CENTRO UNIVERSITÁRIO DA FEI

VICTOR SIQUEIRA MARTINS BRAGA

PROJETO DE CIRCUITOS INTEGRADOS DIGITAIS UTILIZANDO-SE DE FERRAMENTAS COMPUTACIONAIS DE DEMONSTRAÇÃO E PROFISSIONAL

São Bernardo do Campo

2014

VICTOR SIQUEIRA MARTINS BRAGA

PROJETO DE CIRCUITOS INTEGRADOS DIGITAIS UTILIZANDO-SE DE FERRAMENTAS COMPUTACIONAIS DE DEMONSTRAÇÃO E PROFISSIONAL

Dissertação apresentada ao Centro Universitário da FEI para a obtenção do título de Mestre em Engenharia Elétrica, orientado pelo Prof. Dr. Salvador Pinillos Gimenez

São Bernardo do Campo

2014

Braga, Victor Siqueira Martins
Projeto de circuitos integrados digitais utilizando-se de ferramentas computacionais de demonstração e profissional / Victor Siqueira Martins Braga. São Bernardo do Campo, 2014.
108 f. : il.

Dissertação de Mestrado - Centro Universitário da FEI.
Orientador: Prof. Dr. Salvador Pinillos Gimenez

1. Circuitos Integrados digitais. 2. Dispositivos Eletrônicos Integrados. 3. Ferramentas CAD. I. Gimenez, Salvador Pinillos, orient. II. Título.

CDU 621.381

CENTRO UNIVERSITÁRIO DA FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Victor Siqueira Martins Braga **Matrícula:** 111304-2

Título do Trabalho: Projeto de circuitos integrados digitais utilizando-se de ferramentas computacionais de demonstração e profissional.

Área de Concentração: Dispositivos Eletrônicos Integrados

Orientador: Prof. Dr. Salvador Pinillos Gimenez

Data da realização da defesa: 09/09/2014

ORIGINAL ASSINADA

A Banca Examinadora abaixo-assinada atribuiu ao aluno o seguinte:

APROVADO ☒ REPROVADO ☐

São Bernardo do Campo, 09 de Setembro de 2014.

MEMBROS DA BANCA EXAMINADORA

Prof. Dr. Salvador Pinillos Gimenez

Ass.: _____

Prof. Dr. Valter Fernandes Avelino

Ass.: _____

Prof. Dr. Wang Jiang Chau

Ass.: _____

VERSÃO FINAL DA DISSERTAÇÃO

ENDOSSO DO ORIENTADOR APÓS A INCLUSÃO DAS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

Dedico este trabalho aos meus familiares, que
sempre tem me dado força e acreditado em mim,
e aos professores e amigos do SENAI,
instituição que tem ajudado a tornar este
trabalho possível.

AGRADECIMENTOS

Neste momento tão especial de minha vida, agradeço aos amigos e familiares, em especial ao meu pai, minha mãe e meu irmão que me deram força e me incentivaram a realizar esta conquista.

Agradeço muito ao meu orientador Prof. Dr. Salvador Pinillos Gimenez, pela ajuda, dedicação e empenho durante este trabalho e também a todo corpo de professores do Centro Universitário da FEI do curso de pós-graduação em Engenharia Elétrica.

Aos funcionários da instituição, principalmente da secretaria da pós-graduação pelos auxílios prestados.

Aos colegas que tive o prazer de conhecer no decorrer do curso, e que sempre estiveram dispostos a me auxiliar.

Ao amigo Filipe Antoine Khatchadourian, por me ajudar nesta conquista.

Agradeço aos colegas do CTI Renato Archer, Tauã, Christian, Wellignton, Seixas e Saulo, pelo suporte dado durante o desenvolvimento deste trabalho.

E finalmente à minha namorada Meire Ayumi, por sempre estar ao meu lado, sendo paciente e me dando o incentivo e auxílio necessário para eu estar aqui neste momento.

RESUMO

Nos dias atuais é cada vez maior a demanda de profissionais na área da microeletrônica, mas a formação de recursos humanos não acompanha este crescimento. Visando estimular a formação e o interesse de novos profissionais, este trabalho é baseado no estudo do fluxo de projeto de circuitos integrados digitais utilizando ferramentas CAD em versões de demonstração e profissionais, com o intuito de demonstrar ser possível iniciar o aprendizado de projetos de circuito integrados digitais com alunos de nível técnico e de graduação. Neste trabalho foi apresentada uma proposta de um fluxo de desenvolvimento de projetos de circuitos integrados digitais utilizando ferramentas computacionais de demonstração, gratuitas, que vão desde a concepção do circuito integrado até a geração automática do leiaute, sendo até ser possível, em alguns casos, fabricar o protótipo do circuito integrado de baixo custo para fins educacionais. Outro objetivo deste trabalho é estudar o fluxo de projeto de circuitos integrados digitais com ferramentas CAD profissionais e realizar a sua implementação no Centro Universitario da FEI. Para a realização desta etapa, foi utilizado um conjunto de *softwares* encontrados na FEI que permitem a realização das descrições HDLs, do processo de sintetização, e da geração de leiaute.

Palavras chave: Dispositivos Eletronicos Integrados. Fluxo de projeto de circuitos integrados digitais. Ferramentas CAD.

ABSTRACT

Nowadays there is an increasing demand for professionals in the field of microelectronics, but the training of human resources does not follow this growth. Aiming to stimulate the interest and training of new professionals, this work is based on the study of the flow of digital integrated circuits design using CAD tools on demo and professional versions, in order to show that it's possible to start learning projects in digital integrated circuit with technical and graduation students levels. In this paper was presented a proposal of a development flow of digital integrated circuits designs using free demo computational tools, ranging from design to the integrated automatic layout generation circuit, being possible to, in some cases, manufacture a prototype of a low cost integrated circuit for educational purposes. Another goal of this work is to study the design flow of digital integrated circuits with CAD tools and professional conduct its implementation in the University Center of FEI. To perform this step, was used a set of software found at FEI, which allow the realization of HDLs descriptions, the synthesizing process, the simulation and layout generation.

Keywords: Integrated Device Electronics. Digital integrated circuits design flow. EDA tools.

LISTA DE FIGURAS

Figura 1- Etapas do desenvolvimento de projetos de CIs digitais utilizando a linguagem VHDL 22
Figura 2- Fluxo de um Projeto digital utilizando a linguagem VHDL..... 24
Figura 3- Estrututa básica de um código escrito em VHDL. 26
Figura 4- Entidade do sistema A constituída pelos subsistemas B, C e D no qual sua representação é dada em diagrama de blocos (a) e sua codificação em linguagem VHDL da entidade do sistema A (b). 27
Figura 5- Arquitetura interna do microcontrolador 8051 da Intel. 30
Figura 6- Etapa inicial do fluxo de desenvolvimento..... 34
Figura 7- Parte do fluxo utilizando ferramentas em versão de demonstração..... 36
Figura 8- Fluxo utilizando ferramentas profissionais..... 37
Figura 9- Fluxograma de projeto de um circuito integrado digital denominado de circuito contador de pulsos de 2 bits..... 40
Figura 10- Diagrama de funcionamento do contador de pulsos 2 bits. 42
Figura 11- Diagrama de blocos do circuito integrado de um contador de pulsos de dois bits..... 43
Figura 12- Código VHDL comentado do Circuito Contador de dois bits..... 44
Figura 13- Resultado do processo de síntese do circuito contador de pulsos de 2 bits. 45
Figura 14- *Hardware* detalhado do circuito contador de pulsos de 2 bits..... 46
Figura 15- Expressões booleanas do circuito contador de pulsos de 2 bits..... 46
Figura 16- Circuito eletrônico completo do contador de pulsos de 2 bits. 47
Figura 17- Simulação lógica do circuito contador de pulsos de 2 bits 48
Figura 18- Simulação de tempo do circuito do contador de pulsos de 2 bits. 49
Figura 19- Esquemático de um *flip-flop* do tipo D em nível de porta lógica. 50
Figura 20- Descrição Verilog do *flip-flop* do tipo D em nível de porta lógica..... 50
Figura 21- Leiaute gerado automaticamente pelo *Microwind* do circuito do *flip-flop* do tipo D. 52
Figura 22- Descrição completa do circuito contador de pulsos de 2 bits, em linguagem Verilog. 53
Figura 23- Leiaute gerado de maneira automática do contador de pulsos de 2 bits. 54
Figura 24- Detalhamento da porta inversora e posicionamento das máscaras utilizadas na tecnologia CMOS 55

Figura 25- Vista em corte das camadas do leiaute 56
Figura 26- Simulação funcional do leiaute do *flip-flop do tipo* D que faz parte do circuito contador de pulsos. 57
Figura 27- Conjunto de portas lógicas que complementam o circuito contador de pulsos, simulado utilizando o *Microwind.* 57
Figura 28- Transcrição para a linguagem Verilog do circuito de portas lógicas que compõe o circuito contador de pulsos de 2 bits 58
Figura 29- Leiaute gerado de maneira automática do circuito de portas lógicas que compõe o circuito contador de pulsos de 2 bits. 59
Figura 30- Resultado da simulação do circuito equivalente a porta ou exclusiva.....59
Figura 31- Tabela que relaciona as combinações de sinais das entradas E1 e E2 com a saída C. 60
Figura 32- Arquitetura gerada a partir do código VHDL do microcontrolador 8051 [18]. 62
Figura 33- Placa didática com FPGA da Altera utilizada para a validação do VHDL do microcontrolador 8051 FONTE:Autor. 63
Figura 34- Diagrama elétrico gerado no *Quartus II,* que foi obtido a partir da compilação do código VHDL do microcontrolador 8051.....64
Figura 35- Esquema elétrico de referência para a execução dos programas de teste no 8051 em FPGA. 65
Figura 36- Indicação dos pinos das portas de entrada/saída para serem utilizadas pelos leds e chave no *Kit* didático de FPGA da ALTERA. 68
Figura 37- Fluxo de projeto de CI digitais onde mostra a utilização do Leonardo Spectrum FONTE: Autor [24]. 70
Figura 38- Fluxo e comandos utilizados no processo de sintetização realizado com *Leonardo Spectrum* [24]. 72
Figura 39- Resultado do comando "report_area" do núcleo do 8051.....73
Figura 40- Arquivo de mapeamento utilizado no projeto do núcleo do 8051. 74
Figura 41- Parte do esquemático do núcleo do 8051.....75
Figura 42- Etapas do fluxo de projeto de CIs digitais da Mentor para a geração automatica de leiaute. 76
Figura 43- Leiaute do núcleo do 8051 gerado de forma automática pelas ferramentas da *Mentor Graphics*.....77
Figura 44- Leiaute completo do núcleo do 8051 (a), leiaute de um bloco que foi colocado em destaque (b) e representação elétrica do bloco em destaque (c). 78

Figura 45- Esquemático do *flip-flop* do tipo D (a) e o correspondente leiaute com suas interconexões (b). 79
Figura 46- Esquemático do *flip-flop* do tipo D com estágio de saída em destaque (a), leiaute do *flip-flop* do tipo D com estágio de saída em destaque (b). 80
Figura 47- Esquemático do circuito inversor da saída Q (a) e o seu leiaute (b), esquemático do circuito inversor da saída QB (c) e o seu leiaute(d). 81
Figura 48- Esquemático de dois circuitos inversores (a) e o leiaute de dois circuitos inversores (b). 82
Figura 49- Tela inicial do Software Quartus. 89
Figura 50- Tela de configuração de projeto. 90
Figura 51- Tela de inclusão de arquivos de projetos. 91
Figura 52- Seleção de Familia de FPGA. 92
Figura 53- Criação de um novo arquivo 93
Figura 54- Tela e criação do VHDL. 94
Figura 55- Código VHDL comentado do Circuito Contador. 94
Figura 56- Project Navigator. 95
Figura 57- Iniciando a Compilação. 96
Figura 58- Mensagem de compilação realizada com sucesso 96
Figura 59- Criação do Arquivo de síntese. 97
Figura 60- Mensagem de criação do arquivo de símbolos. 97
Figura 61- RTL Viewer. 98
Figura 62- Hardware do Contador 98
Figura 63- Descompactação do Bloco Lógico. 99
Figura 64- Diagrama de bloco detalhado do cicuito Contador de Pulsos. 100
Figura 65- Geração das equações booleanas. 100
Figura 66- Expressões Booleanas do circuito contador. 101
Figura 67- Circuito eletrônico do Contador. 102
Figura 68- Iniciando a ferramenta de simulação. 103
Figura 69- Simulator Tool. 103
Figura 70- Janela dos sinais de estímulos para a realização da simulação. 104
Figura 71- Tela de inserção de sinais. 105
Figura 72- Tela de inclusão de sinais para simulação. 105
Figura 73- Sinais encontrados no código VHDL. 106
Figura 74- Arquivo Waveform com sinais já selecionados. 107

Figura 75- Inserindo valor a cada sinal. 107
Figura 76- Arquivo de Waveform com sinais definidos. 108
Figura 77- Iniciando a simulação. 108
Figura 78- Mensagem de simulação ralizada com sucesso 108
Figura 79- Janela da reposta da simulação realizada. 109

LISTA DE ABREVIATURAS

ASIC	Circuitos Integrados digitais de aplicação específica *(Application Specific Integrated Circuit*)
CAD	Projeto assistido por computador *(Computer Aided Design*)
CI	Circuito Integrado
CPLD	Dispositivo Lógico Programável Complexo *(Complex Programmable Logic Device*)
CPU	Unidade Central de Processamento *(Central Processing Unit*)
DARPA	Departamento de Defesa dos Estados Unidos da América *(Defense Advanced Research Projects Agency*)
DRC	Checagem das regas de projeto *(Design Rules Check*)
DSP	Processadores de sinal digital *(Digital Signal Processors*)
EDA	Automação de Projetos Eletrônicos *(Electronic Design Automation*)
FET	Transistor de efeito de campo (Field Effect Transistor)
FPGA	Matriz de Portas Lógicas Programáveis por Campo (*Field Programmable Gate Array*)
GDSII	Sistema de banco de dados gráfico *(Graphic Data System*)
GUI	Interface gráfica do usuário *(Graphical User Interface*)
HS	Alta velocidade *(High Speed*)
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos *(Institute of Electrical and Electronics Engineers*)
IP	Propriedade intelectual *(Intellectual Propreties*)
LED	Diodo emissor de luz *(Light Emiting Diode*)
LSI	Integração de larga escala *(Large System Integration*)
LPLV	Baixa potência e baixa tensão *(Low Power Low Voltage*)
LVS	Leiaute *versus* esquemático *(Layout Versus Schematic*)
MOSFET	Transistor de efeito de campo Metal-Óxido-Semicondutor *(Metal-Oxide-Semiconductor Field Effect Transistor*)
NRE	Engenharia não recorrente *(Nonrecurring Engineering*)
RAM	Memória de acesso aleatório *(Random Access Memory*)
RF	Rádio Frequência
ROM	Memória Apenas de Leitura *(Read Only Memory*)

RTL	Transferência de dados entre os registradores *(Register Transfer Level*)
SoC	Sistema em um *chip (System on-a-chip*)
ULA	Unidade Lógica Aritmética
USB	Barramento serial universal *(Universal Serial Bus*)
VHDL	VHSIC Linguagem de descrição de *Hardware* (VHSIC *Hardware Description Language*)
VHSIC	Circuito Integrado de Muita Alta Velocidade *(Very High Speed Integrated Circuit*)
VLSI	Sistema de integração muito alta *(Very Large System Integration*)

SUMÁRIO

1 INTRODUÇÃO 16

2 CONCEITOS BÁSICOS 20

2.1 Linguagem VHDL 20

2.1.1 Histórico 20

2.1.2 Aspectos gerais da linguagem 21

2.1.3 Fluxo de projeto de um circuito integrado digital em VHDL 23

2.1.4 Estrutura da linguagem VHDL 26

2.2 Linguagem Verilog 28

2.2.1 Aspectos gerais da linguagem 28

2.2.2 Estrutura da linguagem Verilog 29

2.3 Microcontrolador 8051 29

2.3.1 Introdução 29

2.3.2 Arquitetura Interna do 8051 29

2.4 Dispositivos lógicos programáveis 31

2.5 Conceitos básicos para a realização de um fluxo de projeto para circuitos integrados digitais 33

2.5.1 Simulação 33

3 PROPOSTA DE UM FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS DIGITAIS UTILIZANDO FERRAMENTAS COMPUTACIONAIS EM VERSÃO DE DEMONSTRAÇÃO E PROFISSIONAL 34

3.1 Definição do projeto. 34

3.2 Fluxo utilizando ferramentas computacionais em versão de demonstração 35

3.3 Fluxo utilizando ferramentas computacionais em versão profissional 37

4 PROPOSTA DE UM FLUXO DE PROJETO DE CIS DIGITAIS, UTILIZANDO-SE FERRAMENTAS EM VERSÃO DE DEMONSTRAÇÃO PARA FINS ACADÊMICOS 39

4.1 Especificação de circuito digital denominado de Contador de pulsos de 2 bits (bloco 1 da Figura 9) 42
4.2 Descrição VHDL (bloco 2 da Figura 9) 43
4.3 Compilação do código VHDL do circuito Contador de pulsos de 2 Bits (bloco 3 da Figura 9) 44
4.4 Síntese do circuito contador de pulsos de 2 bits (bloco 3 da Figura 9) 45
4.5 Simulação (bloco 4 da Figura 9) 48
4.6 Conversão para verilog e geração do leiaute (bloco 5 e 6 da Figura 9) 49
4.7 Verificação da geração das máscaras utilizadas pelo *Microwind* (bloco 7 da Figura 3-1) 54
4.8 Simulação Final (bloco 8 da Figura 9) 56
4.9 Conclusão 60

5 IMPLEMENTAÇÃO EM FPGA DE UM MICROCONTROLADOR 8051 A PARTIR DE UM CÓDIGO VHDL 62

5.1 Caracteristicas do VHDL do 8051 62
5.1.1 Criação do código VHDL do microcontrolador 8051 no *software* Quartus II para teste no *Kit* Didático de FPGA, modelo DE2-70, da fabricante Altera 63

6 GERAÇÃO AUTOMÁTICA DO LEIAUTE DO NÚCLEO DO 8051 UTILIZANDO AS FERRAMENTAS DE CAD DA *MENTOR GRAPHICS* 69

6.1 Ferramentas profissionais de leiaute da Mentor Graphics 69
6.2* Sintetização dos arquivos VHDL do núcleo do 8051 utilizando-se o *software Leonardo Spectrum 70
6.3* Criação das células padrão e do esquemático do núcleo do 8051 no *software Pyxis Schematic 73
6.4 Geração automática do leiaute do núcleo do 8051, utilizando-se o *software Pyxis Layout* 75
6.5 Conclusão 83

7 CONCLUSÕES 85
REFERÊNCIAS 87
APÊNDICE A – TUTORIAL DE UTILIZAÇÃO DO SOFTWARE *QUARTUS II* PARA CRIAÇÃO DE BLOCOS VHDL, SINTETIZAÇÃO E SIMULAÇÃO DE PROJETOS DIGITAIS 89

1 INTRODUÇÃO

Nos dias de hoje, quase tudo que nos rodeia, é resultado do avanço tecnológico da eletrônica, que normalmente nos passa despercebido em nível de equipamentos eletrônicos, mas que passou por um grande e longo processo de evolução até chegar aos níveis atuais [1].

