implementação de circuitos lógicos. Introduzidos pela empresa Philips no início de 1970, esses dispositivos consistem em dois níveis de portas lógicas: um plano de portas *AND* seguido por um plano de portas *OR*, ambos programáveis, como mostra a figura 1.

Um PLA é estruturado de forma que cada saída do plano *AND* pode corresponder a qualquer produto das entradas. Da mesma forma, cada saída do plano *OR* pode ser configurada para produzir a soma lógica de quaisquer saídas do plano *AND*.

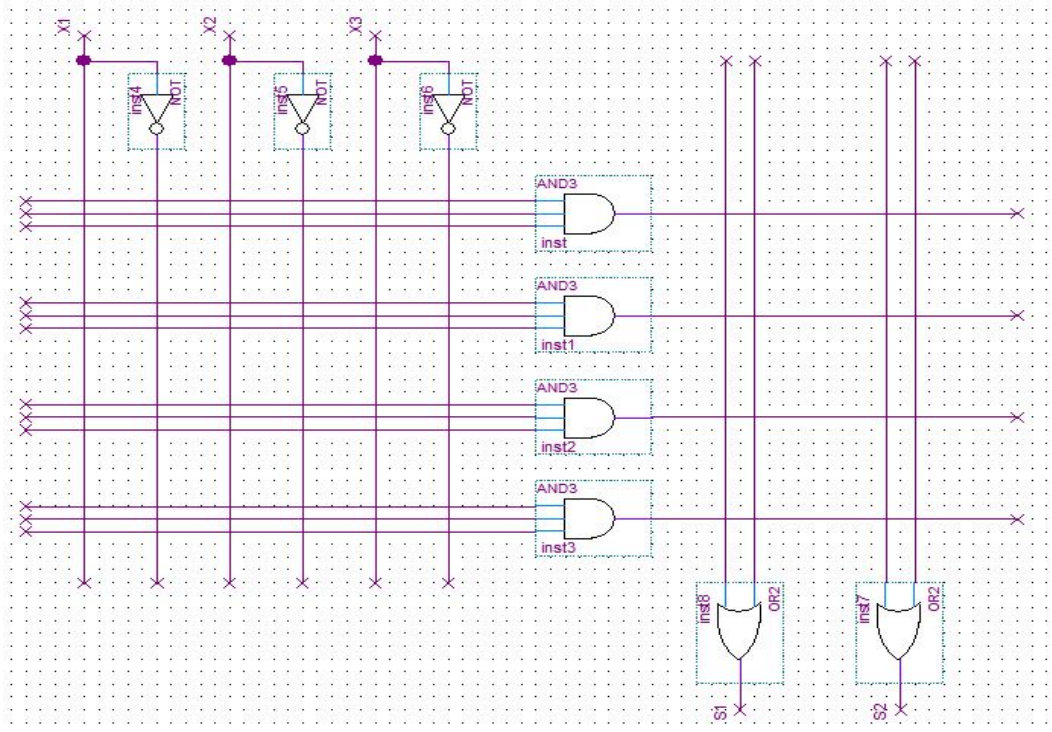


Figura 1 Estrutura simplificado de um PLA.

Em virtude da estrutura montada, os PLA são adequados para as implementações de funções lógicas na forma de produtos de soma, e eles se apresentam muito versáteis, pois tanto os termos *AND* como os termos *OR* podem possuir muitas entradas.

Entretanto, essa tecnologia também apresenta problemas como alto custo de fabricação e baixo desempenho em termos de velocidade. Essas desvantagens existem devido aos dois níveis de lógica configurável. Os planos lógicos programáveis são difíceis de serem fabricados e introduzem atrasos significativos de propagação dos sinais elétricos.

A tecnologia PAL (*Programmable Array Logic*) foi então desenvolvida para super deficiências apresentadas pela tecnologia PLA. Os PALs apresentam um único plano *AND* configurável, figura 2, custo menor e melhor desempenho.

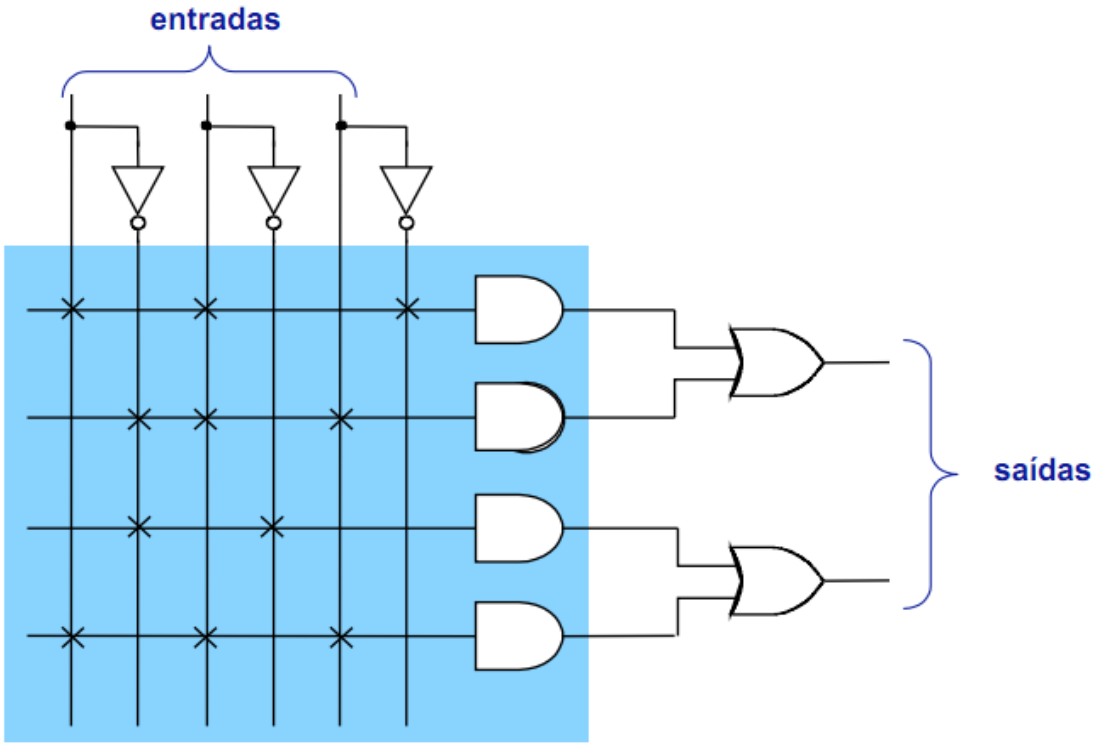


Figura 2 Estrutura simplificado de um PAL.

HCPLDs

Quanto maior o número de portas de um PLD, maior será a sua complexidade. Os Dispositivos Lógicos de Alta Complexidade (HCPLD) dividem-se basicamente em dois grupos: **CPLD e FPGA.**

A diferença básica entre os dois dispositivos está na estrutura interna de suas células lógicas e na metodologia da interligação dessas células. De forma geral, internamente, os HCPLDs podem ser vistos como dispositivos que integram na sua estrutura centenas de macrocélulas programáveis, que são interligadas por conexões também programáveis.

Os Dispositivos Lógicos Programáveis Complexos (CPLDs) foram introduzidos no mercado internacional pela empresa Altera Corp. em 1983, inicialmente como Dispositivos Lógicos Programáveis Apagáveis (EPLDs) e, posteriormente, como CPLDs.

Os CPLDs são dispositivos programáveis e reprogramáveis pelo usuário, com alto desempenho, baixo custo por função e alta capacidade de integração. Um CPLD pode ser aplicado, por exemplo, como uma máquina de estado ou decodificador de sinais, substituindo centenas de circuitos discretos que implementariam a mesma função.

Os CPLDs implementam capacidade lógica de até 50 dispositivos PLDs típicos. Suas principais vantagens em relação aos circuitos discretos ASICs tradicionais são:

(a) Programabilidade e Reprogramabilidade: Permite que funções lógicas possam ser alteradas, simplificando o desenvolvimento de protótipos.

(b) Tecnologia CMOS: Menor Consumo de energia elétrica.

(c) Integração em larga escala: Redução de tamanho da placa de circuito impresso, pois possibilita a eliminação de diversos componentes discretos.

(d) Simplificação e redução do tempo de desenvolvimento: Simplifica e reduz o tempo de desenvolvimento da placa de circuito impresso, pois permite que o

projetista defina os sinais elétricos conforme desejado: entradas ou saídas podem ocupar o mesmo terminal do dispositivo.

(e) Teste e depuração: As linguagens utilizadas na programação do dispositivo permitem a simulação, teste e depuração rápida do protótipo.

2.1. Tecnologia FPGA

Os dipositivos ASICs, SPLDs e CPLDs, descritos nas seções anteriores, permitem a implementação de uma grande variedade de circuitos lógicos. Contudo, com exceção dos CPLDs, aqueles componentes possuem baixa capacidade lógica e são viáveis apenas para aplicações relativamente pequenas.

Até mesmo para os CPLDs, apenas circuitos moderadamente grandes podem ser acomodados em um único circuito integrado. Para se implementar circuitos lógicos maiores, é conveniente utilizar-se de outro tipo de dispositivo HCPLD que possua capacidade lógica maior.

O FPGA é um HCPLD que suporta a implementação de circuitos lógicos relativamente grandes. Consiste em um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado. Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas. O primeiro FPGA disponível comercialmente foi desenvolvido pela empresa Xilinx Inc., em 1983.

Os FPGAs não possuem planos *OR* ou *AND*, consistem em um grande arranjo de células configuráveis que podem ser utilizadas para a implementação de funções lógicas. Basicamente é constituída por blocos lógicos, blocos de entrada e saída, e chaves de interconexão. Os blocos lógicos formam uma matriz bidimensional, e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas de blocos lógicos. Esses canais de roteamento possuem chaves de interligação programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projeto. A figura 3 mostra a estrutura básica de um FPGA.

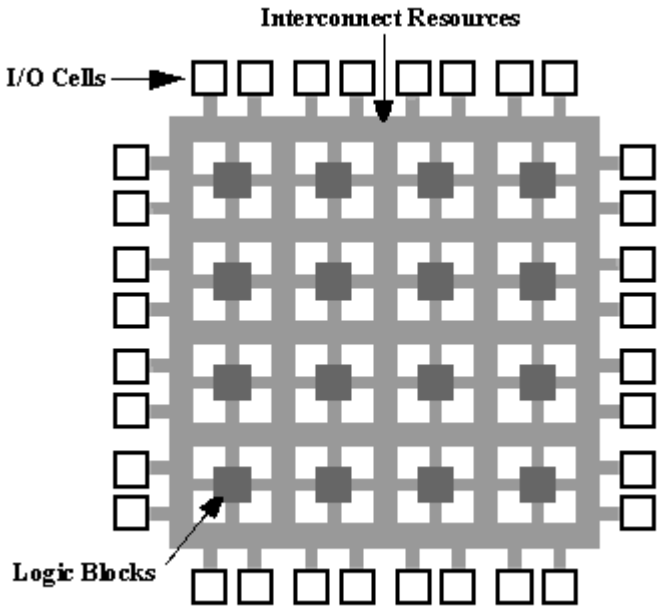


Figura 3 Estrutura simplificada de um FPGA.