Com o passar dos anos, a cada nova descoberta da área da microeletrônica, mudava-se o rumo da tecnologia e consequentemente a dos equipamentos eletrônicos. A década de 60 foi o ponto inicial para o avanço tecnológico com a utilização de componentes semicondutores, que se tornaram primordiais para que tudo isso acontecesse como, por exemplo, o Transistor de Efeito de Campo Metal-Óxido-Semicondutor *(*MOSFET*, Metal-Oxide-Semiconductor Field Effect Transistor)*, os circuitos integrados (CI) e a tecnologia de fabricação de circuitos integrados digitais, utilizando a tecnologia Complementar Metal-Óxido-Semicondutor (CMOS) [1].

Com essas descobertas tornou-se possível a criação de circuitos integrados que revolucionaram a maneira de realizar projetos eletrônicos, como o microprocessador e as memórias, que substituíam diversos circuitos integrados e permitiram ao projetista configurar o funcionamento do projeto através de *software* [1] [2]. Este mercado de microprocessadores movimenta hoje uma grande quantia de dinheiro por ano [2].

Outro ponto que mostra a evolução da microeletrônica é a Lei de Moore, onde Gordon E. Moore, co-fundador da Intel Corporation, líder no mercado de microprocessadores, diz que a cada 18 meses a quantidade de transistores dentro de um circuito integrado dobra. Com o passar do tempo, a lei de Moore foi adotada como meta das grandes empresas do segmento e cada vez que se pensava que não seria possível alcançá-la, pois chegava-se no limite físico da tecnologia, outras ainda mais modernas eram descobertas para que a lei de Moore permanecesse válida [1]. Hoje é possível ter dispositivos integrados com mais de 200 milhões de transistores por cm^2 [2].

Com todo este desenvolvimento da microeletrônica, os equipamentos alcançaram níveis de desempenho muitas vezes superiores à necessidade da maioria dos usuários, os quais trocariam o excesso deste desempenho por confiabilidade, personalização e, principalmente, na diminuição de consumo de energia, parâmetro que se tornou primordial [2]. Especialistas mostram que neste início de século XXI, as pesquisas na área de microeletrônica foram impulsionadas pelo mercado de eletroportáteis, que acarretaram em grandes avanços dos equipamentos eletrônicos de baixa potência e baixa tensão (*low power low voltage*, LPLV),

que mantêm os altos índices de desempenho elétrico, porém com um menor consumo de energia, o que significa, por exemplo, dizer que eles permitem uma maior durabilidade da bateria de um equipamento eletrônico portátil (celular, calculadora, relógios, etc...) [1] [2].

Uma tendência do mercado de eletrônicos é a substituição da maneira de confeccionar projetos, que tende a migrar das maneiras fixas de realização de projeto para as maneiras modulares. Nos projetos realizados de maneira fixa qualquer mudança é complexa, pois envolve mudança de *hardware*. Já nos projetos modulares é possível adequar o projeto eletrônico para o mais próximo da real necessidade [2]. Este método de projeto modular é conhecido como Sistema no Circuito Integrado (*System-on-chip,* SoC), onde a necessidade é baseada na realidade do projeto. Este tipo de projeto, por ser modular, pode usar circuitos integrados já validados por outras empresas, chamados de propriedade intelectual (*intelectual property*, IP). Esse tipo de projeto vem ganhando uma fatia do mercado que antigamente pertencia aos microprocessadores [2].

Existem duas maneiras de se implementar SoCs: através de dispositivos lógicos programáveis ou através de um circuito integrado feito para aquela finalidade. Para a realização do projeto eletrônico através dos dispositivos lógicos, pode ser utilizado o Dispositivo Lógico Programável Complexo (*Complex Programmable Logic Device*, CPLD) ou a Matriz de Portas Lógicas Programáveis por Campo (*Field Programmable Gate Array*, FPGA), que são CIs onde o projetista tem a possibilidade de configurar não somente o *software* da aplicação como, por exemplo, o *firmware* de um microprocessador, mas também o de alterar o seu *hardware* [3]. Já no caso do circuito integrado dedicado ou circuito integrado *full custom,* como é conhecido, é projetado um circuito integrado que é implementado em nível de transistor, através de máscaras específicas em uma lâmina de silício, onde ele se torna fixo e sob medida para uma determinada aplicação [4] [5].

A diferença entre os dois métodos está basicamente definida no tempo de realização do projeto e no custo da implementação [4]. Caso sejam utilizados os dispositivos programáveis para a confecção do projeto digital, este levará um tempo menor, porém com um custo final por equipamento mais elevado, já que um FPGA poderá custar na faixa de 100 a 10.000 dólares [3] [4]. Optando pelo método do circuito integrado *full custom*, este levará um tempo maior para ser projetado, fabricado e também terá um custo maior para sua implementação, porém considerando-se a implementação em grandes quantidades, o custo final é muito inferior se comparado ao primeiro método, portanto, torna sua utilização viável [3] [4].

O mercado de projetos de circuitos integrados está cada vez mais preocupado em ter fluxos de projetos e ferramentas CAD que aumentem a produtividade, projetos com altos

níveis de segurança e a reformulação dos métodos de ensino para formação de recursos humanos [6] [7] [8] [9].

Com o aumento pela procura por projetos dedicados de circuitos integrados digitais foi necessário o desenvolvimento de ferramentas específicas para atender a demanda, chamadas de ferramentas de Automação de Projetos Eletrônicos (*Eletronic Design Automation,* EDA), que auxiliam o projetista na sua realização, pois circuitos integrados que levavam meses para serem desenvolvidos, hoje, com o auxílio destas ferramentas, levam apenas algumas semanas [2] [5] [6].

A segurança da informação tem sido algo que vem ganhando cada vez mais espaço no mercado de projetos de circuitos integrados. Existem diversas formas de aumentar esta segurança, alguns trabalhos demonstram ser possível melhorar a confiabilidade da informação através de uma mudança no fluxo de projeto convencional [7].

Na área de pesquisas acadêmicas, a preocupação com a formação de recursos humanos vem aumentando significativamente. Existe a necessidade de implantação de disciplinas que estimulem os estudantes dos cursos de graduação, relacionados à área, a desenvolverem projetos de circuitos integrados dedicados, com o intuito de garantir a qualidade do aprendizado e a preparação para o mercado de trabalho [8].

Dentro deste contexto, este trabalho visa estudar dois fluxos de projeto de circuito integrado digital utilizando ferramentas EDA em versão de demonstração e em versão de uso profissional, para a realização e confecção de um projeto de Circuito Integrado Digital de Aplicação Específica (*Application Specific Integrated Circuit,* ASIC), detalhando-se todas as etapas do fluxo de projeto.

O primeiro fluxo de projetos estudado utilizou ferramentas em versão de demonstração, que são distribuídas gratuitamente por diferentes fabricantes. Este estudo tem como principal objetivo estimular a formação de recursos humanos na área de microeletrônica, pois através dele é possivel a criação de um CI ASIC sem nenhum investimento na compra de licenças de *software*. Isso dará acesso aos alunos, de instituições de ensino que não possuam essas licenças de *software,* ao desenvolvimento de projetos de circuitos integrados digitais. Para a realização desta proposta foi escolhido um circuito contador de 2 bits, com o intuito de ser um circuito simples.

Após o estudo do fluxo com ferramentas em versão de demonstração, foi estudado um novo fluxo utilizando ferramentas EDA profissionais, que apresentam um desempenho bastante superior as utilizadas anteriormente, porém suas licenças de utilização devem ser adquiridas previamente. Para a utilização de um fluxo de projeto que usa ferramentas

profissionais, que neste caso foi a da *Mentor Graphics*, foi definido como projeto o núcleo do microcontrolador 8051, por ser um circuito mais complexo para verificar a capacidade da ferramenta computacional profissional e por apresentar consolidada utilização no mercado dos microcontroladores de 8 *bits*, ainda hoje fabricado por diversas empresas tais como: Texas Instruments, Philips, Siemens entre outras, inclusive sendo utilizados em aplicações médicas e aeroespaciais [10]. O fluxo do projeto utilizando ferramenta profissional, é formado por sete etapas que serão detalhadas no decorrer desta dissertação, que consiste desde a especificação do projeto até a fabricação do CI, passando por etapas de simulação, testes físicos, implementação automática de leiaute, entre outras, onde a principal vantagem de realizá-lo é a diminuição do tempo de desenvolvimento de projeto dos CIs digitais [5].

2 CONCEITOS BÁSICOS

Este capítulo tem como objetivo explicar alguns conceitos fundamentais da área de projetos de circuitos integrados digitais. Esses conceitos são necessários para o entendimento deste trabalho.

2.1 Linguagem VHDL

Com a crescente expansão do mercado por equipamentos eletrônicos, cada vez mais exigentes, fez com que as técnicas para projetos de circuitos integrados digitais, que dependiam exclusivamente do projetista, fossem perdendo sua utilização, em relação a sua complexidade, pois se fosse necessário fazer uma interpretação futura deste projeto, isso se tornaria algo extremamente trabalhoso e complicado, caso ele não tenha sido documentado de maneira satisfatória [11].

Visando solucionar estes problemas, as empresas de tecnologia, vem cada vez mais, utilizando as linguagens de descrição de hardware (Hardware Description Languages, HDL) para realizar seus projetos de circuitos integrados digitais, devido a facilidade de sua realização e modificações futuras [11].

2.1.1 Histórico

No inicio da década de 80, o Departamento de Defesa dos Estados Unidos da América (*Defense Advanced Research Projects Agency*, DARPA), encontrou um grave problema em sua sistemática de projetos de circuitos integrados, pois ao iniciar novos trabalhos com o intuito de acompanhar o avanço tecnológico, se defrontou com a falta de documentação adequada do projeto anterior, necessitando assim de muito tempo para a análise de cada bloco lógico do projeto, para que assim fosse possível realizar mudanças e melhorias no novo projeto [11]. Porém este não era o único problema, pois o fato de não se ter uma linguagem padronizada para a confecção destes blocos lógicos, fazia com que este retrabalho ficasse ainda mais difícil [11].

Visando solucionar estes problemas em 1981, o DARPA realizou um encontro de especialista da área para a discussão dos métodos utilizados na criação de projetosde circuitos integrados digitais [11]. Após este encontro, ficaram definidos os requisistos que a linguagem de descrição de *hardware* deveria atender, e em 1983 foi concedida a determinadas empresas

de tecnologia de *software* a incumbência do desenvolvimento da linguagem de descrição de *hardware* e das ferramentas para sua utilização [3]. Em 1987, surgia a linguagem de descrição de *Hardware (Very High Speed Integrated Circuit Hardware Description Language,* VHDL*)*, uma linguagem de descrição de *hardware*, que viria a ser padronizada pelo Instituto de Engenheiros Eletricistas e Eletrônicos (*Institute of Eletrical and Eletronic Engineer,* IEEE), como o padrão 1076-1987 [3] [11].

Com o passar dos anos, o VHDL vem sendo cada vez mais utilizado, e sofrendo atualizações e melhorias, resultando em novas versões do padrão da linguagem, que atualmente se intitula IEEE 1076-2008 [12].

2.1.2 Aspectos gerais da linguagem

A linguagem VHDL é uma excelente ferramenta na implemetação de projetos eletrônicos digitais, pois permite ao projetista a realização da descrição do funcionamento de seu projeto em diversos níveis de abstração e hierárquicos, o que estabelece flexibilidade e rapidez ao projeto [11].

A figura 1 ilustra o fluxo de desenvolvimento de projetos de CIs digitais utilizando a linguagem VHDL [11].

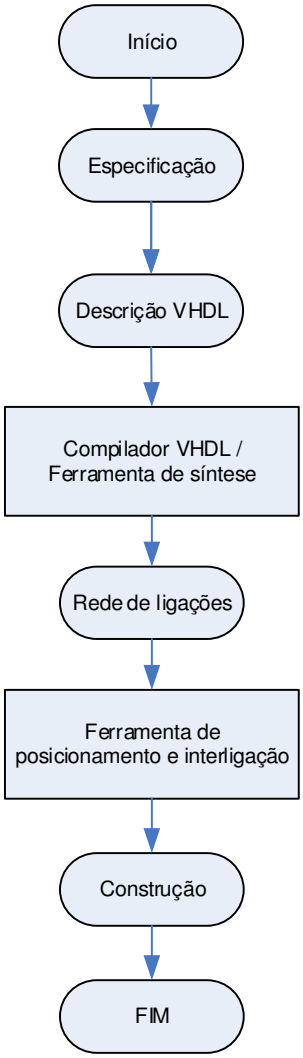


Figura 1- Etapas do desenvolvimento de projetos de CIs digitais utilizando a linguagem VHDL
FONTE: [11].

O fluxo de projeto apresentado na Figura 1 apresenta, de maneira bem simples, as etapas de projetos de circuitos integrado digital. A primeira etapa é a definição das especificações do projeto, onde é definido o seu funcionamento. Posteriormente a esta etapa, temos a descrição VHDL em função das especificações do projeto do circuito integrado digital, que é responsável por transcrever todas as suas funcionalidades. A próxima etapa é a compilação do código VHDL, que tem como função, transformá-lo para que seja implementados por estruturas de *hardwares* específicos (CPLD, FPGA), além de identificar possíveis erros de sintaxe [11]. Com o código VHDL sendo compilado pelo compilador VHDL, o produto desta compilação será usado em um *software* simulador, onde serão analisadas as funcionalidades do projeto digital. No caso em que o código VHDL seja

compilado por uma ferramenta de síntese, o produto desta compilação será utilizado nas próximas etapas do fluxo do projeto [11].

Com o resultado da compilação, realizada pela ferramenta de síntese, tem-se todas as funcionalidades descritas do projeto digtal, representadas em circuitos lógicos (portas lógicas, flip-flops, etc...). Com estes circuitos lógicos prontos e interligados entre si, será realizada a etapa de posicionamento e interligação, que tem como objetivo, dispor todos os circuitos lógicos em um leiaute (conjunto de máscaras para serem construídas num determinado processo de fabricação), visando a otimização do espaço e possibilitando as interligações. Para finalizar o fluxo de projeto, a última etapa do processo (Figura 1), é a sua construção, que recebe o leiaute finalizado na etapa anterior, e o transformará em um circuito integrado digital [11].

A descrição VHDL do projeto pode ser feita de três maneiras basicamente: a descrição estrutural, a comportamental e através do fluxo de dados. O modo estrutural é utilizado para descrever a organização física e a topológica do sistema digital, especificando as entradas e saídas dos componentes lógicos e a interligação entre eles. No modo comportamental, não é necessário descrever a organização física e topológica do sistema, mas as funções do circuito digital, e a descrição através do fluxo de dados, como o próprio nome já diz, será informado o fluxo de dados através dos registradores do hardware. Dessa forma, o projetista tem autonomia de escolher o método de descrição a ser utilizado e ainda se for conveniente, combinar os dois processos [4].

Outro ponto importante da linguagem VHDL é que ela trabalha de maneira concorrente, ou seja, não diferencia a ordem das instruções descritas pelo código VHDL, e a ocorrência de um sinal acarreta na execução de todas as instruções que são sensíveis a este sinal, de uma maneira simultânea, assim como ocorre em um circuito real [11]. Porém caso seja necessário, pode-se definir determinados trechos da linguagem VHDL, que trabalha de forma sequencial [11].

2.1.3 Fluxo de projeto de um circuito integrado digital em VHDL

Quando utilizamos o VHDL como linguagem de descrição do projeto de circuitos integrados digitais, é interessante seguir determinados passos, conforme ilustrado na Figura 2, para que se tenha uma maior confiabilidade na realização do projeto [4].

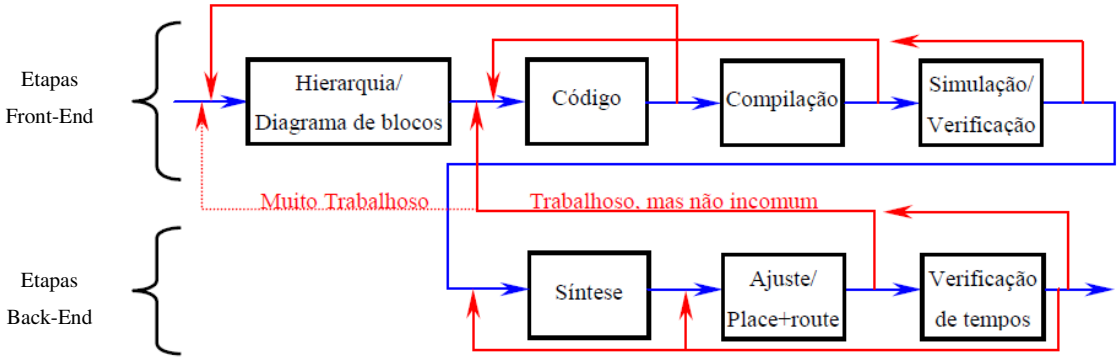


Figura 2- Fluxo de um Projeto digital utilizando a linguagem VHDL
FONTE: (4,p.34).

O fluxo do projeto se divide em duas etapas, a *"front-end"* e a *"back-end"*. A etapa *front-end* é iniciada com a definição da hierarquia e da arquitetura do circuito integrado digital, a partir das necessidades impostas ao projeto digital. Após essas definições, cria-se os códigos VHDL de cada um dos blocos do projeto do circuito integrado digital. Nesta etapa, como a linguagem VHDL é baseada em textos, pode se usar qualquer editor de texto para sua criação, porém nos dias atuais, com as ferramentas de Projeto Assistido por Computador (C*omputer-Aided Design,* CAD), o trabalho se torna muito mais fácil, pois tem-se diversos recursos, tais como: facilidade da descrição do código com geração de padrões pré-definidos (templates), destaque de palavras chaves, correção de pequenos erros, entre outras facilidades oferecidas. Outro ponto importante é que normalmente, essas ferramentas podem ser utilizadas nas próximas etapas do processo, realizando a compilação, a simulação e a síntese [4] [11].

Hoje no mercado, existem diversas empresas que disponibilizam estes ambientes de projeto de circuitos integrados digitais, tais como: Altera (Quartus II), que possui versões pagas e gratuitas, Xilinx (ISE); Mentor Graphics (FPGA Advantage) e Cadence (RTL compiler) [3].