2.2. Blocos Lógicos de FPGA

No interior de cada bloco lógico de FPGA existem vários modos possíveis para implementação de funções lógicas. O mais utilizado pelos fabricantes de FPGA como, por exemplo, a empresa Altera Corp., é o bloco de memória LUT (*Look – Up Table*).

Esse tipo de bloco lógico contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas. Cada célula é capaz de armazenar um único valor lógico, zero (0) ou um (1).

Nos FPGAs disponíveis comercialmente como, por exemplo, da empresa Altera Corp., os blocos lógicos LUTs possuem geralmente quatro ou cinco entradas, o que permite endereçar 16 ou 32 células de armazenamento.

Quando um circuito lógico é sintetizado em um FPGA, os blocos lógicos são programados para realizar as funções necessárias, e os canais de roteamento são estruturados de forma a realizar a interconexão necessária entre os blocos lógicos.

As células de armazenamento dos LUTs de um FPGA são voláteis, o que implica perda do conteúdo armazenado, no caso de falta de suprimento de energia elétrica. Dessa forma, o FPGA dever se programado toda vez que for energizado. Geralmente utiliza-se uma pequena memória FLASH ou EEPROM (*Electrically Erasable*

Programmable Read Only Memory), cuja função é carregar automaticamente as células de armazenamento, toda vez que o FPGA for energizado.

2.3. Granularidade

Granularidade é uma característica dos FPGAs relacionada com o grão. O grão é a menor unidade configurável que compõe um FPGA. A fim de classificar os FPGAs quanto ao bloco lógico, foram criadas algumas categorias.

(a) Grão Grande: Os FPGAs dessa categoria podem possuir como grão unidades lógicas aritméticas, pequenos microprocessadores e memórias.

(b) Grão Médio: Os FPGAs de grão médio frequentemente contêm dois ou mais LUTs e dois ou mais flip-flops. A maioria das arquiteturas de FPGAs implementam a lógica em LUTs de quatro entradas.

(c) Grão Pequeno: Os FPGAs de grão pequeno contêm um grande número de blocos lógicos simples. Os blocos lógicos normalmente contêm uma função lógica de duas entradas ou multiplexadores 4x1 e um flip-flop.

2.4. Arquitetura geral de roteamento

A arquitetura de roteamento de um FPGA é a forma pela qual os seus barramentos e as chaves de comutação são posicionadas para permitir a interconexão entre as células lógicas.

Essa arquitetura deve permitir que se obtenha um roteamento completo e, ao mesmo tempo, alta densidade de portas lógicas. Para melhorar a compreensão dessa arquitetura é necessária a definição de alguns conceitos básicos, sendo que parte deles é exemplificada na figura 4.

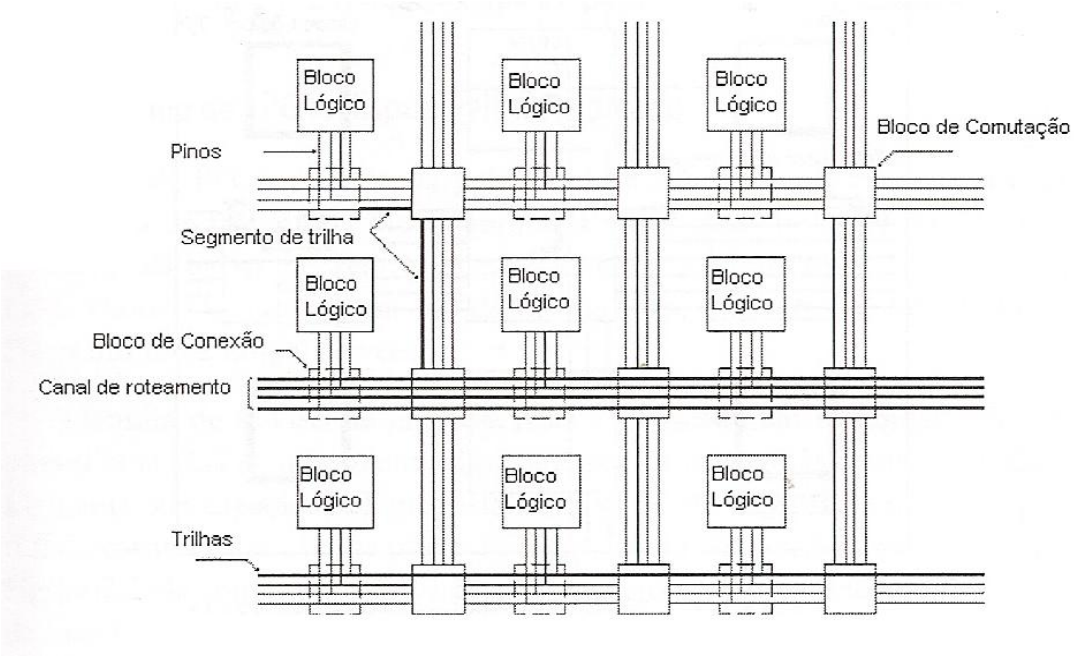


Figura 4 Arquitetura básica de roteamento de um FPGA.

(a) Pinos: Entrada e Saída de blocos lógicos.

(b) Conexão: Ligação elétrica de um par de pinos

(c) Rede: Conjunto de pinos que estão conectados.

(d) Bloco de Comutação: Utilizado para conectar os segmentos da trilha.

(e) Segmentos da Trilha: Segmento não interrompido por chaves programáveis.

(f) Canal de Roteamento: Grupo de 2 ou mais 3 trilhas paralelas.

(g) Bloco de Conexão: Permite a conectividade das entradas e saídas de um bloco lógico com os segmentos de trilhas nos canais.

2.5. Técnicas de Configuração

As chaves programáveis de roteamento apresentam algumas propriedades, tais como tamanho, resistência, capacitância e tecnologia de fabricação, que afetam principalmente a velocidade e o tempo de propagação dos sinais e definem características como volatilidade e capacidade de reprogramação.

Na escolha de um dispositivo reconfigurável, esses fatores devem ser avaliados. Basicamente existem três tipos de tecnologia de programação das chaves de roteamento.

(a) SRAM (*Static Random Access Memory*): Nessa tecnologia, a chave de roteamento ou comutador é um transistor de passagem ou um multiplexador controlado por uma memória estática de acesso randômico SRAM. A figura 5 ilustra essa tecnologia de programação, na qual uma célula SRAM é utilizada para controlar a porta (*gate*) do transistor de passagem. Devido a volatilidade dessas memórias, os FPGAs que se utilizam dessa tecnologia precisam de uma memória externa tipo FLASH ou EEPROM. Essa tecnologia ocupa muito espaço no circuito integrado, porém é rapidamente reprogramável.

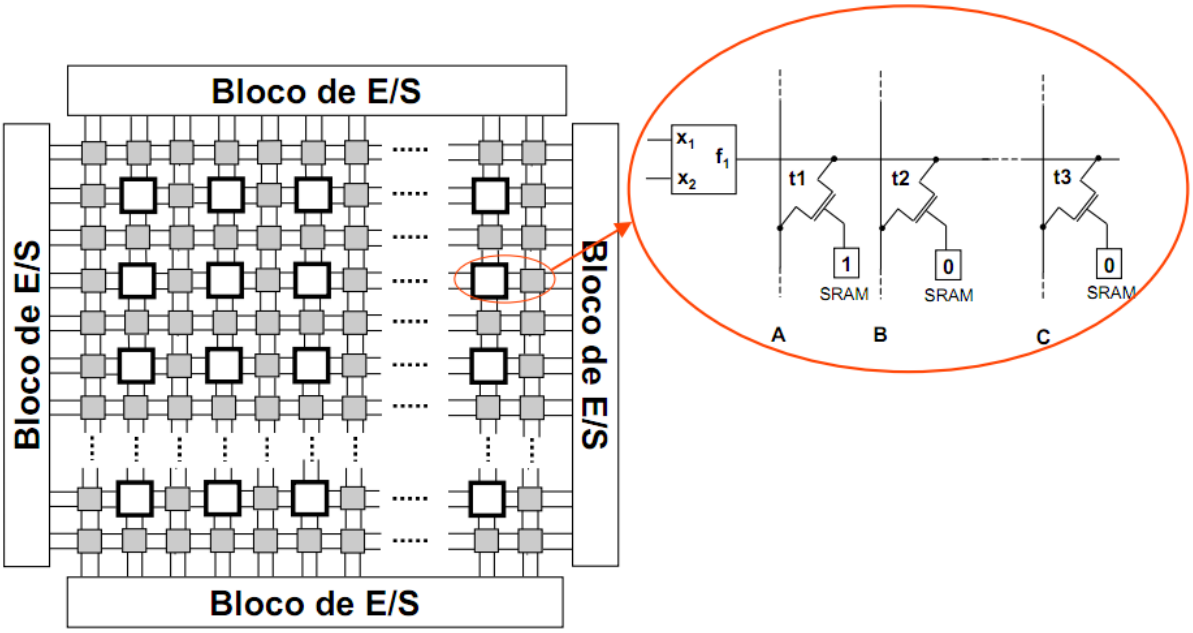


Figura 5 Utilização da tecnologia SRAM para controle de transistor de passagem.

(b) *Antifuse*: Essa tecnologia baseia-se num dispositivo de dois terminais, que no estado não programado apresenta uma alta impedância (circuito aberto). Aplicando-se a uma tensão, por exemplo, entre 11 e 20 V_{dc}, o dispositivo forma um caminho de baixa impedância entre seus terminais.

(c) *Gate* Flutuante: A tecnologia *Gate* flutuante baseia-se em transistores MOS (*Metal Oxide Semicondutor*), especialmente construídos com dois *Gates* flutuantes semelhantes ao usados nas memórias EPROM (*Erasable Programmable Read Only Memory*) e EEPROM (*Electrical EPROM*). A maior vantagem dessa tecnologia é sua capacidade de programação e a retenção dos dados. Além disso, da mesma forma que

uma memória EEPROM, os dados podem ser programados com o circuito integrado instalado na placa, característica denominada ISP (*In System Programmability*).

2.6. Família de FPGAs disponíveis no mercado

A família de FPGAs, fabricada pela empresa Altera Corp., consiste em uma hierarquia de três níveis muito similar à encontrada nos CPLDs. Contudo, o nível mais baixo da hierarquia consiste em um conjunto de células lógicas como nos SPLDs (*Single Programmable Logic Devices*).

A família de FPGAs da empresa Altera é baseada em tecnologia SRAM e possui uma LUT de quatro entradas como seu elemento lógico básico LE (*Logic Element*). Sua capacidade lógica está na faixa de 576 a 68.416 elementos lógicos (LEs). A arquitetura interna básica da família Altera, figura 6, contém três tipos de células lógica: Elemento lógico(LE), bloco de matriz lógica (LAB) e bloco de memória embutido (EAB).

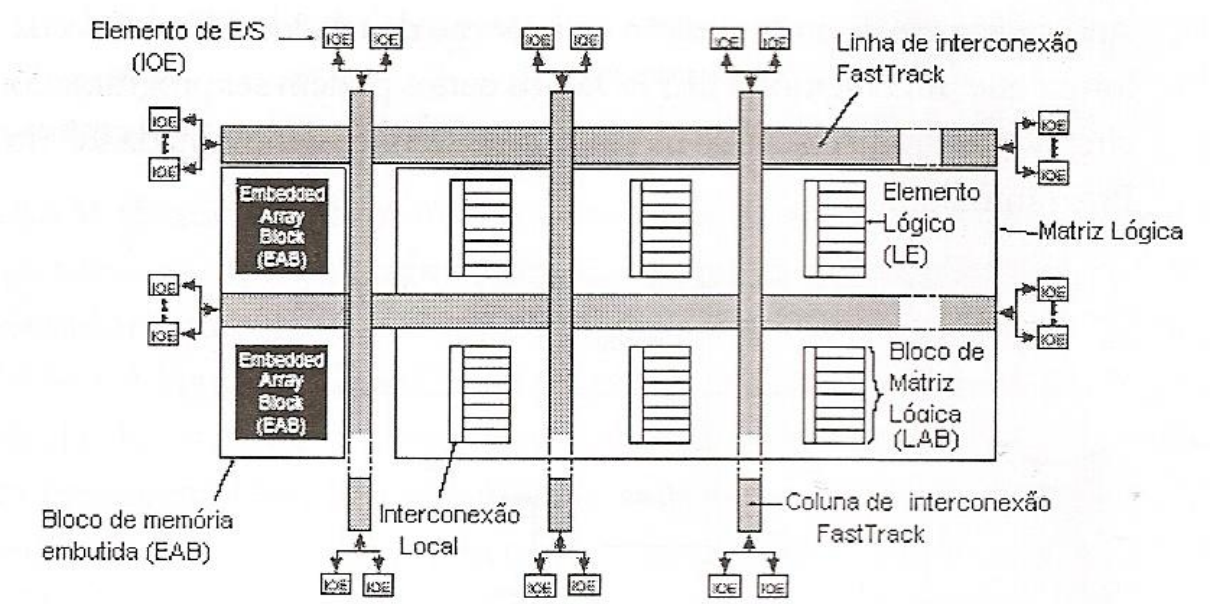


Figura 6 Arquitetura interna básica dos FPGA Altera.

Os blocos de memória embutidos EABs (*Embedded Array Block*), constituem-se no elemento principal da arquitetura básica. Cada bloco EAB pode conter, por exemplo, 2.048 bits programáveis que podem ser configurados como RAM, FIFO (*First-In First-Out*), etc. Pode ser configurado para atuar como um bloco de memória RAM com

tamanhos variáveis entre: 256x8, 512x4, 1k x2 ou 2kx1. Além disso, um bloco EAB pode ser configurado para implementar um circuito lógico complexo, como um multiplexador, microcontrolador, máquina de estado ou uma função DSP (*Digital Signal Processing*).

Na arquitetura interna básica apresentada, o bloco lógico chamado Elemento Lógico LE, pode conter uma LUT de quatro entradas, um *flip-flop* e circuitos de *carry* de finalidade especial para circuitos aritméticos. O LE também inclui circuito em cascata que permitem a implementação eficiente de funções *AND* de várias entradas. Os LE são agrupados em dez, formando um conjunto chamado Bloco de Matriz Lógica, LAB (*Logic Array Block*).

Cada LAB contém interconexão local, e cada trilha local pode conectar qualquer LE a outro LE na mesma LAB. A interconexão local também pode ser ligada à interconexão global, chamada *FastTrack*. Cada trilha *FastTrack* entende-se pela altura ou largura completa do dispositivo. Isso facilita a configuração automática feita pelo *software* EDA.

Todas as *FastTrack* horizontais são idênticas, portanto os atrasos de interconexões são mais previsíveis que em outros FPGAs que empregram segmentos menores, pois há menos chaves comutadoras programáveis em caminhos longos.

Os pinos de entrada e saída (E/S) são alimentados individualmente por um elemento chamado IOE (*Input Output Element*) localizado no final de uma linha ou coluna de interconexão global. Os elementos IOE contém *buffers* bidirecionados e *flip-flops* que podem ser configurados como registradores de entrada ou saída.

3. Desenvolvimento de projetos utilizando FPGA

O processo de projeto com FPGA envolve várias etapas que geralmente são automatizadas. Atualmente, a utilização de ferramentas de *software* EDA tem simplificado e acelerado todo o ciclo de projeto. Um sistema típico de desenvolvimento de projetos, com ferramentas de *softwares* EDA, consiste em vários programas interconectados. Esse processo envolve as seguintes etapas:

(1) Especificação e entrada do projeto
(2) Síntese e mapeamento da tecnologia
(3) Posicionamento e roteamento
(4) Verificação e teste
(5) Programação do FPGA

3.1. Especificação e entrada de projetos

A entrada de projetos pode ser realizada de duas formas: um diagrama lógico, desenvolvido a partir de um editor gráfico, no qual é possível utilizar portas lógicas e macroinstruções, ou por meio de um editor de texto que utilize uma linguagem de descrição de hardware HDL (*Hardware Description Language*).

A especificação do projeto é apresentada em termos abstratos ou em métodos formais, seguida pela análise da viabilidade da implementação por meio de simulação de alto nível. Nessa fase é importante que a linguagem utilizada seja o mais próximo possível da linguagem humana.

As ferramentas de captura de diagramas lógicos ou editores gráficos permitem que o projetista especifique o circuito como um diagrama lógico em 2D (duas dimensões), conectando componentes lógicos com recursos de roteamento. O *software* utilizado para a edição de projetos utilizando FPGAs da Altera Corp. e o Quartus®II.

Os componentes lógicos estão contidos em uma biblioteca de macroinstruções fornecidas pelo *software* ou que podem ser definidas pelo próprio usuário. Geralmente, as bibliotecas contém portas lógicas, pinos de entrada e saída, *buffers*, multiplexadores,

flip-flops, *latches*, decodificadores, registradores, contadores, comparadores, memórias, funções aritméticas e outras funções especiais.

À medida que os projetos ficam mais complexos, as descrições em nível de esquemas lógicos tornam-se inviáveis, fazendo-se necessário descrever esses projetos em modos mais abstratos. As linguagens de descrição de hardware, também conhecidas como HDLs, foram desenvolvidas para auxiliar os projetistas a documentar projetos e simular grandes sistemas, principalmente em projetos de dispositivos ASICs.

Existem diversas linguagens de descrição de hardware disponíveis, sendo as mais comumente utilizadas: ABEL (*Advanced Boolean Equation Language*), VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) e Verilog.

A linguagem ABEL foi a primeira linguagem HDL a ser desenvolvida. Criada pela empresa americana Data I/O Corp. para programar dispositivos SPLD, é uma linguagem mais simples que a lingaguem VHDL

Já as linguagens VHDL e Verilog são capazes de programar sistemas de maior complexidade como, por exemplo, os dispositivos FPGA.

As linguagens de descrição de hardware HDL são utilizadas para descrever o comportamento de um sistema digital de variadas formas, inclusive equações lógicas, tabelas verdade e diagramas de formas de onda, que utilizam declaração de constante, estados, configurações, bibliotecas, módulos, etc.

Nessa apostila será trabalhado apenas o desenvolvimento de projetos a partir de um editor gráfico.

3.2. Síntese Lógica e mapeamento da tecnologia

A síntese lógica consiste em duas fases distintas: otimização lógica para minimizar equações booleanas e mapeamento da tecnologia para converter equações em células da biblioteca da tecnologia-alvo.

Como a lógica inicial não está otimizada, algoritmos de síntese são utilizados para simplificar as equações booleanas geradas. A síntese permite, na prática, a redução de área a ser ocupada no circuito integrado, como também reduz o atraso da propagação (*delay*) dos sinais envolvidos.

A fase de mapeamento da tecnologia seleciona um conjunto de portas lógicas de uma dada biblioteca para implementar as representações abstratas, enquanto melhora a área, o atraso ou a combinação de ambos, levando em considerações as restrições arquiteturais da tecnologia-alvo, nesse caso, os FPGAs.

3.3. Posicionamento e Roteamento

Após a minimização lógica e o mapeamento da tecnologia, o projeto consiste em uma representação textual de componentes lógicos a serem designados componentes físicos de uma arquitetura FPGA.

O posicionamento e roteamento são dois processos mutuamente dependentes. O posicionamento é a atribuição de componentes particulares do circuito integrado aos componentes lógicos de projeto. O roteamento é a atribuição de trilhas e elementos programáveis, consumindo os recursos disponíveis de interconexão para a comunicação entre os componentes.

O *software* de roteamento aloca os recursos de roteamento do FPGA para interconectar as células posicionadas. As ferramentas de roteamento devem assegurar que 100% das conexões requeridas sejam realizadas e devem procurar maximizar a velocidade das conexões críticas, porém essa meta nem sempre é alcançada.

3.4. Verificação e Teste

A simulação é o tipo mais comum de verificação utilizada em projetos com FPGAs. Ela é realizada geralmente em fase inicial, para verificação funcional, podendo ser realizada em nível comportamental ou em nível de portas lógicas.

A simulação também é realizada antes da configuração do FPGA, para verificar restrições e temporização. Vários simuladores para a tecnologia FPGA estão disponíveis comercialmente como, por exemplo, os *softwares* EDA:

(a) Viewsim®: desenvolvido pela empresa Mentor Graphics.
(b) Synopsys®: desenvolvido pela empresa Synopsys.

(c) Quartus®: desenvolvido pela empresa Altera Corp.

3.5. Programação do FPGA

Após a verificação e teste a implementação do projeto é completada, mas ainda resta um passo final que é a programação do FPGA. Nesse ponto é gerado um arquivo de configuração, que deve ser carregado no dispositivo-alvo.

Um FPGA pode ser programado de diversos modos. Um dos modos recomendados é a interface JTAG (por um cabo de impressora padrão), pois o arquivo de configuração pode ser transferido, pela porta de comunicação paralela do computador para o dispositivo FPGA.

4. Ambiente de Software EDA

Software EDA é uma categoria de ferramentas de *software* para projetar sistemas eletrônicos, tais como placas de circuitos impressos e circuitos integrados. Nessa apostila o *software* utilizado será o Quartus®II *Web Edition* versão 9.2.

Por intermédio desse ambiente é possível fazer simulação, testes e programação de dispositivos FPGAs. As fases de implementação de um projeto de circuito digital podem ser divididas em entradas de dados, compilação, simulação e programação.

A entrada de dados do projeto consiste em fornecer ao programa a especificação do projeto. Essa entrada pode ser realizada das seguintes formas:

(1) Editor Gráfico: Um diagrama lógico, desenvolvido a partir de elementos primitivos, portas lógicas básicas e outros componentes disponíveis em bibliotecas, que podem ser inseridos e interligados para criar o projeto.

(2) Editor de Texto: Uma descrição abstrata do circuito lógico, utilizando se comandos de uma linguagem estruturada de descrição de *hardware* como AHDL, VHDL ou Verilog, que descrevem o comportamento ou o funcionamento do circuito lógico.

(3) Editor de Símbolo Gráfico: Nesse caso, os elementos do diagrama lógico são símbolos gráficos criados pelo usuário ou macroinstruções gráficas existentes nas bibliotecas do *software*, que implementam alguma função lógica.

(4) Editor de forma de onda: Nesse caso, os dados de entrada são formas de onda que implementam alguma função desejada. A partir das formas de onda de entrada e saída o programa implementa a função lógica.