Após o código VHDL ter sido escrito, é necessário realizar a compilação desse código, pois é nesta etapa que o compilador analisará possíveis erros de sintaxe, de relações entre blocos existentes, entre outros. Enquanto todos os erros apresentados pelo compilador não forem corrigidos, não é possível avançar para a próxima etapa, por isso é interessante não deixar para compilar o código VHDL todo, de uma só vez, com o intuito de não permitir que os erros se proliferem [4].

Após o término da compilação, o próximo passo é a simulação do circuito integrado digital, onde o projetista testa de maneira virtual (em computador), se o código VHDL descrito atende as especificações iniciais do projeto digital. Essas simulações são principalmente em nível funcional, onde o projetista aplica sinais de entrada e analisa os sinais de saída do seu projeto. Este procedimento pode ser realizado de maneira manual, se a quantidade de variáveis do seu sistema for pequena, e de forma automática, se a quantidade de variáveis for grande, são os chamados *test benches*, onde um código VHDL, específico para cada projeto digital, manipula os sinais de entrada e analisa as respostas, obtendo-se assim o mesmo processo da verificação [4].

Como já foi dito anteriormente, a simulação está baseada principalmente na verificação funcional do sistema, porém outro ponto importante a ser verificado, são os tempos de propagação e os atrasos dos sinais ao longo do circuito digital, pois pode comprometer o funcionamento dos circuitos, principalmente se forem sequenciais, tais como os circuitos digitais que utilizam *flip-flops* [4].

Estes processos de simulações são muito importantes, pois caso algum erro passe despercebido e seja encontrado posteriomente, isso significará a necessidade de refazer as etapas anteriores novamente, o que ocasionará um gasto maior de tempo. Após a conclusão da simulação, a próxima etapa é a síntese do circuito digital, onde o seu código VHDL será transformado em circuitos compostos de portas lógicas ou circuitos digitais primitivos (exemplos, *flip-flops*), onde também será gerado um arquivo *netlist,* que descreve todas as interligações existentes do projeto digital [4].

O arquivo *netlist,* na etapa de ajuste, será analisado de modo a fazer um levantamento das necessidades de portas lógicas adicionais para possíveis correções de tempo de propagação de sinais e atrasos. Após essa etapa, será definido o leiaute padrão para cada tipo de sub-circuito, a fim de possibilitar as interligações entre eles, onde esta atividade é conhecida como *place and route*, cujo objetivo é a implementação do leiaute do circuito integrado, onde se define o posicionamento de cada bloco pertencente ao circuito digital e as interconexões entre eles [4].

A partir do leiaute já implementado, realiza-se novamente uma simulação, levando-se em conta todo o funcionamento do projeto digital, principalmente quanto aos aspectos de propagação do sinal ao longo do circuito digital. Após essa etapa, se os resultados não atenderem a todas as especificações exigidas inicialmente pelo projeto, é necessário identificar qual etapa do fluxo do projeto digital falhou, devendo-se executar as devidas

correções e repetir todo o fluxo de desenvolvimento de projetos de circuitos integrados digitais novamente [4].

2.1.4 Estrutura da linguagem VHDL

A linguagem VHDL, como qualquer outra linguagem estruturada, tem como princípio, definir a funcionalidade do projeto digital. Essa linguagem é baseada na linguagem de programação PASCAL. A estrutura básica de uma programação utilizando a linguagem VHDL pode ser dividida em: bibliotecas, entidade e arquitetura, conforme é mostrado na Figura 3 [4].

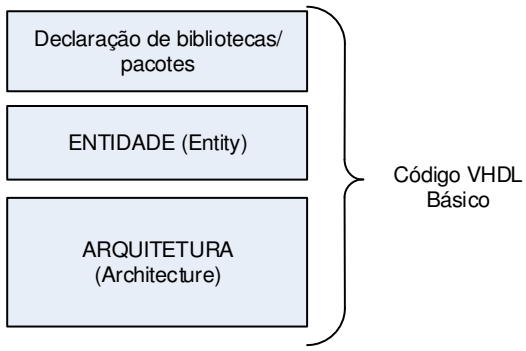
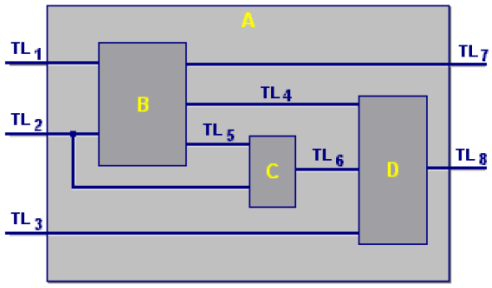


Figura 3- Estrututa básica de um código escrito em VHDL.
FONTE: (3, p.433).

As bibliotecas normalmente consistem de todas as informações que o compilador necessita para processar o projeto digital [3]. Existem bibliotecas que normalmente já são definidas, tais como a STD e a WORK, e caso seja necessário incluir bibliotecas específicas, basta declará-las no código VHDL [3].

A entidade (*entity*) é a parte do projeto que mostra ao compilador VHDL, como são as interfaces do projeto digital (entradas e saídas). Para ilustrar, a Figura 4a existe um projeto digital chamado de "A", que recebe três sinais de entrada (TL_1, TL_2, TL_3) e possui dois sinais de saídas (TL_7, TL_8). Além disso, internamente pode-se observar os sinais TL_4, TL_5, TL_6, e os subcircuitos B, C e D [11]. No entanto para fim da codificação em VHDL da entidade do sistema digital em questão, os únicos sinais que são relevantes, são os que fazem interface com o mundo externo (TL_1, TL_2, TL_3, TL_7, TL_8) e os demais serão tratados posteriormente [13].



(a)

```
entity A is
port (
        TL1: IN bit;
        TL2: IN bit;
        TL3: IN bit;
        TL7: OUT bit;
        TL8: OUT bit;
    );
End entity A
```

(b)

Figura 4- Entidade do sistema A constituída pelos subsistemas B, C e D no qual sua representação é dada em diagrama de blocos (a) e sua codificação em linguagem VHDL da entidade do sistema A (b).

FONTE: Autor "Adaptada de [13]".

Observe na Figura 4.b, que a entidade começa com a declaração inicial *"entity"*, seguida do nome que a identifica, nesse caso "A". Depois de declarada a entidade, é necessário declarar as entradas e saídas do sistema, através da diretiva "*port*". Nesta diretiva são listados todos os sinais de entrada e saída e suas possíveis direções (a direção do sinal indica se o sinal é somente de entrada, saída ou entrada e saída) e os respectivos tipos de sinais (bit, byte, etc...). Note que existem três sinais de entrada especificados através da diretiva "*IN*" e dois sinais que são saídas definidos pela diretiva "*OUT*". Além destes dois modos, pode se ter ainda o modo "INOUT", onde o sinal pode ser definido de forma bidirecional (Entrada e Saída) e o BUFFER, que se trata de uma saída que pode ser referenciada internamente [11].

Depois de configurados os pinos do sistema e quais as suas direções, é necessário definir o tipo de sinal que será manipulado. Existem diversos tipos de sinais possíveis, desde números inteiros, até matrizes de caracter, onde neste exemplo foi utilizado o mais simples, onde o sinal aceita somente dois estados, o "0" e o "1". Para eles serem declarados desta maneira, foi utilizado o comando "BIT" [11].

O último item da estrutura básica de um programa VHDL é a sua arquitetura, também conhecida como corpo da programação, que define toda a funcionalidade do sistema digital, ou seja, é onde os sinais declarados na entidade são processados pelo circuito digital [13].

Para que haja uma arquitetura, é necessário ter uma entidade. Estes dois blocos do código VHDL são indispensáveis para o pleno funcionamento do sistema digital [11].

Para se declarar uma arquiterura, utiliza-se o comando "*architecture of entity"*, onde é dado um nome a ele, que está relacionado à entidade. Além disso, é possível ter mais de uma arquitetura para a mesma entidade [13].

Após ser declarada a arquitetura do programa, pode se dar inicio ao tratamento dos sinais, tais como a realização das declarações de constantes, declaração de subprogramas, entre outras possibilidades. Os comandos "*BEGIN*" e "*END*" são utilizados para delimitar o começo e o fim da estrutura do programa, onde nesta região todos os sinais serão tratados de forma concorrente, como já dito anteriormente [11].

2.2 Linguagem Verilog

A linguagem Verilog HDL, ou simplesmente Verilog, é também uma linguagem de descrição de hardware muito utilizada pelos projetistas de circuitos integrados digitais. Introduzida no mercado em 1984 pela empresa *Gateway Design Automation,* tornou-se popular apenas em 1989*,* quando a empresa *Cadence Design System* adquiriu a empresa *Gateway*.

Nos dias atuais, as ferramentas mais populares utilizadas na síntese lógica de circuitos digitais permitem ao projetista escolher qual linguagem HDL ele irá utilizar, fazendo com que o mercado se divida praticamente na metade entre a linguagem VHDL e Verilog.

2.2.1 Aspectos gerais da linguagem

A linguagem Verilog permite que o projeto seja realizado nos seguintes níveis de abstração:

Comportamental – este é o nível mais elevado de abstração, onde é descrito um algoritmo do funcionamento do sistema, sem se preocupar com sua implementação;

Fluxo de dados – neste nível é informado como os dados fluem entre os registradores de hardware e como os dados são processados;

Portas – neste nível o projetista irá descrever o circuito através de portas lógicas e suas interconexões;

Chaves – este é o nível mais baixo de abstração, onde as funcionalidades dos circuitos serão implementadas descrevendo o circuito em chaves e suas interconexões.

O verilog permite que o mesmo projeto tenha módulos criados com níveis de abstração diferentes, quanto mais alto for o nível de abstração mais flexível irá se tornar o projeto.

2.2.2 Estrutura da linguagem Verilog

A linguagem Verilog baseia sua estrutura de programação em módulos, onde cada módulo terá sua função e se relacionará com o restante do projeto através de seus *ports* (entradas e saídas), isso ocorre independente da maneira que foi construído o módulo, o que permite que o projetista possa alterar a construção interna de um módulo de maneira independente, sem interferir nas funcionalidades dos outros módulos.

Para se definir um módulo em Verilog é utilizada a palavra *module*, cada modulo terá em sua descrição um *module_name* que identifica o nome do módulo e um *module_terminal_list* que define as entradas e saídas do módulo. Para finalizar o módulo é utilizada a descrição *endmodule.*

2.3 Microcontrolador 8051

2.3.1 Introdução

Na década de 80, a Intel criou uma família de microcontrolador, denominada 8051, que seria a herdeira do então microcontrolador 8048 [14]. No início, existiam basicamente três modelos desta família, onde a principal diferença estava na memória de programa, isto é, o "8051", que vinha com Memória Apenas de Leitura (*Read Only Memory,* ROM) interna programada de fábrica (ROM máscara), o "8751" implementado com Memória Programável Apagável de Apenas de Leitura (*Erasable Programmable Read Only Memory,* EPROM), onde o usuário poderia programá-la com auxílio de um programador de EPROM e o "8031", que necessitava de uma memória de programa externa. Com o passar do tempo, está família foi evoluíndo e ganhando novas versões, como por exemplo, o 8052, que tem um temporizador interno a mais, entre outras versões, tais como, aqueles que possuem conversor A/D e saídas PWM, etc [14].

Hoje esta família é denominda MCS-51 e algumas empresas como Siemens, Philips entre outras, fabricam seus próprios microcontroladores com o núcleo do 8051 [14].

2.3.2 Arquitetura Interna do 8051

Arquitetura interna do microcontrolador 8051 é apresentada na Figura 5.

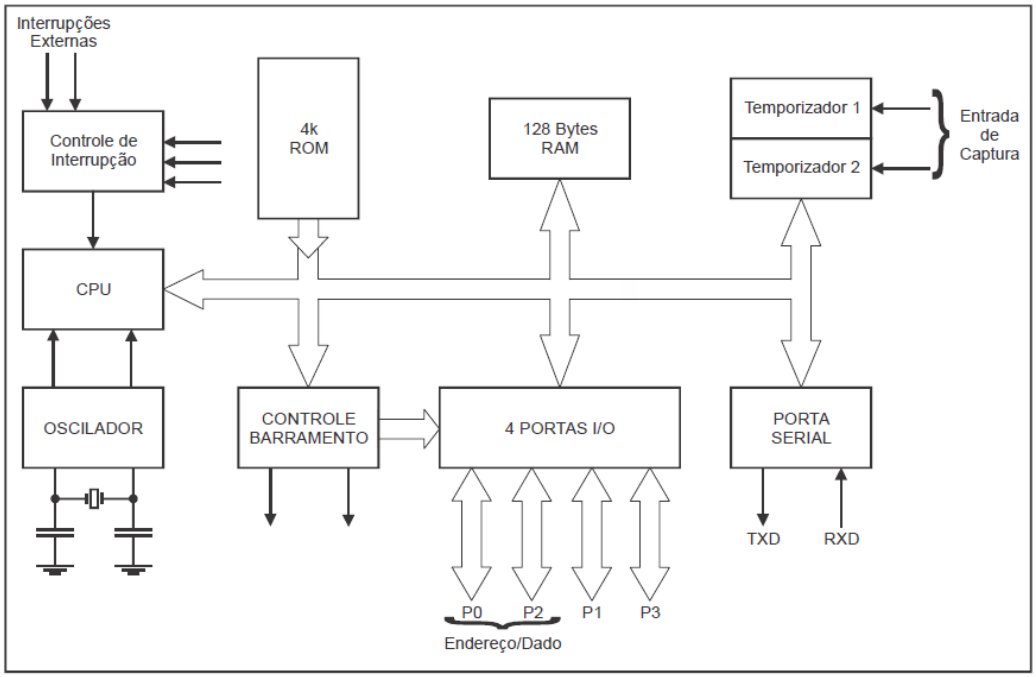


Figura 5- Arquitetura interna do microcontrolador 8051 da Intel.
FONTE: [15].

Na Figura 5, TXD é o pino de transmissão de dados da porta serial, RXD é o pino de recepção de dados da porta serial, os pinos P0, P1, P2, P3 são as portas de entrada e saída do 8051, o bloco "Controle Barramento" é o responsável pelo controle do fluxo de informações, quando se utiliza memória de programa externa, o bloco "oscilador" é responsável pela geração do sinal de *clock* para o microntrolador, a "CPU" é o bloco principal do microcontrolador, o qual comanda todos os outros periféricos, o controle de interrupção é o bloco que coordena o acontecimento de algum evento que deve ser tratado como uma interrupção, os dois próximos blocos são os blocos de memória de programa e de dados, onde a memória de programa é de 4K*Bytes* e a memória de dados é de 128 *Bytes,* e por fim os blocos temporizador, que são responsáveis por gerar tempo [14].

Como apresentado na Figura 5, todos os blocos periféricos trabalham conforme as designações da Unidade Central de Processamento (*Central Processing Unit,* CPU), que na família MCS-51, trabalha com a capacidade de interpretação de códigos de 8 *bits*, os quais foram definidos pelo usuário, através de uma linguagem de programação e gravada na memória de programa. Todos estes códigos serão interpretados pela CPU e tais instruções são executadas por ela. Este processo de interpretação e execução de instruções é algo cíclico dentro microcontrolador [10]. O microcontrolador 8051 possui 4 K*bytes* de memória não volátil, que tem como função principal armazenar o programa que define a funcionalidade de

um determinado projeto e possui também 128 *bytes* de Memória de Acesso Aleatório (*Random Access Memory,* RAM), que é utilizada para manipulação de dados. Adicionalmente, o 8051 tem capacidade de endereçar externamente até 64K*bytes* de memória de programas e de dados, simultaneamente [15].

O 8051 conta com 32 pinos de entrada/saída (input/output, I/O), que estão divididos em 4 portas (P0,P1,P2 e P3), que tem como função realizar a interface com o mundo externo. Estes pinos são bidirecionais, ou seja, podem ser tratados ou como uma entrada ou como uma saída, e suas configurações são realizadas via *software*. Quando o microcontrolador trabalha com memória externa, os portes P0 e P2 são usados como barramento de endereço e dados, sendo utilizados unicamente para esse fim [10].

Existem outros periféricos importantes, como a da porta serial, que realiza comunicação de dados (Transmissor/Receptor Universal Assíncrono, UART) e dois temporizadores internos de 16 *bits,* que são responsáveis pela geração de tempo, conforme a necessidade do sistema, com a vantagem que estes contadores operam por interrupção, que nada mais são do que sinais elétricos, que foram gerados via *hardware,* sinalizam para o 8051, que o tempo programado foi atingido. Quando isto acontece, o sistema finaliza a execução da instrução, que esta sendo realizada no programa principal e desvia para um determinado endereço de memória, onde é executado uma sub-rotina de atendimento a essa interrupção, conforme definido pelo usuário. Após isso, o microprocessador retoma para o programa principal apontando para uma instrução após onde o sinal de interrupção foi detectado pelo microprocessador. Caso não se queira tratar uma interrupção é possível desabilitá-la por software. A família 8051 possui normalmente 5 entradas de interrupções, são elas: a dos dois timers/contadores, duas interrupções externas e uma da comunicação serial de dados [10].

2.4 Dispositivos lógicos programáveis

Ao realizar um projeto digital, uma das etapas iniciais, é a escolha da tecnologia a ser utilizada, pois isto influenciará em todo o fluxo do projeto digital. Hoje os métodos mais utilizados são classificados pela sua programabilidade. Pode-se realizar um projeto digital com circuitos não programáveis (utilizando circuitos integrados de portas lógicas, contadores, etc...), circuitos programáveis (microcontroladores) e com o *hardware* programável (CPLD e FPGA). Os circuitos não programáveis, consistem em um circuito integrado de estrutura fixa,

onde não é possível ser alterado com o auxílio de *software*. Esse tipo de circuito integrado foram os primeiros a serem utilizados em projetos de circuito digitais. Os circuitos programáveis também possuem estruturas fixas, porém as tarefas que eles executam podem ser programadas via *software*, dando assim uma maior flexibilidade aos projetos digitais, pois caso seja necessário alterar o seu funcionamento, é possível fazer isso via *software* (microprocessadores e microcontroladores). Os circuitos integrados com *hardware* programável, são os mais recentes e apresentam a maior flexibilização do projeto, pois sua estrutura física não é fixa, ou seja, ela é programável, onde o desenvolvedor tem autonomia para configurar o *hardware* do circuito integrado através de uma código de configuração de *hadware* (CPLD e FPGA). Este último tipo de circuito permite a realização de sistemas muito complexos [3].