4.1. Software Quartus®II

Ao longo do curso será utilizado o *software* Quartus®II da Altera Corp., para se implementar e testar projetos de sistemas digitais, a serem desenvolvidos em aulas práticas. Este *software* é um ambiente de desenvolvimento integrado, ou EDA (*Electronic Design Automation*), que permite a realização de todas as etapas envolvidas no projeto de um sistema digital, desde a descrição de sua lógica, por meio de diagramas esquemáticos ou linguagens de descrição, simulação do circuito desenvolvido, até a gravação do projeto em um dispositivo lógico programável como um CPLD ou FPGA. O Quartus®II trabalha com um sistema orientado a projetos, ou seja, o *software* associa diversos arquivos à um projeto. Este sistema possibilita a utilização de componentes básicos em diversos projetos e simplifica a elaboração de projetos mais complexos uma vez que as funcionalidades de um grande sistema podem ser divididas em diversos sistemas mais simples e por fim, agrupados.

4.2. Criando Projeto com Quartus®II

A forma mais simples de se criar um projeto no Quartus®II é por meio do aplicativo *Project Wizard*. Para executar tal aplicativo, clique no menu **File > New Project Wizard.**

Com isto a tela de apresentação do aplicativo Project Wizard irá se abrir. Clique em **Next**. Nesta tela pode-se escolher o diretório do projeto, o nome do projeto e o nome da entidade principal do projeto. É recomendável que para cada projeto, uma pasta diferenciada seja criada, para que não haja conflito entre os diversos arquivos gerados para cada projeto.

Uma vez preenchidos os campos corretamente, clique em **Next** novamente. Nesta tela é possível incluir arquivos ao projeto. Através do botão "**...**" ao lado do campo **File Name** é possível definir o diretório do arquivo a ser incluído. Uma vez selecionado o arquivo, clique em **Add** para adicioná-lo ao projeto. Quando todos os projetos desejados forem incluídos, clique em **Next**. Com isso a próxima tela aparecerá.

OBS: Caso esteja criando um projeto que não se utiliza de componentes

Figura 7 Tela de Introdução ao *Project Wizard*

previamente criados apenas clique em **Next**.

Nesta tela deve-se escolher o componente a ser utilizado para implementar o projeto, ou seja, em qual dispositivo lógico programável o sistema será gravado. A família do dispositivo pode ser definida no campo **Device Family**. Na seção **Show in Avaliable device list**, na lista **Available devices** são apresentados todos os componentes da família escolhida que atendam aos requisitos definidos pelo filtro de componentes. Selecione o dispositivo adequado e clique em **Next** para prosseguir para a próxima tela.

OBS: Observe no kit de desenvolvimento qual o dispositivo utilizado.

Nesta janela é possível utilizar outras ferramentas de desenvolvimento em conjunto com o Quartus®II, para isto a ferramenta a ser utilizada deve estar instalada no computador, uma vez que estas ferramentas não acompanham o pacote de instalação do Quartus®II. No curso de sistemas digitais não será utilizada nenhuma ferramenta adicional. Assim sendo, clique em **Next.**

Esta tela apresenta um resumo do projeto a ser criado. Verifique se todos os dados conferem com os desejados, caso haja algo incorreto, retorne à tela adequada, por meio do botão **back** e corrija o erro. Caso não haja erros, clique em **Finish**. Ao final do processo, a janela de projeto (*Project Navigator*) conterá o projeto criado.

4.3. Desenvolvendo um projeto usando diagrama esquemático

O uso de diagramas esquemáticos é uma das maneiras de se definir a lógica de um sistema digital dentro do ambiente de projeto do Quartus®II. Neste modo de projeto os componentes básicos de eletrônica digital como portas lógicas, flip-flops e multiplexadores, são representados por meio de blocos, os quais podem ser arranjados e associados de forma a se definir uma lógica de funcionamento.

Para se iniciar o desenvolvimento de um projeto que utilize o modo de diagrama

esquemático, é necessário, primeiramente, adicionar ao projeto principal um arquivo que comporte este tipo de entrada. Assim sendo, para adicionar um arquivo de diagrama esquemático ao projeto clique em **File -> New...**, e uma nova janela aparecerá.

Nessa janela é possível escolher o tipo de arquivo que se deseja criar. Selecione a opção **Block Diagram / Schematic File** e clique em **Ok**. Um novo arquivo de diagrama esquemático aparecerá na janela principal do Quartus®II. Para salvar este arquivo clique em **File > Save as...** e dê um nome ao arquivo, salvando-o na pasta do projeto principal.

OBS: Para arquivos de descrição de hardware, deve-se sempre lembrar que o nome da entidade principal do arquivo deve ser igual ao nome do arquivo.

A tela de edição de diagramas esquemáticos é aberta, onde devemos destacar as funções de alguns botões:

(1) Atracar/Desatracar Janela: Permite separar a janela de diagrama esquemático da janela principal do Quartus®II.
(2) Ferramenta de Seleção: Permite selecionar elementos inseridos no arquivo.
(3) Ferramenta de texto: Permite inserir texto no arquivo, com fins explicativos ou de documentação.
(4) Ferramenta de símbolo: Permite inserir símbolos (componentes). Seu uso será detalhado posteriormente.
(5) Ferramenta de conexão: Realiza a ligação entre terminais dos componentes.
(6) Ferramenta de barramento: Realiza a ligação de barramentos (conjunto de sinais associados).

Através da ferramenta de símbolos é possível inserir componentes no arquivo. Clique nela, ou dê dois cliques na tela em branco, para inserir um componente. Feito isso uma janela com a biblioteca de componentes do *software* será aberta.

Nessa janela, no campo **Libraries**, é possível explorar as diversas coleções de componentes do Quartus®II. As mais relevantes são listadas abaixo:

(1) Megafunctions: Tratam-se de funções complexas já prontas, necessitando muitas vezes de configuração. Serão melhor explicadas futuramente.

(2) Others: esta coleção possui modelos de componentes equivalentes aos da família de dispositivos 74xx. Dessa forma é possível realizar qualquer combinação lógica feita com componentes discretos dentro do CPLD.

(3) Primitives: componentes básicos:

- **(a) Buffers**: diversos tipos de *buffers*, incluindo os de saída *tri-state*.
- **(b) Logic**: elementos lógicos básicos (AND, OR, NOR, XOR, etc.) com diversos números de entradas.
- **(c) Others**: elementos diversos.
- **(d) Pin**: diversos tipos de pinos suportados pelo componente: entrada, saída ou bidirecional.
- **(e) Storage**: elementos de armazenamento de dados, ou seja, flip-flops de diversos tipos.

Selecione o componente desejado e clique **OK.**

EXEMPLO: Para exemplificar o modo de desenvolvimento via diagrama esquemático, será realizada a montagem de um circuito combinacional capaz de executar a operação:

$$S(AB) = (A.B) \oplus (C.D)$$

Primeiramente insira as portas lógicas necessárias e os pinos de entrada e saída, como mostrado na figura 8. Para renomear os pinos de entrada e saída, dê duplo clique sobre o pino a ser alterado.

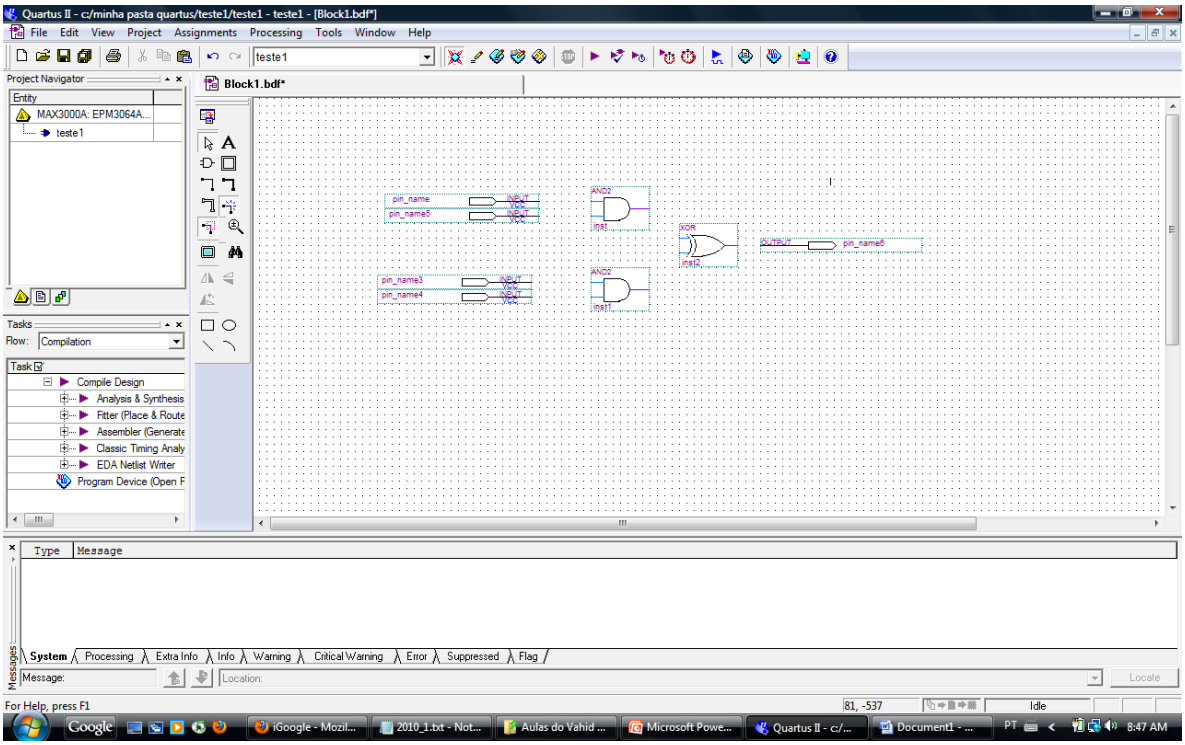


Figura 8 Criação da função $S(AB) = (A.B) \oplus (C.D)$.

Faça as conexões entre os pinos de entrada e saída e as portas lógicas utilizando a ferramenta de conexão. Para isso clique nos terminais a serem conectados, observando se de fato foi feito o contado.

Compile o projeto. Para isso clique em **Processing > Start Compilation**, ou clique no item de compilação na janela principal. Esse processo é bastante complexo e pode demorar vários segundos para ser concluído. Na janela de status é possível acompanhar a evolução de cada uma das etapas de compilação.

Ao final do processo de compilação aparecerá, abaixo da tela do diagrama esquemático, uma janela de mensagens com o resultado da compilação. Caso algum erro ocorra no processo de compilação, ele será mostrado nesta janela, com indicações do motivo da ocorrência e, no caso de linguagens de descrição de *hardware*, da parte do código onde se encontra o erro. Corrija sempre os ocasionais erros e repita o processo de compilação.

4.4. Simulação de Projetos

O Quartus®II permite realizar simulações do projeto desenvolvido. Essas simulações podem ser de dois tipos: funcional e de tempo.

A simulação funcional permite verificar se a funcionalidade do projeto desenvolvido está correta, em outras palavras, a simulação funcional verifica a lógica implementada. Por sua vez, a simulação de tempo verifica as características de temporização, que são inerentes à construção interna do CPLD ou FPGA usado e da forma como o projeto é construído dentro do chip, ou seja, simulação temporal verifica o funcionamento real do projeto, considerando os atrasos das portas envolvidas.