A tecnologia de dispositivos com *hardware* programáveis teve início em meados de 1970, onde os primeiros dispositivos deste tipo se chamavam Matriz Lógica Programável (*Programmable Array Logic*, PAL) e eram compostos somente de portas lógicas convencionais e tinham como principal utilização a confecção de circuitos combinacionais. Melhorias foram implementadas, isto é, ocorreu a inclusão dos *flip-flops,* a fim de permitir a construção de circuitos sequenciais simples. Estes novos dispositivos foram denominados de Dispositivos Lógicos Programáveis (*Programmable Logic Device*, PLD). Na década 80, os PLDs ganharam novos integrantes em sua estrutura e foram realizadas novas conexões internas a fim de oferecer maior flexibilidade. Vários destes PLDs foram construídos em um único circuito integrado, com técnicas sofisticadas de roteamento e diversos tipos de pinos de entradas e saídas. Este tipo de tecnologia ficou conhecida como PLD complexo (Complex PLD, CPLD). Este tipo de dispositivo é muito popular pelo seu alto desempenho e baixo custo [3].

Na década de 80, a Xilinx (fabricante de componentes) desenvolveu um sistema similar ao CPLD, isto é, a Matriz de Portas Lógicas Programáveis por Campo (*Field Programmable Gate Array*, FPGA). Estes dispositivos diferem entre si pela suas arquiteturas, características embutidas e custos. O FPGA tem como alvo principal os sistemas de altíssimas complexidade e de elevado desempenho. Uma outra diferença está relacionada a memórias dos dispositivos, ou seja, o CPLD trabalha com memória não volátil para armazenar a configuração de suas interconexões, e o FPGA trabalha com memórias voláteis, o que implica na necessidade de uma memória externa, não volátil, para armazenar suas interconexões [3].

2.5 Conceitos básicos para a realização de um fluxo de projeto para circuitos integrados digitais

2.5.1 Simulação

O projeto de um circuito integrado digital demanda tempo e dinheiro e uma vez que o circuito foi implementado é praticamente impossível realizar qualquer alteração ou depurá-lo, atuando em suas conexões internas. Qualquer mudança resulta na fabricação de um novo circuito integrado. Para que o projetista tenha certeza que o seu circuito, ao ser implementado, irá funcionar, ele conta com o auxílio da simulação, que nada mais é, que um procedimento de teste usado para verificar se o circuito proposto realiza as funções que foram definidas na concepção do projeto. Com esta etapa, o projetista consegue encontrar possíveis falhas no circuito integrado.

Nos projetos de circuitos integrados digitais são utilizados três tipos de simulação: a simulação lógica, a simulação de tempo e a simulção elétrica.

A simulação lógica é responsável por verificar o funcionamento do circuito, sem levar em consideração os tempos de propagação dos sinais elétricos. Ela visa principalmente verificar as lógicas booleanas e as máquinas de estado lógico do circuito. Analisando também o comportamento do circuito a nível de eventos.

A simulação de tempo tem por função analisar a condição lógica do circuito, levando-se em consideração as propagações dos sinais e seus devidos atrasos.

Finalmente, a simulação elétrica tem por objetivo analisar o comportamento de cada componente como dos transistores, dos capacitores e dos resistores do circuito levando em consideração os níveis de tensões, a propagação do sinal entre outras grandezas, fazendo com que essa seja o tipo de simulação mais próximo da realidade.

3 PROPOSTA DE UM FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS DIGITAIS UTILIZANDO FERRAMENTAS COMPUTACIONAIS EM VERSÃO DE DEMONSTRAÇÃO E PROFISSIONAL

Este capítulo tem como objetivo mostrar as principais características das propostas de fluxo de projeto utilizando ferramentas computacionais em versão de demonstração e profissional.

O fluxo analisado está dividido em três etapas: a etapa básica (utilizada em ambas as propostas), a etapa utilizando as ferramentas em versão de demonstração e a etapa utilizando as ferramentas em versão profissional.

3.1 Definição do projeto

A Figura 6 ilustra o fluxo de projeto de circuitos integrados digitais proposto nesse trabalho.

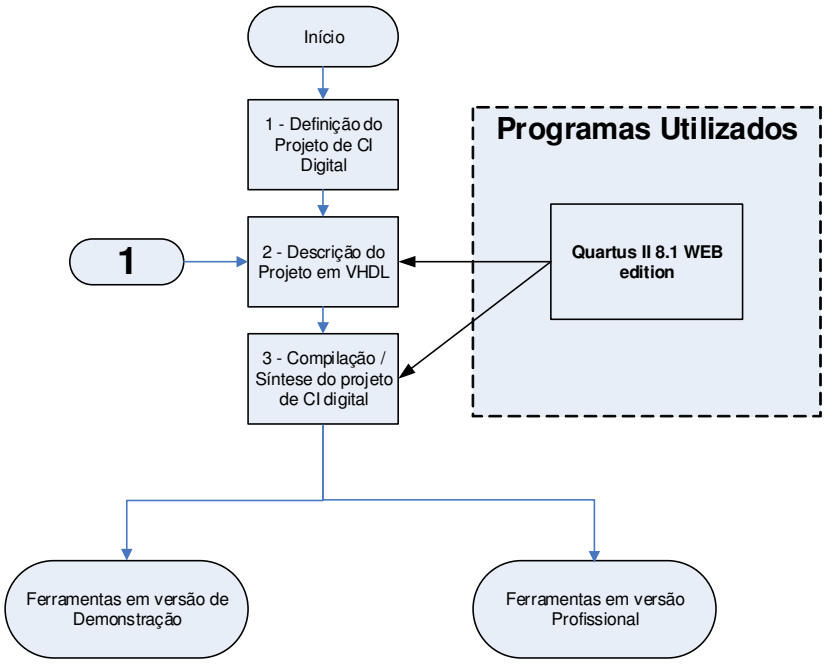


Figura 6- Etapa inicial do fluxo de desenvolvimento

Analisando a Figura 6, observa-se que o fluxo de circuito integrado digital inicia-se com a definição do projeto (bloco 1 da Figura 6). Este passo consiste basicamente na

definição da ideia geral do projeto, onde é definido quais são as suas funcionalidades e qual é sua aplicação.

Em um projeto comercial esta definição é criada após uma pesquisa de mercado realizada pelo departamento de marketing, que identificará uma necessidade. Essa informação será encaminhada ao departamento de engenharia produto que definirá as funcionalidades que o projeto deve apresentar para que possa atender a demanda [10].

O passo seguinte é a descrição do projeto (bloco 2 da Figura 6), onde as definições do projeto são descritas utilizando uma linguagem de *hardware* (VHDL ou Verilog). Neste momento o projetista tem liberdade de realizar a programação utilizando o grau de abstração que lhe for conveniente. Posteriomente, caso a descrição não atenda plenamente as especificações do projeto, o projetista deve retomar esta etapa, detectando as falhas de programação e realizando as devidas correções.

O último passo desta etapa básica do fluxo é a compilação e síntese. A compilação consiste numa análise realizada pela ferramenta computancional, neste caso a ferramenta *Quartus II* versão 8.1, que verifica a sintaxe utilizada na descrição de hardware e converte este código em uma linguagem de máquina que será utilizada na simulação do circuito. Já a síntese consiste na transformação deste código HDL em um circuito que realize todas as funções definidas na descrição do projeto.

Depois de realizados todos esses passos o projetista tem duas possibilidade de continuar o projeto: através das ferramentas computacionais em versão de demonstração ou através das ferramentas computacionais em versão profissional, cada versão apresenta suas particularidades conforme será apresentado.

3.2 Fluxo utilizando ferramentas computacionais em versão de demonstração

Considerando o fluxo mostrado na Figura 6, caso o projetista opte por seguir a proposta utilizando ferramentas computacionais em versão de demonstração, o mesmo seguirá o fluxo apresentado na Figura 7, esta parte específica do fluxo utiliza somente ferramentas em versão de demonstração, que são disponibilizadas de maneira gratuita.

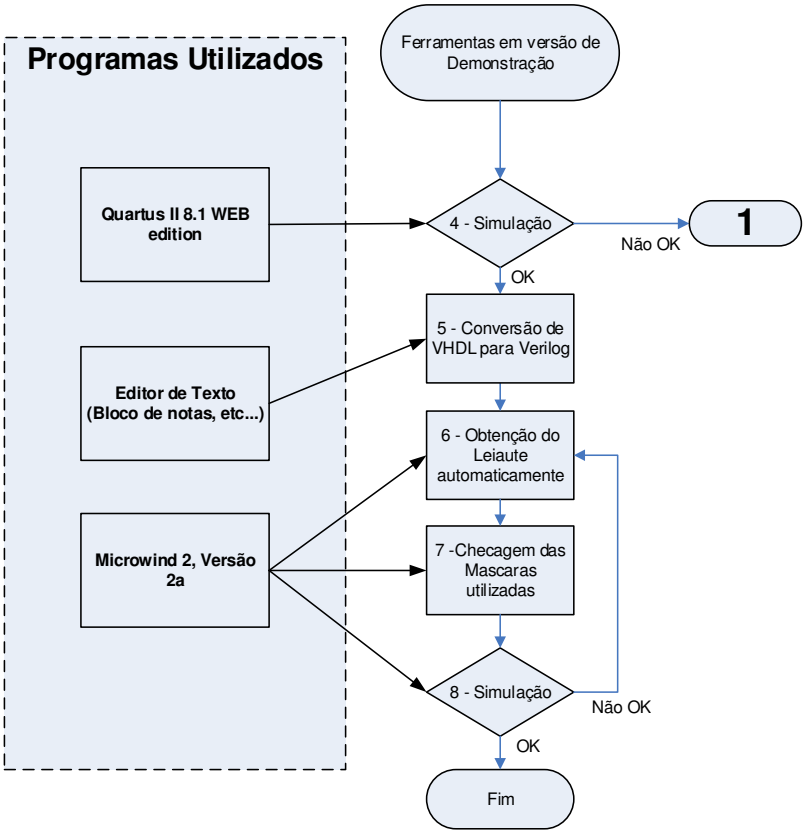


Figura 7- Parte do fluxo utilizando ferramentas em versão de demonstração

O primeiro passo desta parte do fluxo é a simulação (bloco 4 da Figura 7), que realiza os testes funcionais e de tempo no circuito proposto, caso seja detectada alguma falha nesse passo o projetista deve retornar ao passo de descrição do projeto (bloco 2 da Figura 6) e realizar as devidas correções.

Caso a simulação seja realizada de maneira satisfatória, inicia-se a conversão do VHDL para Verilog (bloco 5 da Figura 7). Esse passo é necessário porque o software que será utilizado para geração do leiaute, o *Microwind II*, só aceita como arquivo de entrada, para criação do leiaute, a linguagem Verilog.

Realizada a conversão inicia-se a etapa de geração de leiaute de maneira automática. O software *Microwind* receberá o arquivo Verilog e converterá em um leiaute, de acordo com a tecnologia escolhida.

Após o leiaute ser gerado, inicia-se o sexto passo do fluxo de projeto de circuitos integrados digitais, que consiste na verificação das máscaras implementadas. Esta verificação certifica que as todas as máscaras utilizadas na geração do leiaute fazem parte da tecnologia escolhida.

E por fim é realizada a simulação pós-leiaute. Esta última simulação verificará se todos os passos do fluxo foram realizados com sucesso, caso a simulação não seja adequada ao que se espera do projeto, é necessário retornar os passos anteriores para detectar o erro.

Atendidas as especificações do projeto na simulação, o leiaute é convertido para um formato de arquivo GDSII, e o mesmo é enviado para as empresas que realizarão a confecção do circuito integrado.

3.3 Fluxo utilizando ferramentas computacionais em versão profissional

Ainda considerando o fluxo mostrado na Figura 6, caso o projetista opte por seguir a proposta utilizando as ferramentas profissionais, deve-se seguir o fluxo apresentado na Figura 8.

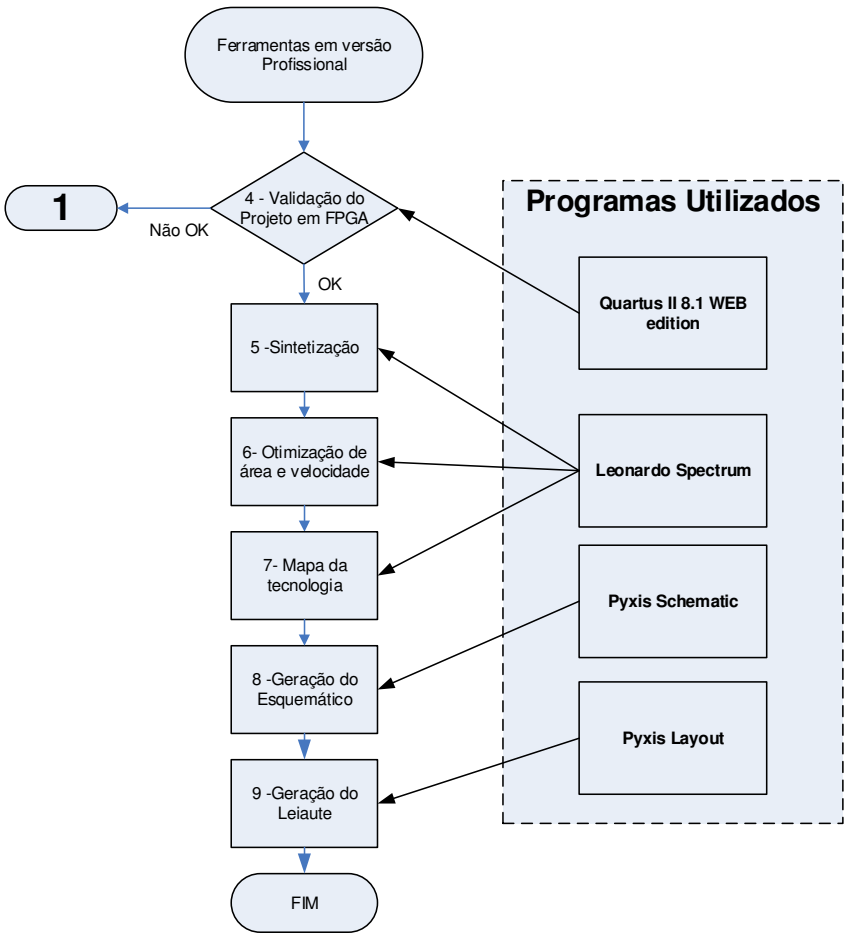


Figura 8- Fluxo utilizando ferramentas profissionais

O primeiro passo nesta etapa, utilizando ferramentas profissionais, é a validação do VHDL (bloco 4 da Figura 8), que tem como finalidade certificar que o código VHDL apresenta todas as funcionalidades existentes na definição do projeto.

Após a validação do VHDL, inicia-se o passo de síntese, que consiste, da mesma maneira que nas ferramentas em versão de demonstração, na criação de um circuito eletrônico que descreva as funcionalidades do projeto.

O próximo passo é o processo de otimização, que visa aumentar o desempenho do circuito ou ainda reduzir as células utilizadas para uma melhor utilização de área de silício na implementação física.

Executada a otimização pelo *software*, realiza-se a criação do mapa da tecnologia, onde o circuito resultante será transcrito para uma linguagem Verilog que será utilizada no próximo passo, a geração do esquemático.

A geração do esquemático consiste em criar um esquemático geral do circuito através do arquivo em Verilog resultante do passo anterior, onde é possível obter todas as conexões do circuito. A partir do esquemático será gerado o leiaute do circuito, esse processo é conhecido como *Schematic Driven Layout* (Leiaute Guiado pelo Esquemático).

Com a geração do leiaute concluída, o arquivo é convertido para o formato GDSII, que é o formato utilizado pelas empresas que realizam a confecção do circuito integrado.

4 PROPOSTA DE UM FLUXO DE PROJETO DE CIS DIGITAIS, UTILIZANDO-SE FERRAMENTAS EM VERSÃO DE DEMONSTRAÇÃO PARA FINS ACADÊMICOS

Neste capítulo é apresentado uma proposta de fluxo de projeto de CIs digitais de forma semi-automática, utilizando *softwares* em suas versões de demonstração, com o intuito da formação de recursos humanos em instituições de ensino que não possuam as ferramentas de desenvolvimento de projetos profissionais de circuitos integrados digitais, devido ao seu alto custo.

Para a validação da proposta apresentada, será realizado um projeto de um circuito integrado digital denominado de contador de pulsos de 2 bits, desde a sua concepção até a geração semi-automática do leiaute para, posteriormente, ser manufaturado.

Para a realização deste projeto foi utilizado um fluxo de desenvolvimento de circuitos integrados digitais conforme mostrado na Figura 9. Nesta figura também são apresentados os programas que foram utilizados para cada etapa do processo.

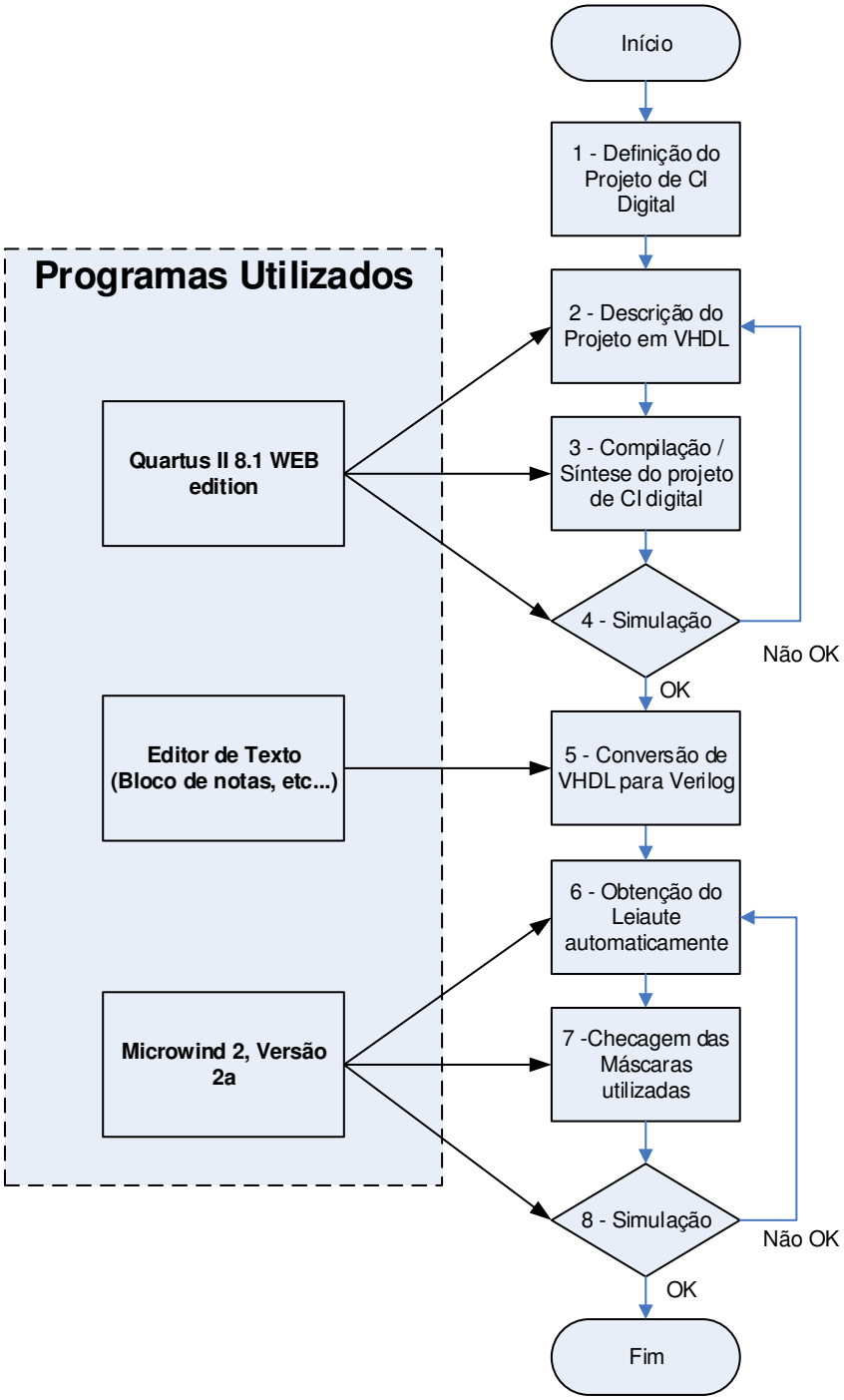


Figura 9- Fluxograma de projeto de um circuito integrado digital denominado de circuito contador de pulsos de 2 bits

FONTE: Autor.

O fluxo se inicia com a definição do projeto (bloco 1 da Figura 9), onde é definada a funcionalidade do circuito. Após esta etapa, cria-se a descrição VHDL (bloco 2 da Figura 9), onde as operações do circuito são colocadas em forma de um código-fonte VHDL. Depois de todas as funcionalidades implementadas no código VHDL, ele deve ser compilado e

posteriormente deve passar pelo processo de síntese (bloco 3 da Figura 9). Durante o processo de compilação os erros de sintaxe da codificação VHDL são encontrados e devem ser corrigidos.

Com todos os possíveis erros sanados é possível passar para próxima etapa do processo, que se refere a síntese, ou seja, é realizada uma tradução do código-fonte VHDL para um circuito eletrônico, que tem como função reproduzir fielmente as características do código VHDL implementado. Após estas etapas, são executadas basicamente pelo *software* Quartus II, inicia–se a primeira etapa de simulação (bloco 4 da Figura 9) que tem como função realizar uma verificação das etapas anteriores. Caso a simulação não esteja de acordo com a especificação do projeto digital é necessário retornar a etapa de descrição VHDL e repetir todas as etapas posteriores a ela, até que a simulação torne-se adequada.

Neste fluxo de projeto a ferramenta de síntese (Quartus II) gera uma expressão boolena e um diagrama de bloco, que apresenta os circuitos lógicos necessários para que o projeto do contador de 2 bits seja implementado em *hardware*. Após esta etapa deve-se converter estes circuitos lógicos em um leiaute. Neste primeiro momento, a ferramenta de leiaute que é utilizada é o *Microwind* [16]. Este *software* permite a criação do leiaute de dois modos diferentes, basicamente: o modo manual e o modo automático. No método manual, todas as máscaras dos transistores e suas interligações são realizadas pelo projetista/leiautista. Para este projeto do contador de 2 bits o modo manual se torna inviável, pois o número de transistores a serem implementados é muito grande. Já no modo automático, o leiaute é criado através de uma descrição de *hardware* que é feita pelo projetista, utilizando-se a linguagem de programação Verilog, que resulta na diminuição do tempo de projeto.

Definido o modo de geração que será utilizado, neste caso o modo automático, foi extraído do processo de síntese uma expressão booleana, que por sua vez, foi descrita em linguagem Verilog (bloco 5 da Figura 9) através de um processo manual de edição. Após esta edição do código Verilog, ele foi inserido no *Microwind* que, através de regras de projeto já especificadas, tem a capacidade de gerar o leiaute (bloco 6 da Figura 9).

Depois de gerado o leiaute, inicia-se a verificação das máscaras utilizadas (bloco 7 da Figura 9), onde é possível visualizar as máscaras que foram utilizadas na geração do leiaute para checar a sua conformidade com a tecnologia utilizada. E a última etapa do fluxo de projeto é a simulação final (bloco 7 da Figura 9), onde é verificado se o leiaute gerado é capaz de atender às especificações do projeto. Caso ocorra alguma falha nesta simulação deve-se voltar ao início do fluxograma do projeto apresentado na Figura 9, e analisar os possíveis

erros, sendo algumas vezes necessário mudar a descrição Verilog e realizar novamente o leiaute. Quando a simulação estiver adequada ao projeto, estará finalizado.

4.1 Especificação de circuito digital denominado de Contador de pulsos de 2 bits (bloco 1 da Figura 9)

O intuito deste projeto é realizar um circuito integrado digital capaz de contar a quantidade de pulsos em dois bits de um sinal de *clock*. Na Figura 10, pode-se observar uma carta de tempo que define o funcionamento do contador de dois bits, que mostra a relação entre os sinais de entrada (*clock e reset)* com as saídas (Q0 e Q1).

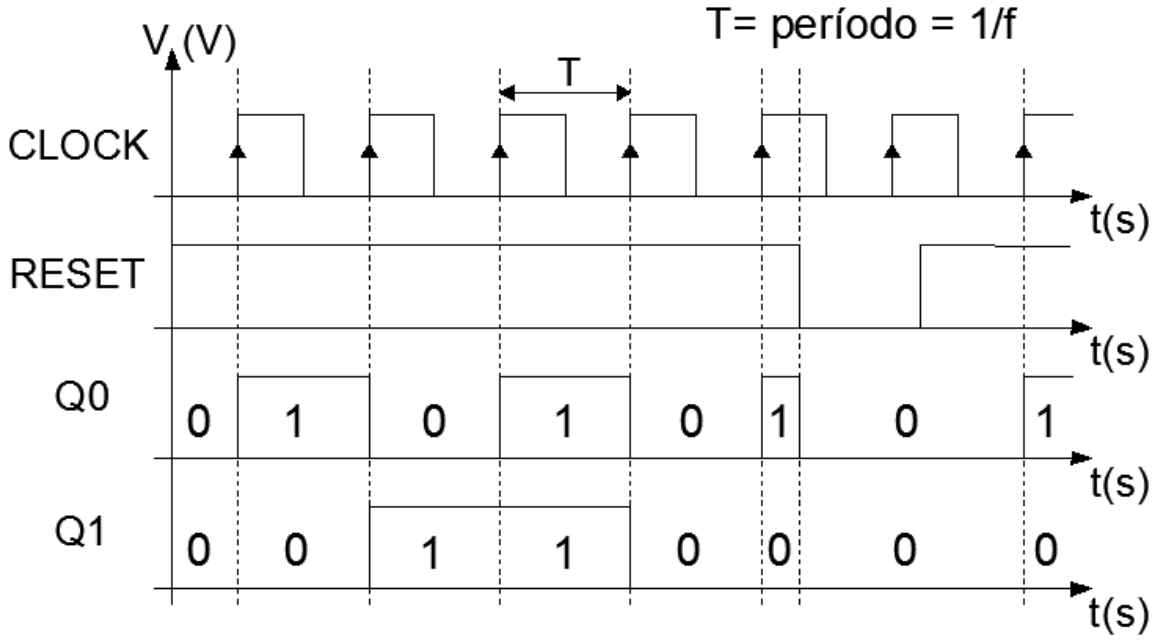


Figura 10- Diagrama de funcionamento do contador de pulsos 2 bits.
FONTE: Autor.

Pode ser visto na Figura 10 que o sinal de *clock* é um sinal de onda quadrada numa frequência (f) constante e a cada borda de subida que o sinal de *clock* realiza é incrementado um número na contagem, este valor de contagem é mostrado através do sistema de numeração binário composto pelos dois bits Q0 e Q1. Se o pino *reset* for colocado em nível lógico 0, a qualquer momento, a contagem é zerada e só é reiniciada no momento que o *reset* for colocado em nível lógico 1.

Após entender o funcionamento de um circuito contador de pulsos de dois bits, ele é descrito através de um diagrama de blocos (Figura 11).

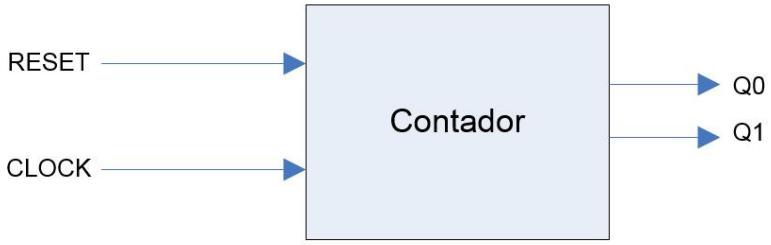


Figura 11- Diagrama de blocos do circuito integrado de um contador de pulsos de dois bits.
FONTE: Autor.

Note que, o circuito contador de *clock* de dois bits é composto por duas entradas digitais (*Clock* e *Reset*) e duas saídas (Q0 e Q1). O circuito funcionará da seguinte maneira: um circuito eletrônico de um contador de *clock* vai contar a quantidade de pulsos gerados pelo sinal de *clock*, chegando ao valor máximo suportado, que neste caso em específico é 4, o contador é reiniciado automaticamente. As saídas variam seu estado lógico de acordo com a contagem, formando o número na codificação binária. Caso a entrada de *reset* seja acionada, as saídas são zeradas automaticamente.

4.2 Descrição VHDL (bloco 2 da Figura 9)

Depois de concebido o projeto digital contendo todas as características que o circuito integrado do contador de pulsos de 2 bits deve atender, o próximo passo é a criação do código VHDL.

Para a realização desta etapa do fluxo do projeto foi utilizado o *software Quartus* II 8.1 *Web Edition*, que é um *software* gratuito, disponibilizado no *site* de seu fabricante (Altera) [17].

Na figura 12 é possível visualizar o código VHDL que foi criado para atender as especificações do projeto em questão.

```
ENTITY Contador IS      --define Contador com a entidade

    PORT ( Clock, Reset: IN STD_LOGIC ; --define Clock e Reset com entrada

        Q : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)) ; --define Q com uma vetor de saída composto de
                                                 --dois bits
    END Contador ;
ARCHITECTURE Projeto OF Contador IS

    SIGNAL Count : STD_LOGIC_VECTOR (1 DOWNTO 0) ; --define Count como uma variavel que
                                                   --trabalhará com o os valores de 0 a 2
        BEGIN
            PROCESS ( Clock, Reset )
            BEGIN
                IF Reset = '0' THEN     -- Se o pino de Reset estiver em nível lógico 0
                        Count <= "00" ; -- Count receberá o valor 00
            ELSIF (Clock'EVENT AND Clock = '1') THEN --Senão, a cada evento de borda de subida do Clock
                        Count <= Count + 1 ;    -- Count é incrmentado
            END IF ;
            END PROCESS ;
        Q <= Count ; -- A saída Q recebe o valor de Count
END Projeto ;
```

Figura 12- Código VHDL comentado do Circuito Contador de dois bits

Como pode ser visto no código VHDL do circuito contador de *clock*, é declarada a entidade do programa como Contador, e nesta mesma entidade é declarado os sinais que a compõe, como as entradas e saídas. Após esta declaração, o código VHDL realiza a contagem de pulsos de *clock* e a cada borda de subida deste sinal é incrementada uma variável (Count) que é automaticamente enviada para as saídas Q0 e Q1, já que a linguagem VHDL é concorrente. No momento em que o número de *clocks* for maior que a capacidade da variável Count, a variável é zerada e o ciclo recomeça. Em qualquer momento deste processo, se o pino de *Reset* for acionado, a variável Count é zerada e o ciclo só será recomeçado quando o *Reset* for desacionado.

4.3 Compilação do código VHDL do circuito Contador de pulsos de 2 Bits (bloco 3 da Figura 9)

A próxima etapa a ser realizada é a compilação do código VHDL, onde uma de suas funções é a análise sintática do código VHDL. Caso alguma instrução não esteja de acordo com o padrão VHDL o compilador acusa um erro e para o processo de compilação, que só prosseguirá após sua correção. Quando todos os erros forem corrigidos, o compilador mostra uma mensagem de sucesso e permitirá a realização da próxima etapa.

Outra função da etapa de compilação é a transformação da descrição de *hardware*, no caso o VHDL, para uma linguagem de máquina, que será utilizada na simulação.

4.4 Síntese do circuito contador de pulsos de 2 bits (bloco 3 da Figura 9)

Após ser gerado e compilado o código VHDL do circuito contador de pulsos de 2 bits, é iniciado o processo de síntese. O processo de síntese consiste numa tradução de um código VHDL para uma estrutura de *hardware*, que por sua vez deve realizar todas as funcionalidades do projeto [3]. Para a realização da síntese foi utilizado o próprio *Quartus II,* conforme demonstra o Apêndice A.

Podemos visualizar o resultado do processo de síntese na Figura 13, que apresenta uma representação eletrônica do circuito integrado desejado.

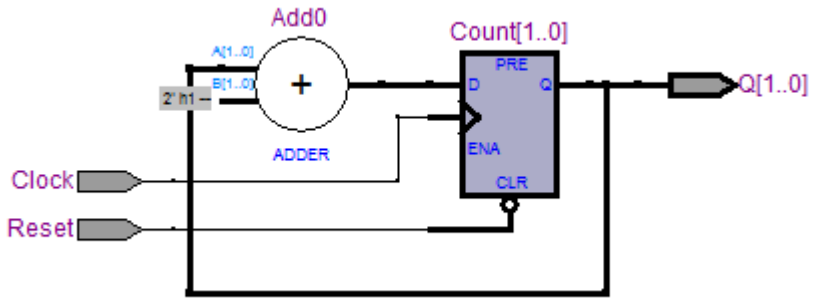


Figura 13- Resultado do processo de síntese do circuito contador de pulsos de 2 bits.

A primeira resposta apresentada pelo *Quartus* em relação ao esquema elétrico do contador foi na forma de blocos. Neste circuito têm-se as entradas (*Clock e Reset)*, um circuito somador e um *flip-flop* do tipo D, porém todos esses blocos correspondem a um circuito mais detalhado internamente. O próximo passo é o detalhamento deste circuito, de forma que se possa obter maiores informações sobre cada parte do projeto. A ferramenta *Quartus* nos permite fragmentar alguns destes blocos.

Para fragmentar os blocos lógicos em blocos mais detalhados é necessário selecionar o bloco lógico desejado, por exemplo, o bloco do contador, clicar sobre o mesmo e desmembrá-lo como é mostrado no Apêndice A. Realizando esse detalhamento, é obtido o resultado mostrado na Figura 14.

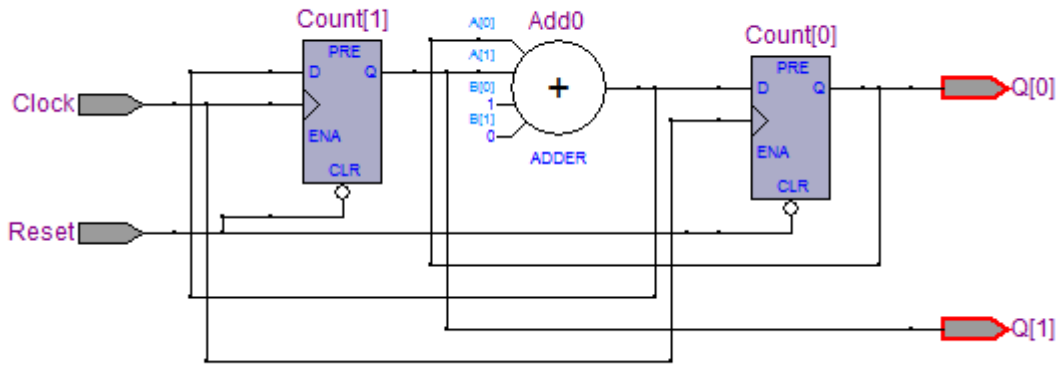


Figura 14- *Hardware* detalhado do circuito contador de pulsos de 2 bits.

Na Figura 14 pode-se ver os sinais de entrada de *clock* e *reset,* que são as entradas do sistema, assim como as saídas Q[0] e Q[1]. O circuito contador também possui dois *flip-flops* do tipo D, denominados de Count[1] e Count [2], e um circuito somador Add0 que é responsável pela lógica combinacional do circuito.

Para uma maior interpretação dos resultados obtidos no processo de síntese é possível visualizar um conjunto de expressões booleanas que definem as relações entre os sinais que compõem o circuito. Na Figura 15 são mostradas as expressões obtidas no circuito do contador de pulsos de 2 bits.

```
--Count[1] is Count[1] at LCFF_X1_Y26_N23
Count[1] = DFFEAS(A1L7, GLOBAL(A1L2), GLOBAL(A1L12),  ,  ,  ,  ,  ,  );


--A1L7 is Count[1]~9 at LCCOMB_X1_Y26_N22
A1L7 = !Count[1] & Count[0] # Count[1] & !Count[0];


--A1L5 is Count[0]~10 at LCCOMB_X1_Y26_N0
A1L5 = !Count[0];


--Q[0] is Q[0] at PIN_D18
--operation mode is output

Q[0] = OUTPUT(Count[0]);


--Q[1] is Q[1] at PIN_E20
--operation mode is output

Q[1] = OUTPUT(Count[1]);
```

Figura 15- Expressões booleanas do circuito contador de pulsos de 2 bits.

Nas expressões apresentadas na Figura 15, pode-se verificar os principais operadores de lógica boolena como: o “!”, que representa a operação lógica da negação (inversão), o ”&”, que representa a operação lógica “E” e o “#” que representa a operação lógica “OU”. Realizando as interações entre os sinais, de acordo com as respectivas equações lógicas, e acrescentando-os aos blocos de resultado do processo de síntese, tem-se o esquemático completo do circuito do contador de pulsos de dois bits proposto, como pode ser visto na Figura 16.