O primeiro passo para simular é criar um arquivo de entradas de simulação. Esse arquivo vai aplicar às entradas de sinais as condições possíveis de funcionamento de forma que a saída possa ser então avaliada. Para criar um arquivo de simulação, clique em **File > New...**, escolha o tipo de arquivo **Vector Waveform File**, e a janela da figura 9 será aberta:

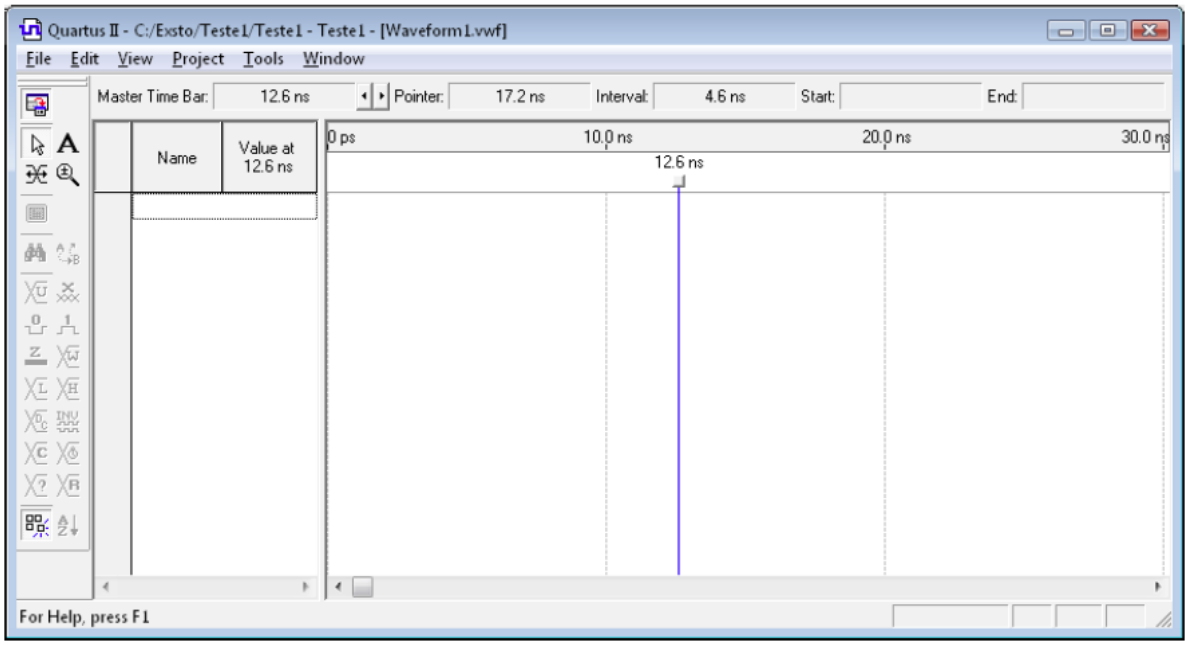


Figura 9 Janela do arquivo de simulação.

Nessa tela é possível a incluir os sinais de entrada e desenhar as formas de ondas dos sinais aplicados. Para inserir sinais vá a **Edit > Insert > Insert Node or Bus...** ou dê um clique duplo na tela em branco logo abaixo da coluna **Name**.

Fazendo isso, a tela da figura 10 será aberta:

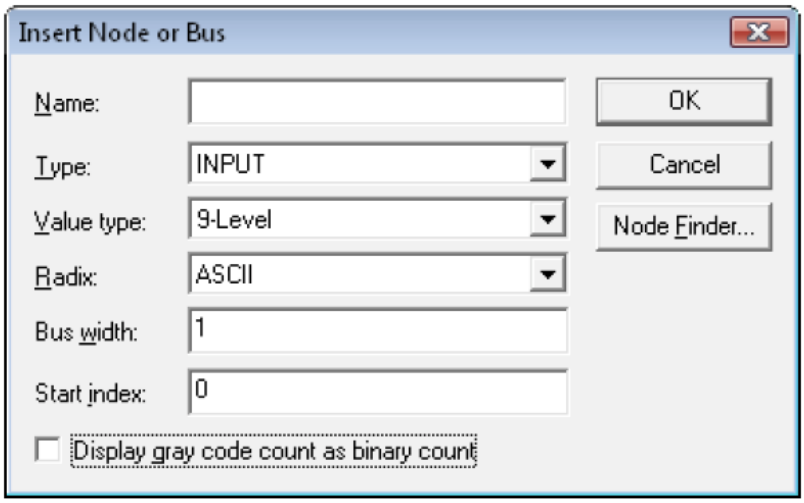


Figura 10 Tela de inserção de variável para simulação.

Nesta tela, clique em **Node Finder...**, e a tela da figura 11 aparecerá:

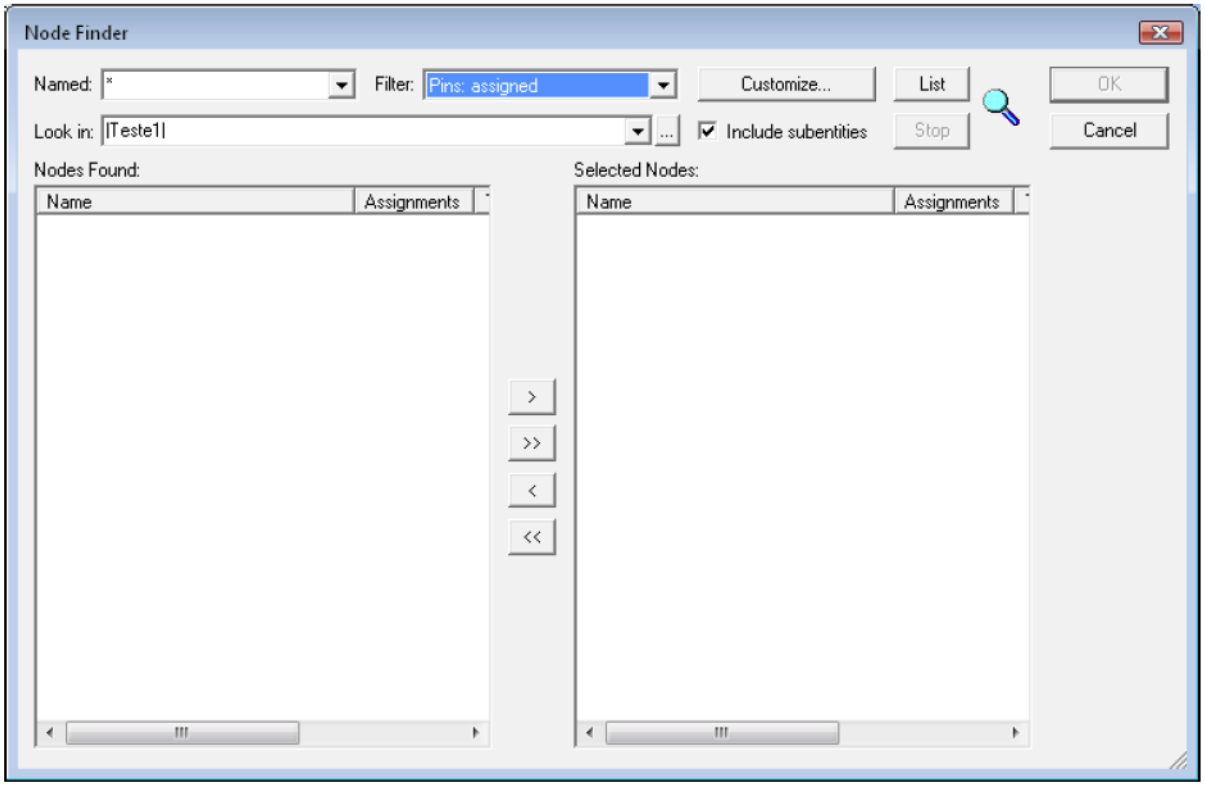


Figura 11 Utilizando o Node Finder.

No campo **Filter** é possível escolher a categoria de sinais com os quais se deseja trabalhar. O campo **Look in** informa qual projeto ou arquivo será usado. Por fim clicando em **List** serão apresentados todos os sinais do projeto. Para ver todos os *pins* disponíveis, selecione **Pins: all** no campo **Filter**. Para incluir um sinal na simulação, selecione-o no campo **Nodes Found** e clique no botão "**>**". Os sinais selecionados serão transferidos para o campo **Selected Nodes**. Devem ser selecionados os sinais de entrada, e os de saída, cujo resultado se deseja analisar. Uma vez que todos os sinais tenham sido escolhidos, clique em **Ok**.

Antes de se iniciar a edição dos sinais de entrada, é necessário definir o tempo total de simulação e o tamanho do grid. Para isso, inicialmente clique em **Edit > End Time...**, a seguinte tela da figura 12 será aberta:

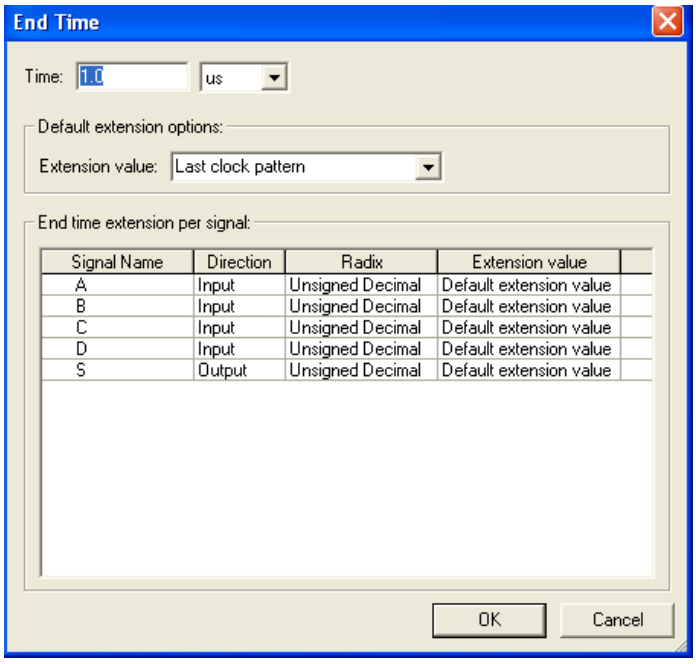


Figura 12 Definindo o tempo total de simulação.

Nesta janela, no campo **Time** pode-se definir o tempo total, no qual se deseja analisar o comportamento do sistema projetado. Uma vez definido o tempo total de simulação, clique em **Edit > Grid Size...**, e defina o tamanho do grid, ou seja, o intervalo de tempo que será demarcado na tela do editor de forma de onda. A utilização do grid facilita o trabalho de desenhar as formas de onda desejadas. A tela do Grid Size é mostrada na figura 13.

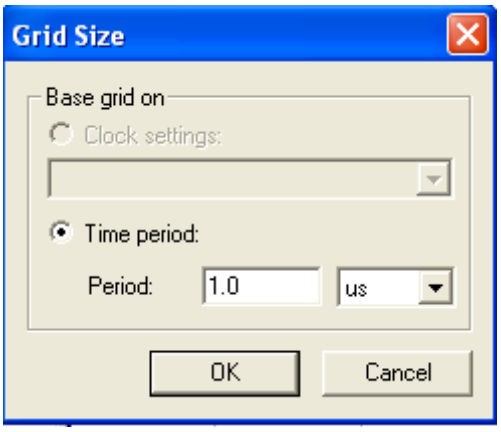


Figura 13 Definindo o tempo de grid.