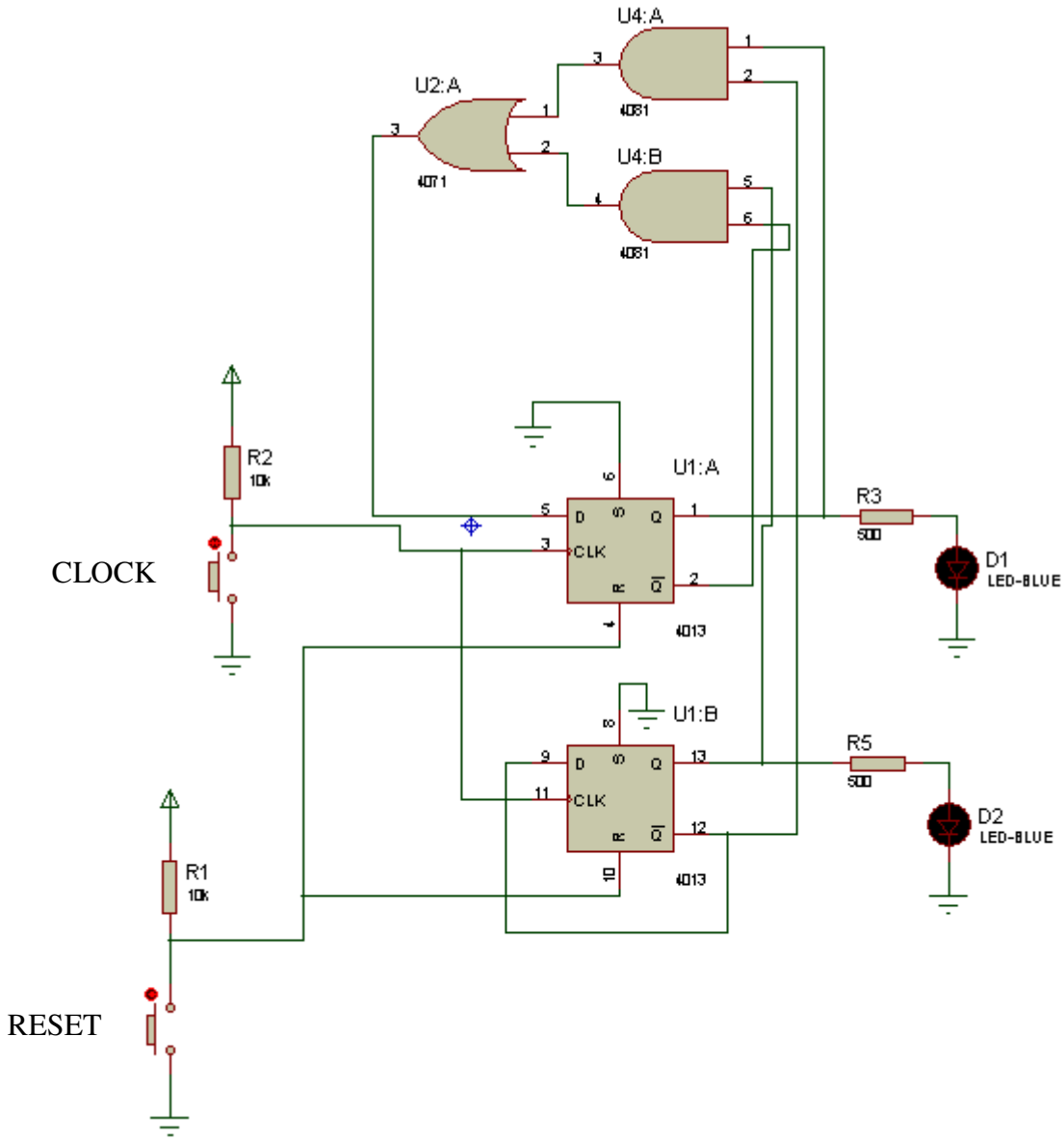


Figura 16- Circuito eletrônico completo do contador de pulsos de 2 bits.

A Figura 16 mostra o circuito do contador de pulsos que é formado, basicamente, por portas lógicas e dois *flip-flops* do tipo D. Com este circuito esquematizado foi possível passar para a próxima etapa do processo, que será a realização de uma simulação funcional e uma de tempo, onde é verificado se o circuito atende aos requisitos do projeto. Caso esta simulação não ocorra de maneira satisfatória, o projeto deverá ser revisto para que possam ser sanados os defeitos encontrados.

4.5 Simulação (bloco 4 da Figura 9)

Após toda a etapa de síntese, inicia-se a primeira etapa de simulação, esta simulação pode ser de dois tipos basicamente: a simulação lógica e a simulação de tempo.

Para que essas simulações ocorram é necessário ter um arquivo de estímulos, isto é, um arquivo onde estarão especificados os sinais de entrada com seus respectivos valores, que serão utilizados pela ferramenta de simulação para obtenção dos sinais de saída. Na Figura 17 é possivel visualisar o resultado da simulação lógica realizada no *Quartus*.

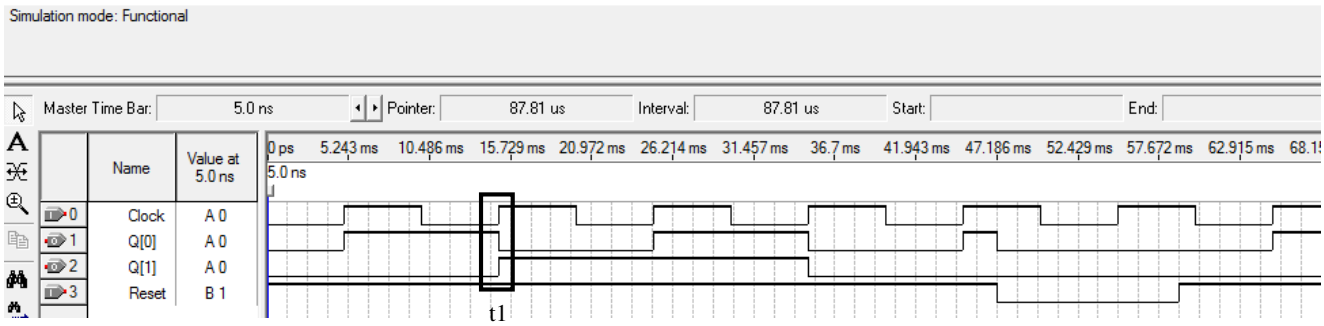


Figura 17- Simulação lógica do circuito contador de pulsos de 2 bits

Ao analisar a Figura 17 nota-se que ela possui 4 sinais (*Clock*, Q0, Q1, *Reset*) que são os mesmos que foram definidos no início do projeto. O sinal de *clock* é uma onda quadrada, simétrica, de frequência fixa de 100Hz usada para fins de simulação, que é a responsável por determinar a velocidade na qual o circuito irá realizar a contagem. Os sinais Q0 e Q1 são os sinais de saída que apresentam as respostas da simulação. E, por fim, o sinal de *Reset*, que enquanto estiver em nível lógico 1 não influência o funcionamento do circuito, mas no momento em que ele for colocado em nível lógico 0 as saídas Q0 e Q1 são zeradas. Analisando o momento de tempo “t1” observamos que no momento da borda de subida do *clock* os sinais Q0 e Q1 tem os seus níveis alterados, e o sinal de *reset,* neste período, permanece desabilitado (nível lógico 1).

Após a simulação lógica ter mostrado que o circuito está correto do ponto de vista funcional, é necessária a realização da simulação de tempos, onde são levadas em considerações as propagações dos sinais do circuito, a fim de analisar, de uma maneira mais próxima da realidade, se o circuito funciona adequadamente, para que os atrasos possam ser acentuados é necessário a utilização de uma maior frequência de clock, para isso na simulação de *timing* foi utilizado a frequência de 10MHz. Na Figura 18 podemos ver o resultado da simulação de *timing,* como ela é chamada.

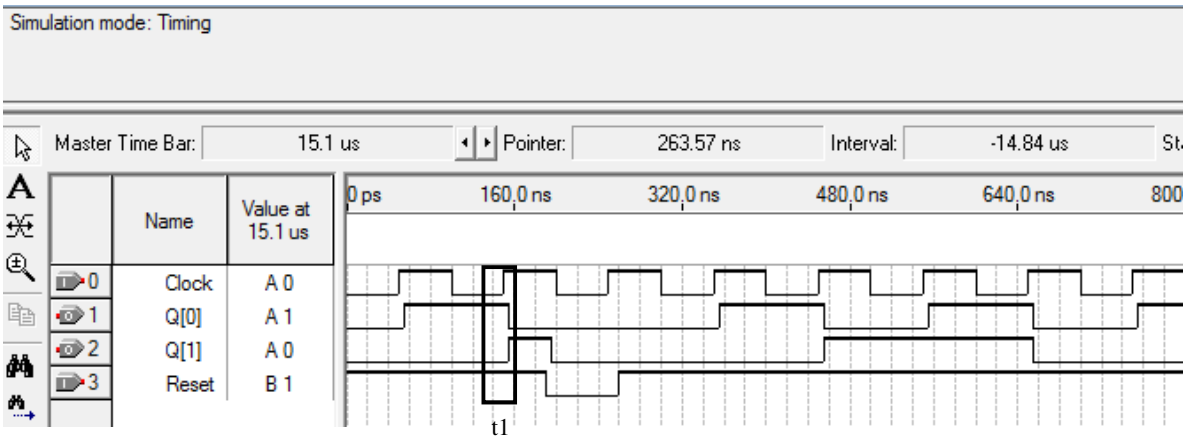


Figura 18- Simulação de tempo do circuito do contador de pulsos de 2 bits.

Depois de finalizada a simulação *timing*, pode-se analisar as formas de onda geradas (Figura 18) e concluir que mesmo levando em consideração todos os atrasos referentes a propagação dos sinais o circuito contador de pulsos apresenta seu perfeito funcionamento Analisando o momento de tempo “t1” observamos que no momento da borda de subida do *clock* os sinais Q0 e Q1 tem os seus níveis alterados, e o sinal de *reset,* neste período, permanece desabilitado (nível lógico 1).

4.6 Conversão para verilog e geração do leiaute (bloco 5 e 6 da Figura 9)

Neste momento, o circuito contador de pulsos de 2 bits tem sua funcionalidade garantida. Agora é necessário converter o circuito digital em um leiaute em nível de transistores MOSFETs.

O primeiro passo para a confecção da etapa de leiaute é a definição do *software* de leiaute a ser utilizado. No caso do projeto do contador de pulsos é utilizado o *software Microwind*, versão 2.0, pois é uma versão de demonstração, que não oferece custos ao projeto.

Na confecção do contador de pulsos de 2 bits é utilizado o método automático de geração de leiaute. Para isso, é preciso transcrever o esquemático apresentado na Figura 16 na única linguagem de descrição de *hardware* aceita pelo *Microwind*, isto é, a linguagem Verilog, onde o circuito deverá ser codificado em nível de porta lógica, contendo também as interconexões entre elas.

Para iniciar este processo de codificação foi realizado um bloco inicial, no caso o *flip-flop* do tipo D. Primeiramente foi definido o esquemático do *flip-flop* do tipo D em nível de porta lógica (Figura 19). Posteriormente, foi realizada a codificação Verilog deste circuito (Figura 20).

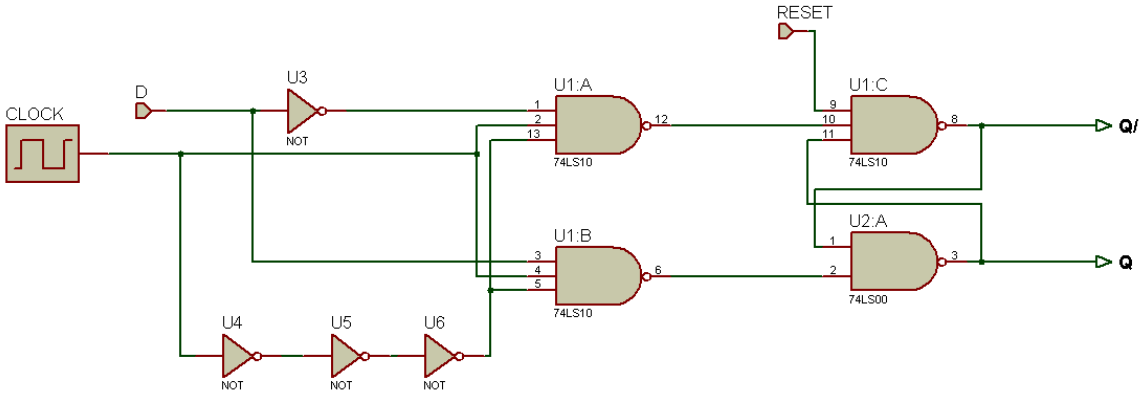


Figura 19- Esquemático de um *flip-flop* do tipo D em nível de porta lógica.

```
MODULE Flip FLop D( D,CLK,Q_INV,Q, RESET);
INPUT D, CLK,RESET;
OUTPUT Q, Q_INV;

wire CLK_INV,CLK1_INV,CLK2_INV,D_INV,S,R;

NOT n1 (CLK_INV, CLK);
NOT n2 (CLK1_INV, CLK_INV);
NOT n3 (CLK2_INV, CLK1_INV);
NOT n4 (D_INV, D);
NAND nand1 (S, CLK,CLK2_INV,D_INV);
NAND nand2 (R, CLK,CLK2_INV,D);
NAND nand3 (Q_INV, S, Q, RESET);
NAND nand4 (Q, R, Q_INV);

ENDMODULE
```

Figura 20- Descrição Verilog do *flip-flop* do tipo D em nível de porta lógica.

Na descrição apresentada na Figura 20, pode-se notar alguns detalhes da descrição Verilog, ou seja, ela se assemelha a descrição VHDL. A primeira instrução da descrição é a diretiva “MODULE”, que define o nome do módulo e de quais sinais eles são compostos, isto é, os sinais D, CLK, Q_INV e Q. Logo abaixo, são definidos quais os tipos de cada sinal, (saída ou entrada). No *flip-flop* do tipo D, os sinais D e CLK foram definidos como entrada e os sinais Q e Q_INV como saída.

Logo após estas declarações iniciais são definidas as portas lógicas e suas interligações como, por exemplo, na linha de código “NOT n1(CLK_INV, CLK);” tem-se uma porta lógica inversora (NOT), chamada de “n1”, que tem como sinal de entrada o CLK e como sinal de saída o CLK_INV. Analisando outra linha de código tem-se: “NAND nand2 (R, CLK,CLK2_INV,D);” uma porta lógica do tipo não E (NAND) de três entradas, chamada de nand2, que tem como sinais de entrada CLK,CLK2_INV,D e como sinal de saída o R. Feito todas as descrições de portas lógicas existentes e suas conexões, a codificação Verilog é finalizada com a diretiva “ENDMODULE”. Está descrição deve ser salva no computador no formato “.txt”, pois será utilizada futuramente.

Com a codificação Verilog já realizada e salva em um formato “.txt”, basta inclui-lá no *software Microwind*.

Nesta inclusão, o primeiro passo a ser realizado é a seleção da tecnologia de fabricação de CIs CMOS que será utilizada no projeto. Esta seleção define as regras de leiaute que devem ser respeitadas como, por exemplo: as dimensões mínimas de cada MOSFET, espaçamento entre as máscaras e todas as outras características físicas do circuito. Este projeto utiliza a tecnologia CMOS de 0,8µm.

Com essas configurações iniciais o leiaute já está pronto para ser implementado de maneira automática e o resultado é mostrada na Figura 21.

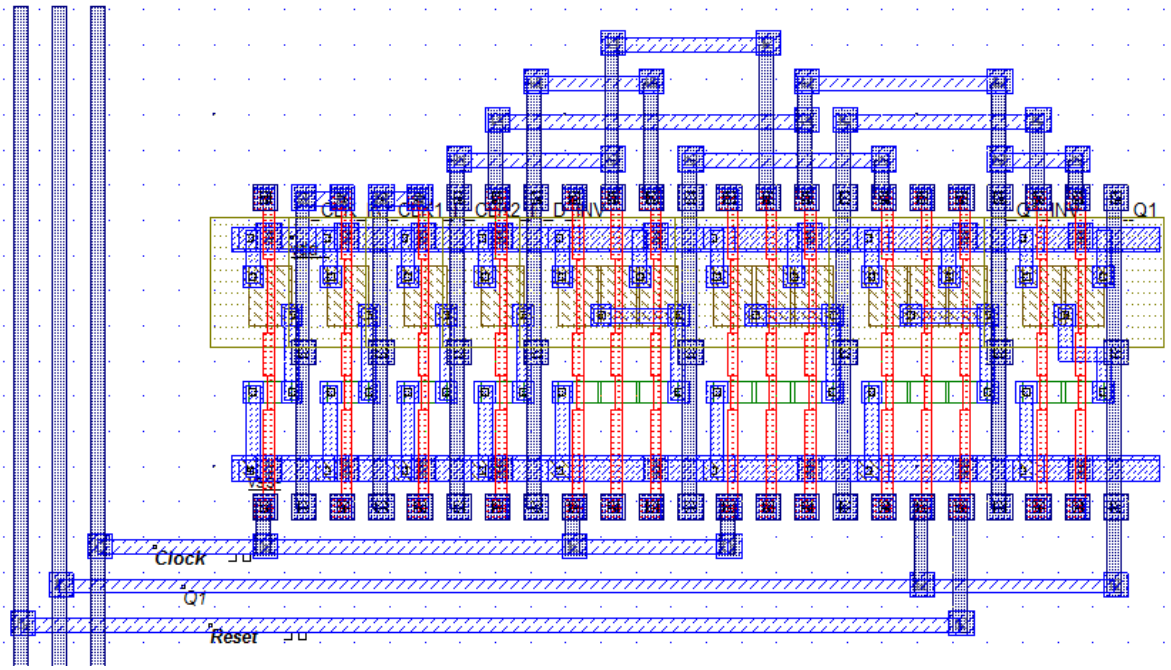


Figura 21- Leiaute gerado automaticamente pelo *Microwind* do circuito do *flip-flop* do tipo D.

Esta primeira etapa do leiaute serviu para validar a idéia de que o leiaute poderia ser realizado de maneira automática.

Após esta comprovação foi realizada a segunda etapa, que consiste em implementar todo o circuito seguindo o mesmo fluxo anterior, para isso com o esquemático geral (apresentado na Figura 16), foi realizado a transcrição Verilog de todo o circuito do contador de pulsos de 2 bits (Figura 22) e a geração do leiaute de forma automática (Figura 23).

```
module CONTADOR( Clock,Reset, Q1, Q2);

 input Clock, Reset;
 output Q1, Q2;
 wire CLK_INV,CLK1_INV,CLK2_INV,CLK3_INV,CLK4_INV,CLK5_INV;
 wire D_INV,D1_INV,Q1_INV,Q2_INV,S,R,S1,R1,A,B,C;



//FLip Flop 1

NOT n1 (CLK_INV, Clock);
NOT n2 (CLK1_INV, CLK_INV);
NOT n3 (CLK2_INV, CLK1_INV);
NOT n4 (D_INV, Q1_INV);
NAND nand1 (S, Clock,CLK2_INV, D_INV);
NAND nand2 (R, Clock,CLK2_INV, Q1_INV);
NAND nand3 (Q1_INV, S, Q1, Reset);
NAND nand4 (Q1, R, Q1_INV);


//FLip Flop 2

NOT n5 (CLK3_INV, Clock);
NOT n6 (CLK4_INV, CLK3_INV);
NOT n7 (CLK5_INV, CLK4_INV);
NOT n8 (D1_INV,C);
NAND nand5 (S1, Clock,CLK5_INV,D1_INV);
NAND nand6 (R1, Clock,CLK5_INV,C);
NAND nand7 (Q2_INV, S1,Q2 , Reset );
NAND nand8 (Q2, R1,Q2_INV);



//PORTAS LOGICAS EXTERNAS

AND and1(A,Q2,Q1_INV);
AND and2(B,Q1,Q2_INV);
OR  or1(C,A,B);

ENDMODULE
```

Figura 22- Descrição completa do circuito contador de pulsos de 2 bits, em linguagem Verilog.

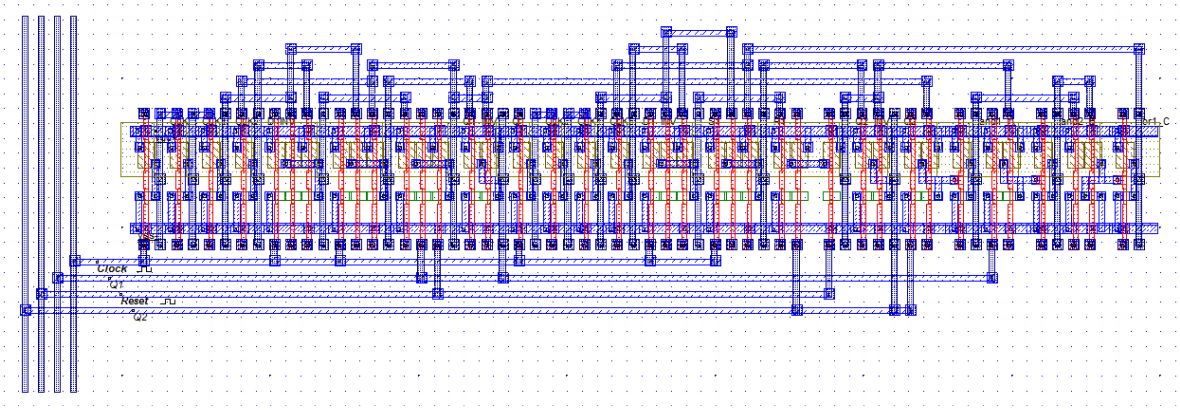


Figura 23- Leiaute gerado de maneira automática do contador de pulsos de 2 bits.

4.7 Verificação da geração das máscaras utilizadas pelo *Microwind* (bloco 7 da Figura 3-1)

Após o término do leiaute é iniciada a verificação das máscaras utilizadas pelo *Microwind*, onde se certifica de que o leiaute apresenta todas as camadas utilizadas na tecnologia definida.

A tecnologia utilizada para a criação do circuito contador de pulsos de dois bits foi a tecnologia CMOS de 0,8µm, que trabalha com uma tensão de 5,0V. As máscaras disponíveis nesta tecnologia são:

a) difusão N+;
b) difusão P+;
c) polisilício;
d) polisilício 2;
e) contato;
f) metal 1;
g) metal 2;
h) poço (N well).

Para conferir as camadas implementadas no projeto foi realizada uma visualização do leiaute de uma porta inversora, conforme demonstrado na Figura 3-22, onde serão indicados os posicionamentos de cada uma das máscaras.

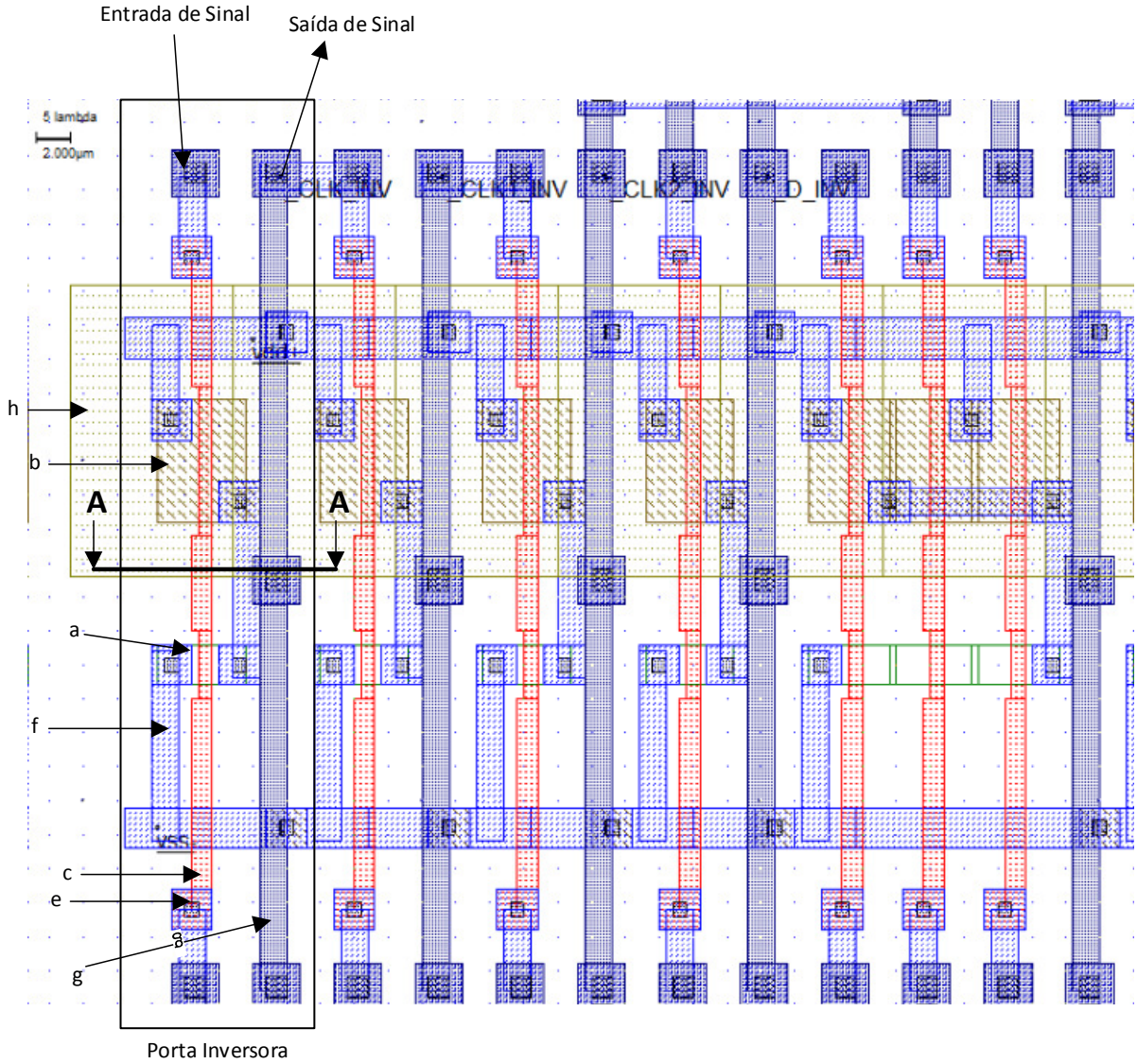


Figura 24- Detalhamento da porta inversora e posicionamento das máscaras utilizadas na tecnologia CMOS

A Figura 24 apresenta parte do leiaute do circuito contador de pulsos de dois bits. Em detalhe temos uma porta lógica inversora, onde será realizada uma visualização em corte na seção A-A (Figura 25). Além disso, na porta inversora estão indicadas cada uma das 8 máscaras definidas na tecnologia.

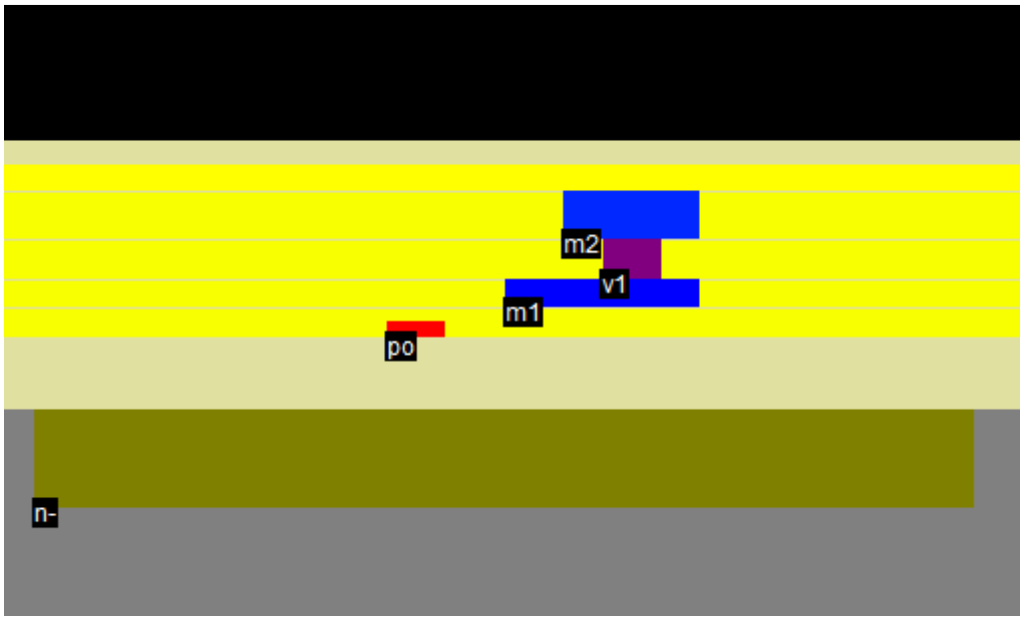


Figura 25- Vista em corte das camadas do leiaute

Na Figura 25 visualizam-se as máscaras que compõe o leiaute: os dois metais(m1,m2) que estão conectados entre si pela via 1 (v1), o polisilício (po) utilizado na porta dos transistores e o poço N (n-) enterrado no substrato.

4.8 Simulação Final (bloco 8 da Figura 9)

A última etapa do fluxo de projeto consiste na simulação do leiaute, onde são aplicados sinais nas entradas e analisam-se os sinais de saída.

Caso a saída esteja condizente com o que se espera do circuito integrado do contador de pulsos a etapa de projeto está finalizada, caso contrário, é necessária uma nova análise para encontrar onde ocorreu à falha durante o desenvolvimento do projeto e, neste momento, é necessário refazer todos os itens anteriores.

O *Microwind* possui diversas limitações na questão de simulação funcional do circuito através do seu leiaute. Portanto para a realização da simulação do circuito contador o mesmo deve ser simulado por blocos, isto é, em partes.

O primeiro circuito a ser simulado foi o do *flip-flop* do tipo D, conforme é apresentado na Figura 26.

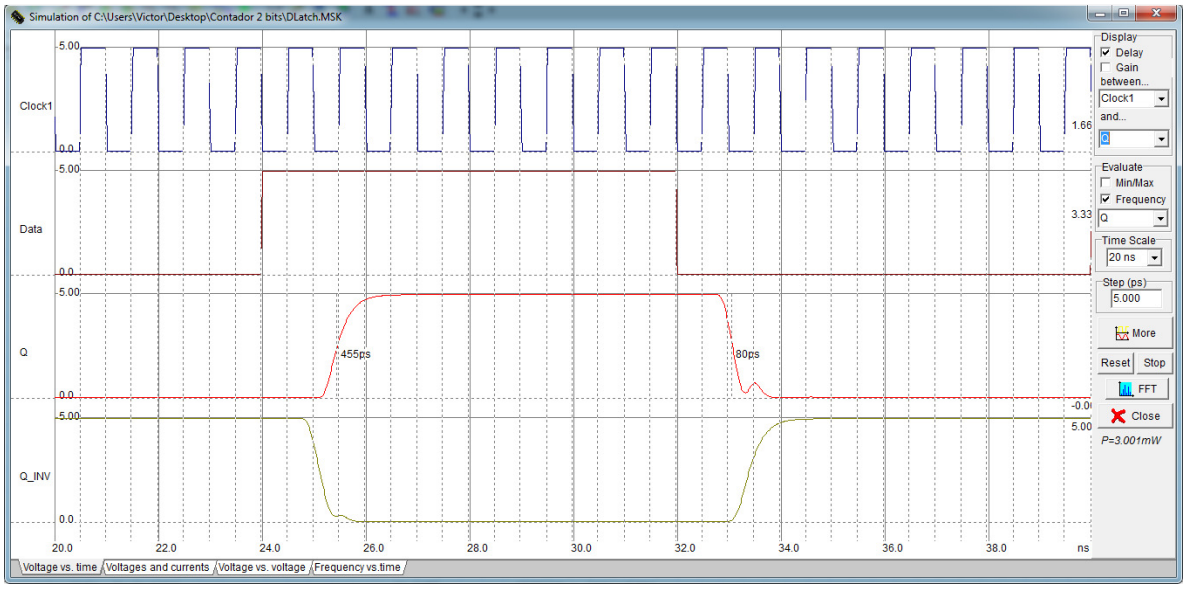


Figura 26- Simulação funcional do leiaute do *flip-flop do tipo* D que faz parte do circuito contador de pulsos.

A Figura 26 apresenta a resposta da simulação do circuito *flip-flop* do tipo D. Nesta resposta tem-se: o sinal de *clock,* que é uma forma de onda quadrada, a entrada de dados (data) e os sinais de saída Q e Q_INV. O comportamento do circuito é adequado ao comportamento esperado de um *flip-flop* do tipo D, que transfere o sinal na entrada de dados para a saída Q a cada pulso de *clock*. A saída Q_INV, como o próprio nome sugere, se trata do sinal da saída Q invertido.

Outro item simulado no *Microwind* foram as portas lógicas que complementam o circuito, este conjunto de portas lógicas que é utilizado no circuito do contador está ilustrado na Figura 27.

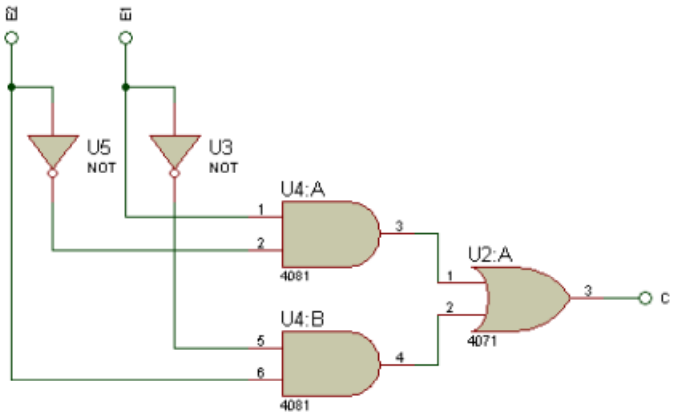


Figura 27- Conjunto de portas lógicas que complementam o circuito contador de pulsos, simulado utilizando o *Microwind.*

A Figura 27 apresenta um conjunto de portas lógicas que pertence ao circuito contador de pulsos de 2 bits. Assim como o *flip-flop* do tipo D, este conjunto foi simulado no *Microwind*. O circuito é composto por: duas portas do tipo inversoras, duas portas do tipo E e duas portas do tipo OU. Como a simulação do *Microwind* é uma etapa que ocorre após o leiaute, foi utilizado o método de geração de leiaute automático para a criação do mesmo, para isso foi realizado a transcrição Verilog (Figura 28).

```
module CONTADOR( E1,E2,C);

input E1, E2;
output C;
wire E1_INV,E2_INV,A,B;


//PORTAS LOGICAS EXTERNAS

NOT not1(E1_INV, E1);
NOT not2(E2_INV, E2);
AND and1(A,E2_INV,E1);
AND and2(B,E1_INV,E2);
OR  or1(C,A,B);

ENDMODULE
```

Figura 28- Transcrição para a linguagem Verilog do circuito de portas lógicas que compõe o circuito contador de pulsos de 2 bits

Esta transcrição, apresentada na Figura 28, descreve em linguagem Verilog todas as portas lógicas e as conexões entre elas. A estrutura da transcrição segue a mesma sintaxe apresentada nas outras transcrições do circuito contador de pulsos. A Figura 29 ilustra o leiaute gerado de maneira automática pelo *Microwind* a partir deste código Verilog.

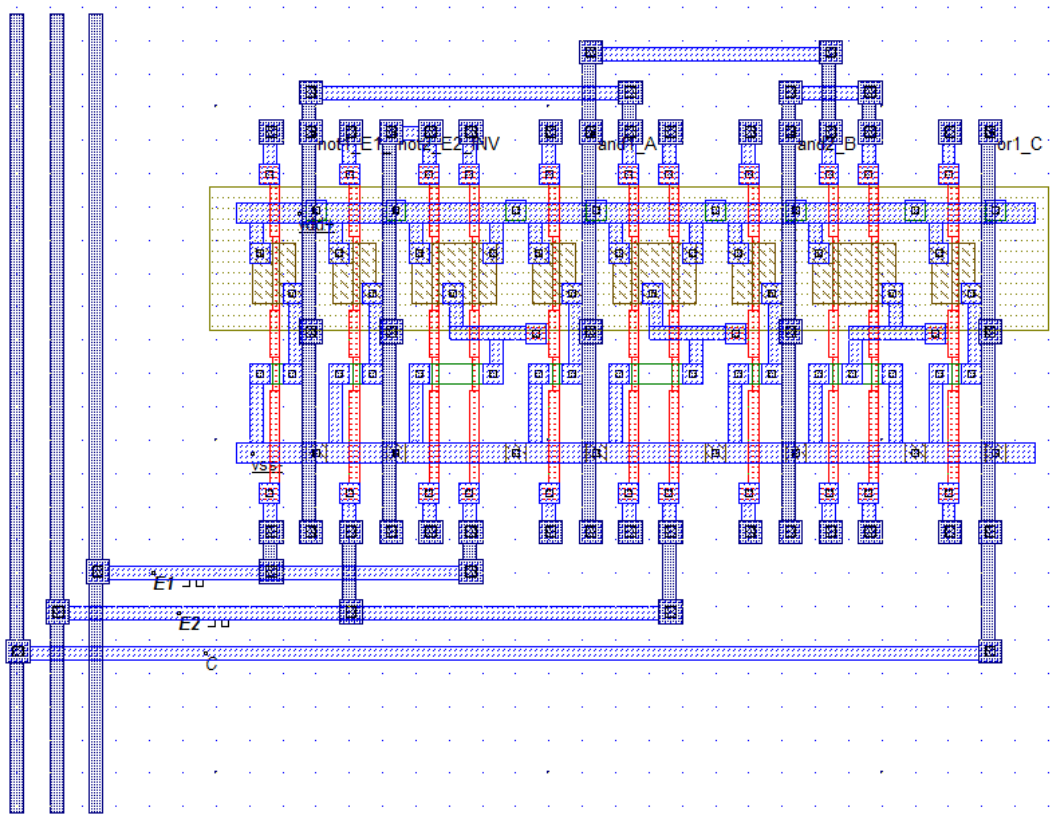


Figura 29- Leiaute gerado de maneira automática do circuito de portas lógicas que compõe o circuito contador de pulsos de 2 bits.