OBS: Para facilitar a visualização de eventos é interessante que o tempo total de simulação e o tamanho do grid sejam da mesma ordem de grandeza.

Uma vez definidos estes itens, pode-se iniciar o processo de desenhar as formas de onda. Para tal utiliza-se os botões encontrados na parte esquerda da tela de edição de formas de onda, dos quais alguns são descrito na figura 14:

Escolha **Functional** no campo **Simulation mode**, dentro de **Assignments > Settings > + Simulator Settings.** Em seguida selecione o arquivo de formas de onda no campo **Simulation Input**. Por fim, clique em **Ok.**

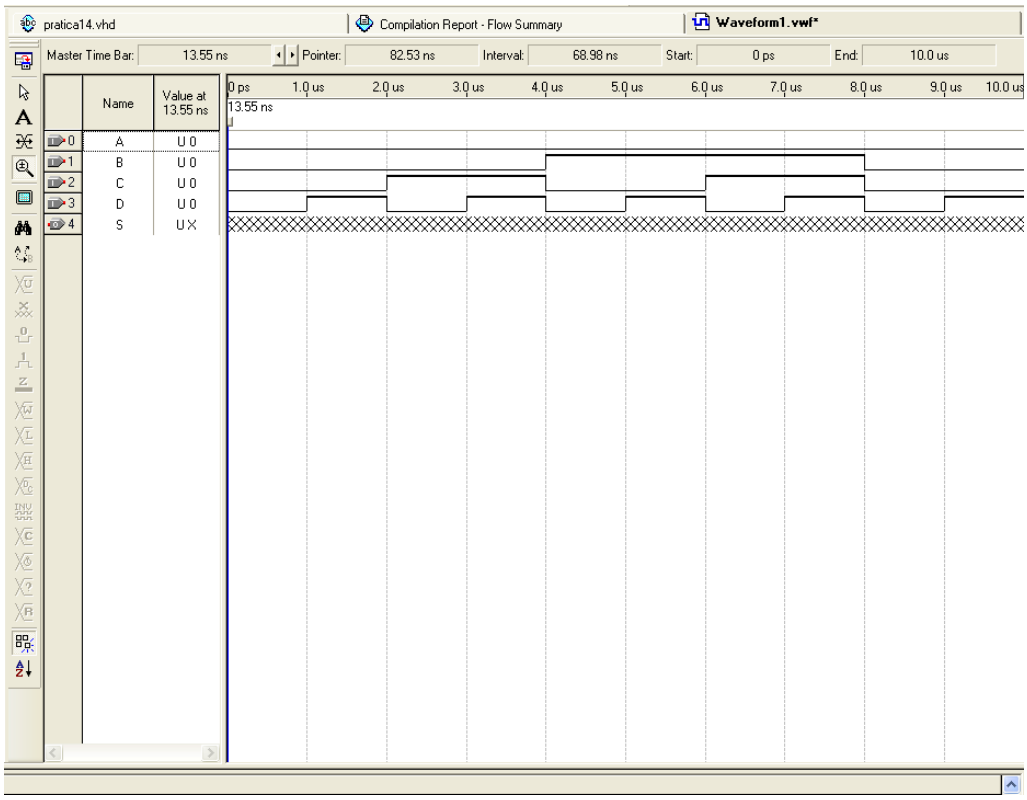
Antes da simulação o projeto precisa ser compilado para a simulação funcional pelo **Processing > Generate Functional Simulation Netlist**. Seguinte, para dar início agora a simulação, clique em **Processing > Start Simulation**. Esses dois processos podem demorar algum tempo.

Ao completar a simulação, uma nova janela semelhante a janela de edição de forma de onda deve ser apresentada. Caso esta janela não apareça, clique em **Processing > Simulation Report**.

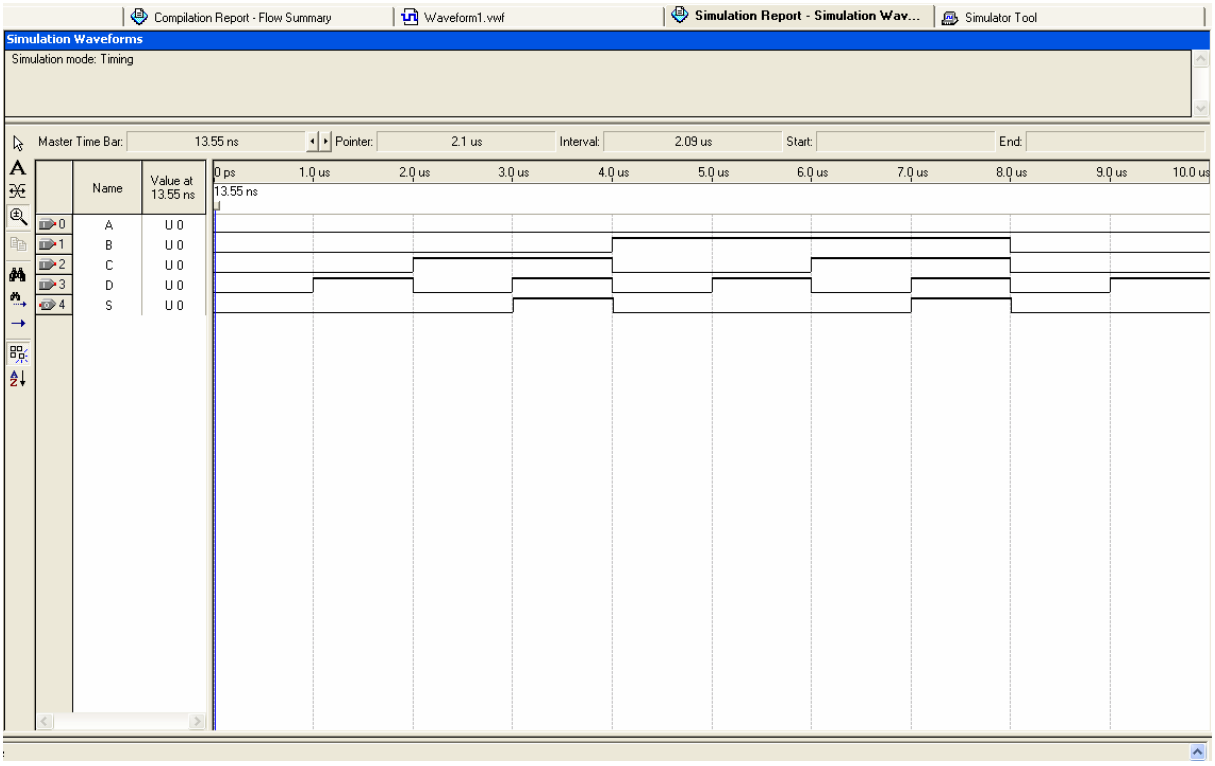
Botão	Nome	Função
	Zoom	Permite aumentar e diminuir o zoom da tela de tempo, isto é, mudar a escala de tempo apresentado. Clique com botão direito: aumenta o zoom Clique com botão esquerdo: diminui o zoom
	Edição de forma de onda	Use essa ferramenta para selecionar o intervalo de tempo que terá seu estado alterado. Uma vez selecionado esse intervalo, os botões a seguir permitem alterar a forma de onda.
	Desconhecido	O estado do pino não é definido, permanece desconhecido
	Nível lógico baixo	Força o pino para 0
	Nível lógico 1	Força o pino para 1
	Alta impedância	Mantém o pino em alta impedância
	Não importa (don't care)	O estado do pino não é definido.
	Contagem	Gera um contagem nos pinos. Só pode ser usado para barramentos (conjuntos de pinos)
	Clock	Aplica um sinal de clock de freqüência definida.

Figura 14 Descrição dos botões do editor de forma de onda.

As figuras 15.a e 15.b mostram as telas do editor de formas de onda e do **Simulator Report**, respectivamente, após a edição das formas de onda dos sinais de entrada e da simulação do projeto proposto pelo exemplo.



(a)



(b)

Figura 15 Telas de simulação do exemplo proposto em **(a)** antes da simulação e **(b)** depois da simulação.

4.5. Definição dos pinos de I/O

Depois de criado e simulado, deve-se definir a qual pino físico do FPGA cada pino do esquemático está ligado. Para isso clique em **Assignments > Pins**. Abrirá a janela da figura 16, que permite definir a ligação entre os pinos físicos e os pinos do diagrama.

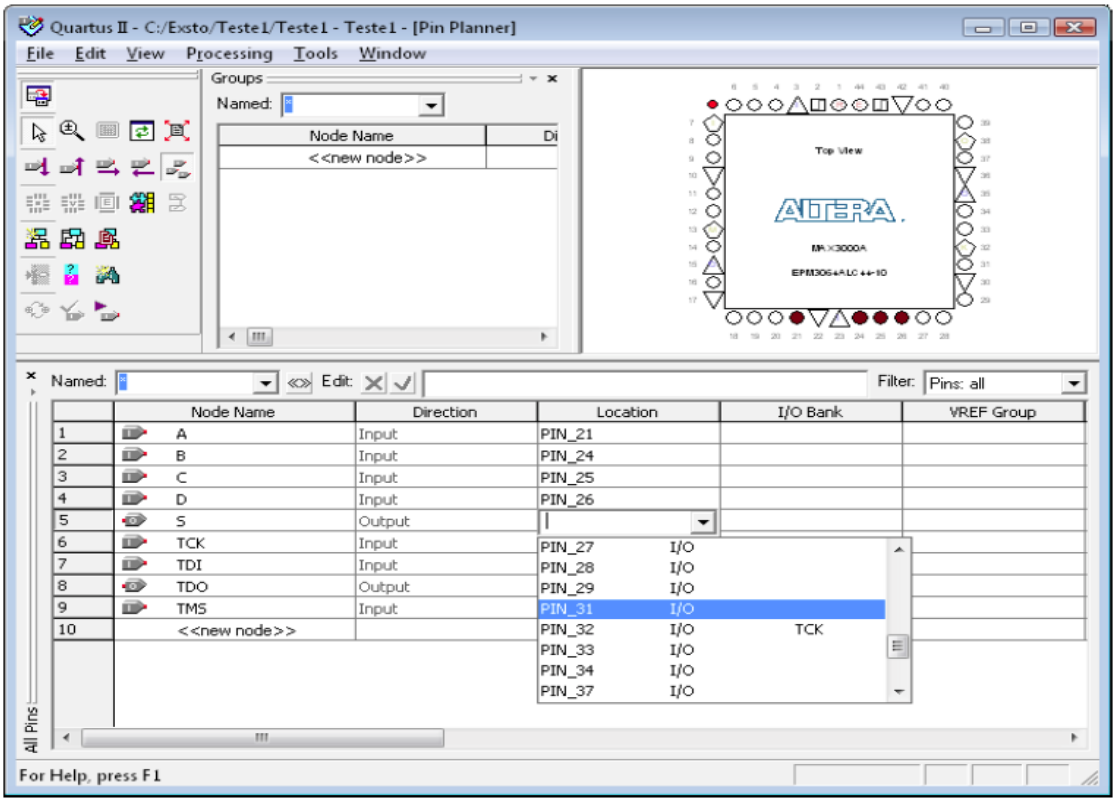


Figura 16 Definição dos pinos do FPGA.

Nessa tela é apresentada uma figura da pinagem do FPGA. Na coluna **Node Name** estão listados os pinos presentes no projeto. Na coluna **Direction** está descrita a direção desse pino. Na coluna **Location,** pode-se definir a qual pino físico estará ligado o pino lógico. Para isso, dê um clique duplo em uma célula da coluna **Location** e escolha o pino desejado na lista que aparecerá. Repita esse processo para todos os pinos.

Para validar a atribuição de pinos repita a compilação e observe se não ocorrem erros.

4.6. Gravação no FPGA

Feita a associação dos pinos do FPGA, pode-se então gravar o projeto no dispositivo lógico programável disponível. Clique em **Tools > Programmer**, abrindo a tela da figura 17:

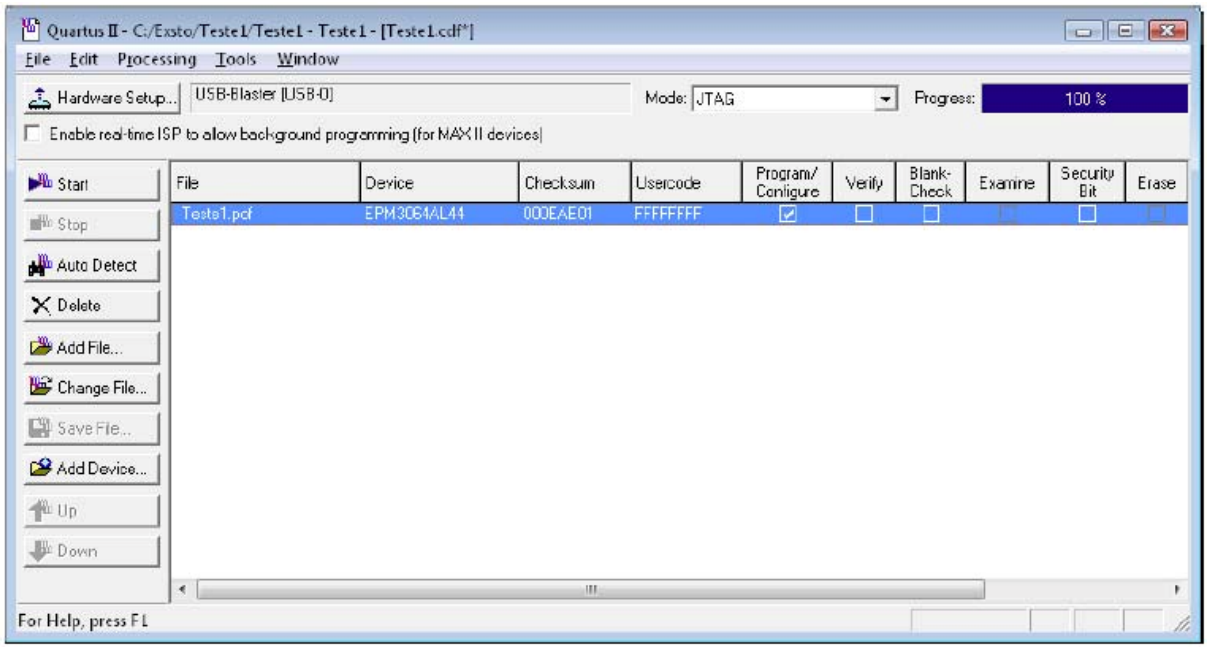


Figura 17 Programando o FPGA.

Na área de configuração apresentada, é preciso assegurar que as opções de *hardware* estejam corretas, através do atalho **Hardware Setup...**. Caso não haja nenhum hardware selecionado, em **Add Hardware,** selecione manualmente ByteBlaster ou USBBlaster, conforme o gravador do kit utilizado.

Ainda é preciso marcar a opção **Program/Configure**. Outras opções como verificação da gravação são também disponibilizadas. Clique em **Start** para iniciar gravação. A janela de mensagens da tela principal do Quartus®II irá informar se a gravação foi executada com sucesso. Caso isso ocorra, seu projeto já estará rodando no FPGA.

Caso haja algum erro de gravação, verifique se os cabos estão corretamente ligados, se o kit está ligado, o modelo de gravador está correto ou ainda se o modelo de CPLD está correto.

5. Dicas

O software Quartus®II apresenta muitos recursos, por meio de suas bibliotecas internas, onde estão disponíveis diversos componentes lógicos, que podem auxiliar muito o projetista, principalmente no modo editor gráfico. As bibliotecas estão dividas em duas categorias: primitivas e megafunções.

5.1. Biblioteca Primitiva

A biblioteca primitiva possui quatro grupos de componentes: elementos lógicos, flip-flops e latches, buffers e terminais de E/S. Para utilizar o componente, basta na janela símbolo (*symbol*) do editor gráfico digitar o nome do componente e dar um clique no botão **Ok**. Cada grupo de componente é listado a seguir:

Elementos Lógicos

Primitiva	Descrição	Nome
AND	Saída = lógica AND das entradas. Número de entradas: 2, 3, 4, 6, 8 ou 12	AND2, AND3, AND4, AND6, AND8 ou AND12
NOR	Saída = lógica NOR das entradas. Número de entradas: 2, 3, 4, 6, 8 ou 12	NOR2, NOR3, NOR4, NOR6, NOR8 ou NOR12
NOT	Saída = inverso da entrada	NOT
OR	Saída = lógica OR das entradas. Número de entradas: 2, 3, 4, 6, 8 ou 12	OR2, OR3, OR4, OR6, OR8 ou OR12
VCC	Define um fio ou barramento para VCC	VCC
GND	Define um fio ou barramento para GND	GND

XNOR	Saída = lógica exclusive NOR das entradas 1 e 2. Número de entradas: 2	XNOR
BAND	Saída = lógica AND das entradas invertidas. Número de entradas: 2, 3, 4, 6, 8 ou 12	BAND2, BAND3, BAND4, AND6, BAND8 ou BAND12
BNAND	Saída = lógica NAND das entradas invertidas. Número de entradas: 2, 3, 4, 6, 8 ou 12	BNAND2, BNAND3, BNAND4, BNAND6, BNAND8 ou BNAND12.
BNOR	Saída = lógica NOR das entradas invertidas. Número de entradas: 2, 3, 4, 6, 8 ou 12	BNOR2, BNOR3, BNOR4, BNOR6, BNOR8 ou BNOR12
BOR	Saída = lógica OR das entradas invertidas. Número de entradas: 2, 3, 4, 6, 8 ou 12	BOR2, BOR3, BOR4, BOR6, BOR8 ou BOR12
XOR	Saída = lógica exclusive OR das entradas. Número de entradas: 2	XOR
NAND	Saída = lógica NAND das entradas. Número de entradas: 2, 3, 4, 6, 8 ou 12	NAND2, NAND3, NAND4, NAND6, NAND8 ou NAND12

Flip Flop e Latches

Primitiva	Descrição	Nome
SRFF	Flip-flop tipo SR, entradas Set, Reset, Clock, Clear e Preset. Saída = Q	SRFF
SRFFE	Flip-flop tipo SR, entradas Set, Reset, Clock, Clear, Preset e Enable. Saída = Q	SRFFE
TFF	Flip-flop tipo T, entradas T, Clock, Clear e Preset. Saída = Q	TFF
TFFE	Flip-flop tipo T, entradas T, Clock, Clear, Preset e Enable. Saída = Q	TFFE
DFF	Flip-flop tipo D, entradas D, Clock, Clear e Preset. Saída = Q	DFF
DFFE	Flip-flop tipo D, entradas D, Clock, Clear, Preset e Enable. Saída = Q	DFFE
DFFEA	Flip-flop tipo D, entradas D, Clock, Clear, Preset, Enable, Asynchronous Data e Asynchronous Load. Saída = Q	DFFEA

DFFEAS	Flip-flop tipo D, entradas D, Clock, Clear, Preset, Enable, Asynchronous Data, Asynchronous Load, Synchronous Clear e Synchronous Load. Saída = Q	DFFEAS
JKFF	Flip-flop tipo JK, entradas J,K, Clock, Clear e Preset. Saída = Q	JKFF
JKFFE	Flip-flop tipo JK, entradas J, K, Clock, Clear, Preset e Enable. Saída = Q	JKFFE
LATCH	Flip-flop tipo Latch, entradas D e Enable. Saída = Q	LATCH

Buffers

Primitiva	Descrição	Nome
LCELL	A primitiva LCELL buffer aloca uma célula lógica para o projeto.	LCELL
GLOBAL	Os sinais de Clock, Preset, Clear, Output Enable, Clock Enable, Synchronous Clear, Memory Read Enable, Memory Write Enable e Synchronous Load podem ser sinais globais. A primitiva GLOBAL buffer possibilita que um sinal assíncrono possa ser utilizado como global. Por exemplo: Clock, Output Enable, Register Control ou Memory Enable Input.	GLOBAL
ROW_GLOBAL	A primitiva ROW_GLOBAL buffer possibilita que um sinal síncrono possa ser utilizado como global. Por exemplo: Clock, Clear, Preset, Output Enable ou Memory Enable Input.	ROW_GLOBAL
CLKLOCK	A primitiva CLKLOCK buffer habilita o circuito de phase loked loop.	CLKLOCK
TRI	A primitiva TRI é um buffer tri-state com entrada, saída e entrada de Enable. Se a entrada de Enable for nível 1, a saída será conectada à entrada.	TRI
SOFT	A primitiva SOFT buffer especifica que uma célula lógica pode ser necessária no projeto.	SOFT
CASCADE	A primitiva CASCADE buffer permite a ligação de saída em cascata entre portas AND e Portas OR.	CASCADE
EXP	A primitiva EXP (expander) buffer especifica que uma expansão de termos produtos é necessária no projeto.	EXP
WIRE	A primitiva WIRE buffer é utilizada para renomear um vetor de entrada (node) ou um barramento. Não é associada a nenhum elemento lógico, e a entrada é igual à saída.	WIRE
CARRY	A primitiva CARRY buffer determina a saída lógica carry out (vai um) para uma função e atua como carry in de entrada em outra função.	CARRY
CARRY_SUM	A primitiva CARRY_SUM buffer possui duas entradas e duas saídas, que determinam o carry e a soma de uma saída lógica para uma função.	CARRY_SUM
OPNDRN	A primitiva OPNDRN buffer é similar a primitiva TRI buffer, com uma única entrada e uma única saída. Não possui entrada de Enable.	OPNDRN

Terminais E/S

Primitiva	Descrição	Nome
BIDIR	A primitiva BIDIR representa um terminal bidirecional, ou seja, entrada e saída.	BDIR
INPUT	A primitiva INPUT representa um terminal de entrada.	INPUT
OUTPUT	A primitiva OUTPUT representa um terminal de saída.	OUTPUT

5.2. Biblioteca de Megafunções

A biblioteca de megafunções oferece uma variedade de módulos parametrizáveis (LPM) e outras funções também parametrizáveis. Cada grupo de componentes é listado a seguir:

Componentes Aritméticos

Megafunção	Descrição	Nome
altaccumulate	Função parametrizável: acumulador	altaccumulate
altfp_mult	Função parametrizável: multiplicador com ponto flutuante	altf_mult
altmemmult	Função parametrizável: multiplicador com memória	altmemmult
altmult_accum	Função parametrizável: multiplicador com acumulador	altmult_accum
altmult_add	Função parametrizável: multiplicador somador	altmult_add
divide	Função parametrizável: divisão	divide
lpm_abs	Função parametrizável: valor absoluto	lpm_abs
lpm_add_sub	Função parametrizável: somador/subtrator	lpm_add_sub
lpm_compare	Função parametrizável: comparador	lpm_compare
lpm_counter	Função parametrizável: contador	lpm_counter
lpm_divide	Função parametrizável: divisor	lpm_divide
lpm_mult	Função parametrizável: multiplicador	lpm_mult
parallel_add	Função parametrizável: somador paralelo	parallel_add

Megafunção	Descrição	Nome
altcsmem	Função parametrizável:gera automaticamente a lógica necessária para executar o TDM (Time Domain Multiplex) entre múltiplos FIFOs.	altcsmem

Portas

Megafunção	Descrição	Nome
busmux	Função parametrizável: multiplexador (mux). Equivalente à função lpm_mux com o parâmetro LPM_SIZE configurado para 2.	busmax
lpm_and	Função parametrizável: porta AND.	lpm_and
lpm_bustri	Função parametrizável: buffer tri_state.	lpm_bustri
lpm_clshift	Função parametrizável: lógica combinacional com deslocamento.	lpm_clshift
lpm_constant	Função parametrizável: converte um parâmetro numa constante.	lpm_constant
lpm_decode	Função parametrizável: decodificador.	lpm_decode
lpm_inv	Função parametrizável: inversor.	lpm_inv
lpm_mux	Função parametrizável: multiplexador (mux).	lpm_mux
lpm_or	Função parametrizável: porta OR.	lpm_or
lpm_xor	Função parametrizável: porta XOR.	lpm_xor
mux	Função parametrizável: multiplexador (mux). Equivalente à função lpm_mux com o parâmetro LPM_SIZE configurado para 2.	mux

Componentes de E/S

Megafunção	Descrição	Nome
altcdr_rx	Função parametrizável: implementa um receptor de CDR (Clock Data Recovery).	altcdr_rx
altcdr_tx	Função parametrizável: implementa um transmissor de CDR (Clock Data Recovery).	altcdr_tx
altclkctrl	Função parametrizável: implementa um buffer que seleciona clocks.Pode selecionar entre clock global, clock regional, dual regionalclock ou caminhos de clocks externos.	altclkctrl
altclklock	Função parametrizável: habilita o circuito clocklock – PLL (phase – locked loop).	altclklock
altddio_bidir	Função parametrizável: entrada bidirecional Double Data Rate (DDR), transmite e recebe dados em ambas as bordas do pulso de clock de referência.	altddio_bidir
altddio_in	Função parametrizável: entrada Double Data Rate (DDR), recebe dados em ambas as bordas do pulso de clock de referência.	altddio_in
altddio_out	Função parametrizável: saída Double Data Rate (DDR), transmite dados em ambas as bordas do pulso de clock de referência.	altddio_out
altdq	Função parametrizável: data strobe, transmite e recebe dados em ambas as bordas do pulso de clock de referência.	altdq
altdqs	Função parametrizável: gera um grupo de pinos DQS usado para strobe read/write data em interfaces de memória externas DDR/FCRAM.	altdqs
altgxb	Função parametrizável: habilita o circuito Gigabit transceiver block (GXB).	altgxb
altlvds_rx	Função parametrizável: implementa a recepeção LVDS, que é uma interface de E/S, de alta velocidade, que usa sinal diferencial sem uma voltagem de referência.	altlvds_rx
altlvds_tx	Função parametrizável: implementa a transmissão LVDS, que é uma interface de E/S, de alta velocidade, que usa sinal diferencial sem uma voltagem de referência.	altlvds_tx
altpll	Função parametrizável: habilita o circuito PLL (Phase-Locked Loop).	altpll
altpll_reconfig	Função parametrizável: habilita o controle de reconfiguração em tempo real do circuito PLL (Phase-Locked Loop).	altpll_reconfig
altremote_ update	Função parametrizável: permite reconfiguração remota, em tempo real, de um dispositivo.	altremote_ update
Altufm_osc	Função parametrizável:permite acesso ao oscilador interno da memória flash (UFM).	Altufm_osc

Compilador de Armazenamento

Megafunção	Descrição	Nome
alt3pram	Função parametrizável: memória RAM (triple port).	alt3pram
altcam	Função parametrizável:memória CAM (Content Addressable Memory).	altcam
altdpram	Função parametrizável: memória RAM (dual port).	altdpram
altqpram	Função parametrizável: memória RAM (quad port).	altqpram
altshift_taps	Função parametrizável: memória RAM baseada em registrador de deslocamento (shift register).	altshift_taps
altsyncram	Função parametrizável: memória RAM (dual port).	altsyncram
Altufm_i2c	Função parametrizável: memória flash (UFM) com o protocolo Inter-Integrated Circuit (I2C).	Altufm_i2c
Altufm_none	Função parametrizável: memória Flash (UFM).	Altufm_none
Altufm_parallel	Função parametrizável: memória Flash (UFM) com o protocolo de interface paralela.	Altufm_parallel
Altufm_spi	Função parametrizável: memória Flash (UFM) com o protocolo de interface SPI.	Altufm_spi
csdpram	Função parametrizável: memória RAM (dual port) com ciclo compartilhado.	csdpram
csfifo	Função parametrizável: memória FIFO com ciclo compartilhado.	csfifo
dsfifo	Função parametrizável: memória FIFO com dual clock.	dsfifo
scfifo	Função parametrizável: memória FIFO com único ciclo.	scfifo
lpm_ff	Função parametrizável: Flip-flop.	lpm_ff
lpm_fifo	Função parametrizável: FIFO (single clock).	lpm_fifo
lpm_fifo_dc	Função parametrizável: FIFO (dual clock).	lpm_fifo_dc
lpm_latch	Função parametrizável: Latch.	lpm_latch
lpm_ram_dp	Função parametrizável: memória RAM (dual port).	lpm_ram_dp
lpm_ram_dq	Função parametrizável: memória RAM com portas de entrada e saída separadas.	lpm_ram_dq
lpm_ram_io	Função parametrizável: memória RAM com uma única porta de E/S.	lpm_ram_io
lpm_rom	Função parametrizável: memória ROM.	lpm_rom
lpm_shiftreg	Função parametrizável: Registrador de deslocamento (shift register).	lpm_shiftreg
lpm_dff	Função parametrizável: Flip-flop tipo D e registrador de deslocamento (Shift Register).	lpm_dff
lpm_tff	Função parametrizável: Flip-flop tipo T.	lpm_tff

5.3. Família 74XX

A biblioteca do software Quartus®II oferece uma variedade da Família 74XX, como por exemplo 7400 (Quatro portas NAND de 2 entradas), 7474 (2 Flip Flop tipo D), 74112 (2 Flip Flop tipo JK), entre outros. Para acessá-los é só procurar na opção symbol, digitar o nome e depois **Ok**.

5.4. Criando símbolos

Criar símbolos é um modo de transformar todo um circuito em somente um bloco, armazenado na bibliteca, deixando assim o circuito mais simples, e mais fácil de entender, vejamos um exemplo mostrado na figura 18:

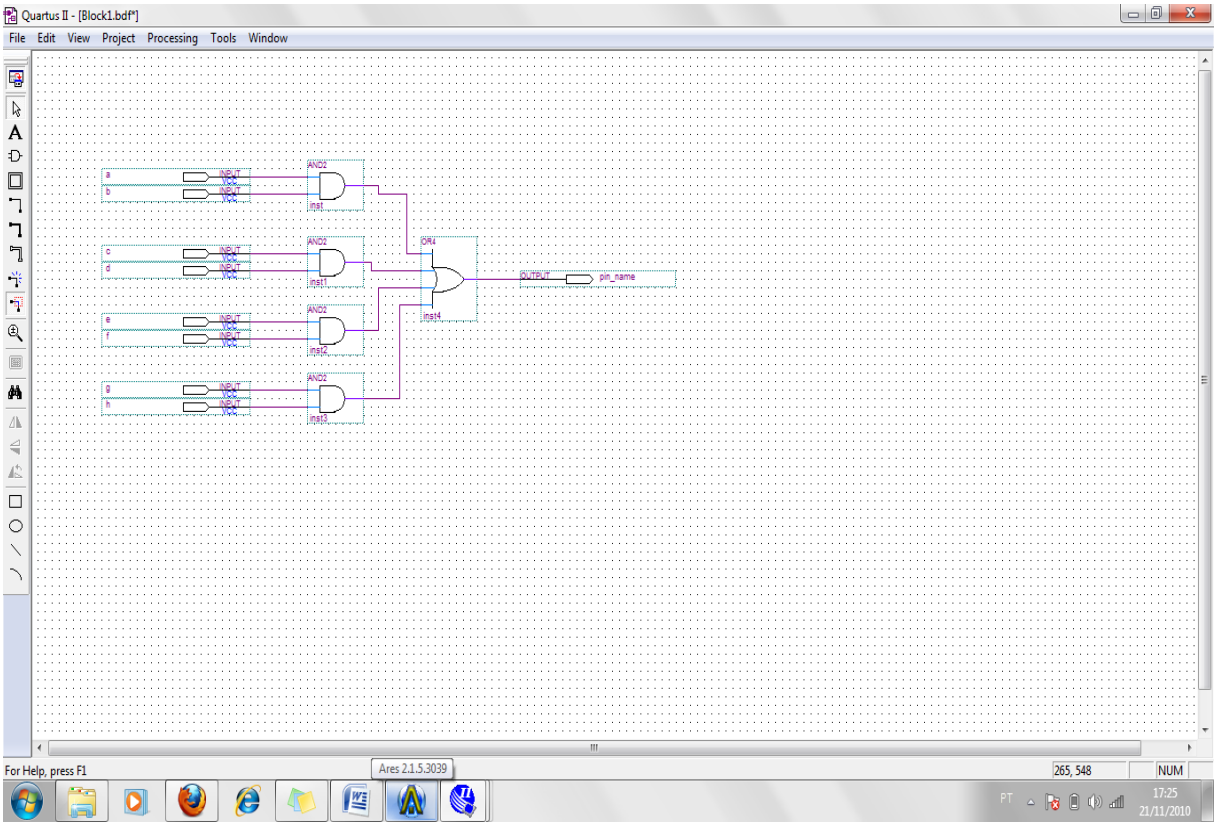


Figura 18 Exemplo utilizado para criação de símbolos.

Para fazer essa criação devemos selecionar o desenho, clicar em **File > Create/Update** e logo depois em **Create Symbol Files for Current File**. Nomeie o símbolo que você está criando e confirme.

Agora o bloco já foi criado e está disponível na biblioteca com o nome escolhido pelo usuário, logo basta inserir seu nome no **Symbol Tool.**

6. Bibliografia

ALTERA. Disponível em: <http://www.altera.com>. Acesso em: 3 ago. 2011.

COIMBRA, J. M. **Altera Cyclone™ II.** Disponível em: <http://pt.scribd.com/doc/38510204/CycloneII>. Acesso em: 3 ago. 2011

COSTA, C. **Projetando Controladores Digitais com FPGA.** São Paulo: Novatec, 2006. 159 p.

RANGEL, F. R. **Aulas FPGAs.** Disponível em: <http://www.dee.ufrn.br/~frangel/ensino/20061/ele365/AulaFPGAs.pdf>. Acesso em: 3 ago. 2011.

WIKIPEDIA: FPGA. Disponível em: <http://pt.wikipedia.org/wiki/FPGA>. Acesso em: 3 ago. 2011.