Utilizando no *Microwind* o código Verilog mostrado na Figura 28 obtem-se o leiaute de forma automática do circuito de portas lógicas, como mostrado na Figura 29. A partir deste leiaute será realizada a simulação do circuito e seus resultados são apresentados a seguir.

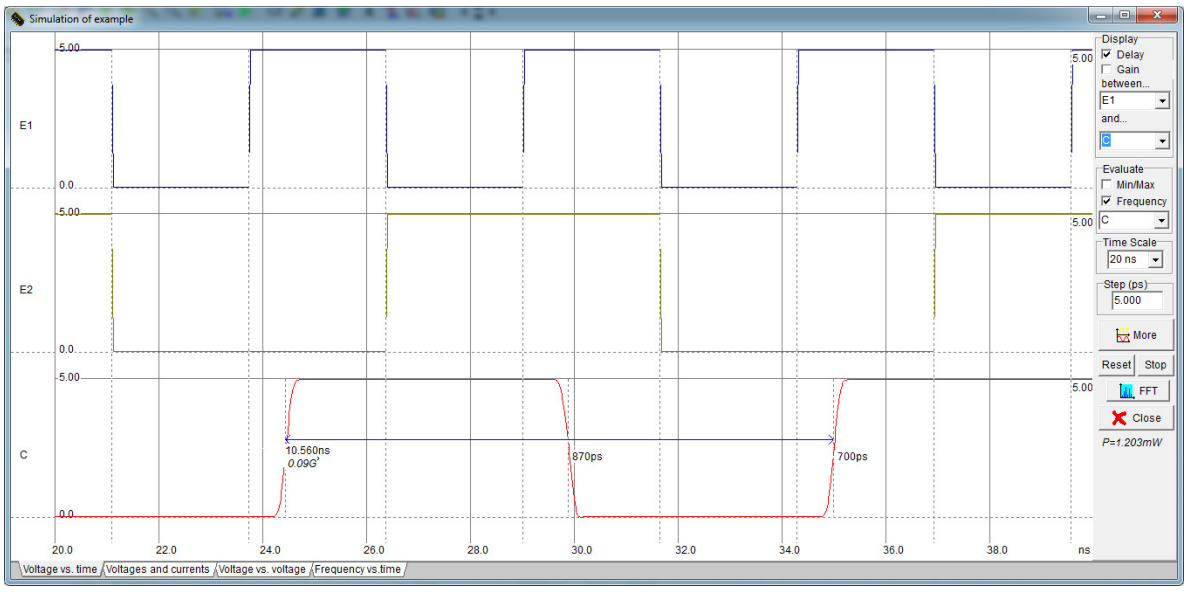


Figura 30- Resultado da simulação do circuito equivalente a porta ou exclusiva

A Figura 30 apresenta os resultados da simulação do circuito de portas lógicas que compõe o circuito contador de pulsos de 2 bits. Nesta simulação existem dois sinais de entrada (E0 e E1) e um sinal de saída (C). Os sinais de entrada juntos formam diferentes combinações e a saída apresenta o respectivo sinal correspondente. A Figura 31 exibe uma tabela que demonstra as combinações formadas pelos sinais de entrada (E0 e E1) e a resposta da saída C.

E2	E1	C
0	0	0
0	1	1
1	0	1
1	1	0

Figura 31- Tabela que relaciona as combinações de sinais das entradas E1 e E2 com a saída C.

Para cada combinação dos sinais de entrada (E1 e E2) temos uma resposta na saída (C), estas combinações de entradas foram definidas, pois são as quatro combinações possíveis de serem realizadas com dois bits, isso garante que todas as possibilidades serão simuladas. Este comportamento dos sinais são os mesmos apresentados na Figura 30, onde é exibido o resultado da simulação.

4.9 Conclusão

A simulação do fluxo utilizando ferramentas computacionais em versão de demonstração foi realizada através do software *Quartus II*, da fabricante Altera. Esta simulação tem como objetivo testar as funcionalidades do projeto, descritas em VHDL. A simulação nesta plataforma da Altera oferece uma enorme quantidade de recursos.

Como o circuito simulado possui um pequeno número de componentes (2 flips-flops do tipo D e 3 portas lógicas) a simulação ocorreu de forma rápida, o que permitiu diversas iterações ao longo do trabalho.

A sintetização do circuito contador de pulsos de dois bits foi realizada utilizando a ferramenta de síntese do *Quartus II*. Seu resultado foi apresentado nesta dissertação e ilustrado pela Figura 13.

Como o *software Quartus II* é bastante difundido nas universidades e instituições de ensino é possível encontrar diversas literaturas e materiais de pesquisa sobre sua utilização. Além disso, por se tratar de um *software* que é executado em plataforma *Windowns* e com interface gráfica, torna o processo mais fácil e intuitivo.

No fluxo utilizando ferramentas em versão de demonstração foi adotado o software *Microwind II*, para a geração do leiuaute. O *software* possui o recurso de realização do leiaute automático através de um arquivo Verilog que descreva todas as portas lógicas e conexões que compõe o circuito. Este arquivo Verilog criado de maneira manual utilizando um editor de texto. Usar utilizar

O arquivo Verilog foi inserido no *Microwind* e através dele foi gerado o leiaute. Durante a geração o *software* permite ao projetista alterar as dimensões dos transistores e definir os níveis de metal que deseja usar.

O posicionamento dos transistores é padrão, o *Microwind* posiciona todas as portas lógicas existentes no projeto alinhadas, uma ao lado da outra, e depois realiza as conexões entre si. Caso o projetista deseje mudar este posicionamento, deverá modificá-lo manualmente após a geração do leiaute.

A simulação pós-leiaute apresentou diversas limitações, tornando inviável a simulação do circuito por inteiro, foi necessário fragmentar o circuito e realizar duas fases de simulação. Porém este processo implica na perda de confiabilidade do sistema como um todo.

5 IMPLEMENTAÇÃO EM FPGA DE UM MICROCONTROLADOR 8051 A PARTIR DE UM CÓDIGO VHDL

Este capítulo tem como objetivo validar em FPGA um código VHDL que descreve um microcontrolador 8051.

Após o microcontrolador 8051 ser programado no FPGA foram executados dois programas de testes escritos em linguagem *Assembly*.

5.1 Caracteristicas do VHDL do 8051

O código VHDL utilizado neste projeto foi obtido no site http://www.cs.ucr.edu/~dalton/i8051/i8051syn/. Este código foi desenvolvido pelo Departamento de Ciências Computacionais da Universidade da California. O código VHDL apresenta a estrutura mostrada na Figura 32 [18].

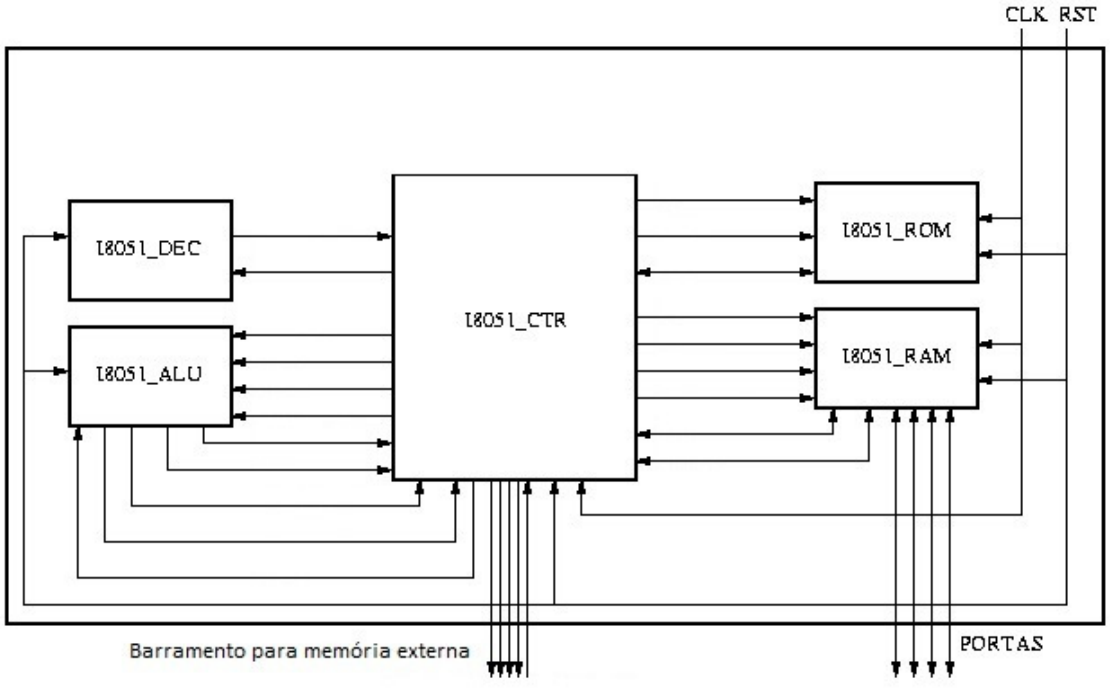


Figura 32- Arquitetura gerada a partir do código VHDL do microcontrolador 8051.
FONTE: [18]

O código VHDL do microcontrolador 8051 é composto por 5 blocos: um decodificador de instrução (I8051_DEC), responsável por interpretar as instruções do programa que está sendo executado, uma unidade lógica e aritimética (I8051_ALU), que é responsável por

realizar todas as operações aritméticas e lógicas do microcontrolador, o núcleo do microcontrolador (I8051_ctr), que representa a máquina de estados do microcontrolador, uma memória RAM (I8051_RAM), que é a memória de dados do microntrolador e uma ROM (I8051_ROM), que é a memória de programa do microcontrolador. Além destes blocos, o microcontrolador possui as portas de entradas e saídas, os pinos de controle (CLK e RST) e um barramento para o mapeamento de memória externa [18].

A validação do VHDL foi realizada com a placa didática de FPGA, modelo DE2-70, da fabricante Altera, que pode ser visualizada na Figura 33.

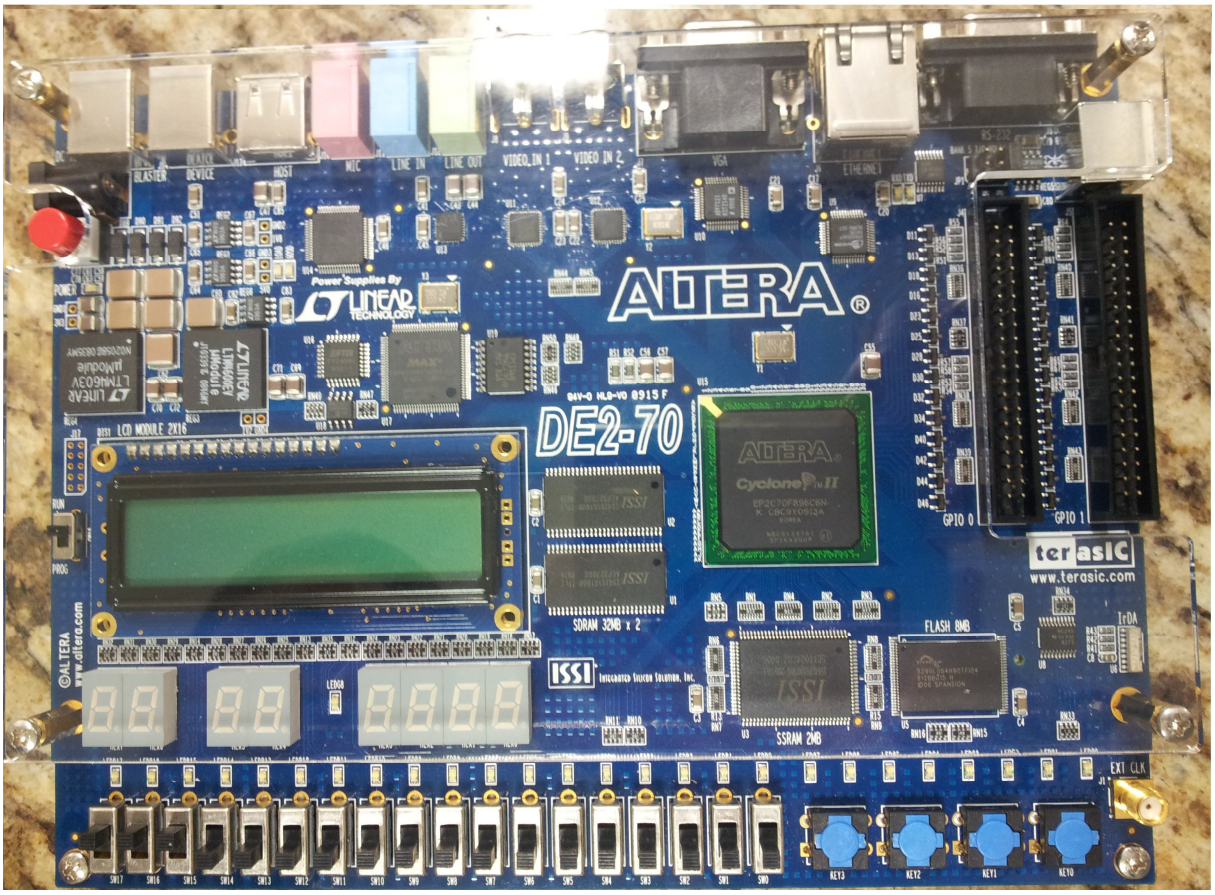


Figura 33- Placa didática com FPGA da Altera utilizada para a validação do VHDL do microcontrolador 8051
FONTE:Autor.

5.1.1 Criação do código VHDL do microcontrolador 8051 no *software* Quartus II para teste no *Kit* Didático de FPGA, modelo DE2-70, da fabricante Altera

Para a realização desta validação, o código VHDL de todos os blocos indicados na Figura 32 devem ser compilados e sintetizados como um único projeto, para que possa ser

realizado o *download* deste arquivo executável no *Kit* didático DE2-70*,* afim de que o FPGA possa realizar as funções de um microcontrolador 8051.

Todo este procedimento é feito utilizando o *software Quartus II* disponibilizado no *site* do fabricante.

Após o processo de compilação da linguagem VHDL do 8051 ser realizado no *software* Quartus II, é gerado um diagrama elétrico que o representa conforme está apresentado na Figura 34.

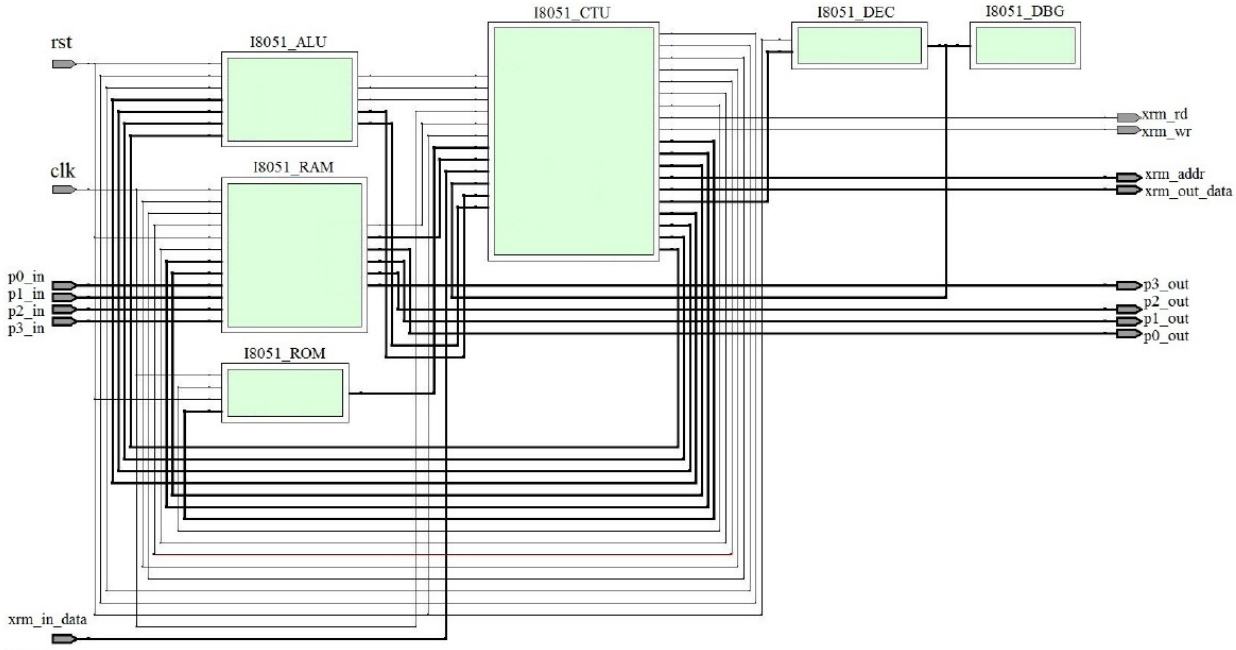


Figura 34- Diagrama elétrico gerado no *Quartus II,* que foi obtido a partir da compilação do código VHDL do microcontrolador 8051.

Podemos ver que o diagrama elétrico representativo do microcontrolador 8051 gerado é semelhante ao apresentado na Figura 34.

O bloco da ROM é o bloco da memória onde são armazenados os programas que serão executados. Os programas de testes, que são gravados na memória ROM, são gerados durante o processo de validação e são compilados em conjunto com o código VHDL do 8051, criando assim um novo bloco de memória ROM. Para a criação deste bloco de ROM contendo um determinado programa de teste, é necessário a criação do código fonte do programa de teste, que será executado pelo 8051. Este código pode ser escrito em qualquer linguagem de programação (*Assembly*, C, *Basic*, etc). Neste caso especificamente, os códigos fontes foram criados utilizando-se a linguagem *Assembly*. Após a criação destes códigos, eles devem ser compilados por um compilador *Assembly*, no qual gera um arquivo com a extensão ".hex". Existem diversos compiladores *Assembly*, que são específicos para microcontralador 8051,

sendo alguns deles gratuitos. Neste projeto foi utilizado o compilador Pequi, que está disponibilizado pela Universidade Federal de Goiás [19]. Uma vez que o programa de teste já foi compilado, é usado um aplicativo do próprio Departamento de Ciências da Computação da Universidade da Califórnia, para converter esse arquivo com extensão ".hex" em um arquivo VHDL. Esse novo arquivo será compilado no Quartus II em conjunto com o restante dos arquivos em VHDL do microcontrolador 8051, gerando um novo bloco de ROM. A cada modificação no programa de teste, é necessário realizar novamente este procedimento [18].

Para o teste de validação do VHDL do 8051 foram criados dois programas testes, que foram executados no *kit* didático do FPGA. A criação dos programas testes levou em consideração o esquema eletrônico mostrado na Figura 4-4, com o intuito de definir os pinos de entradas e saídas que foram utilizados.

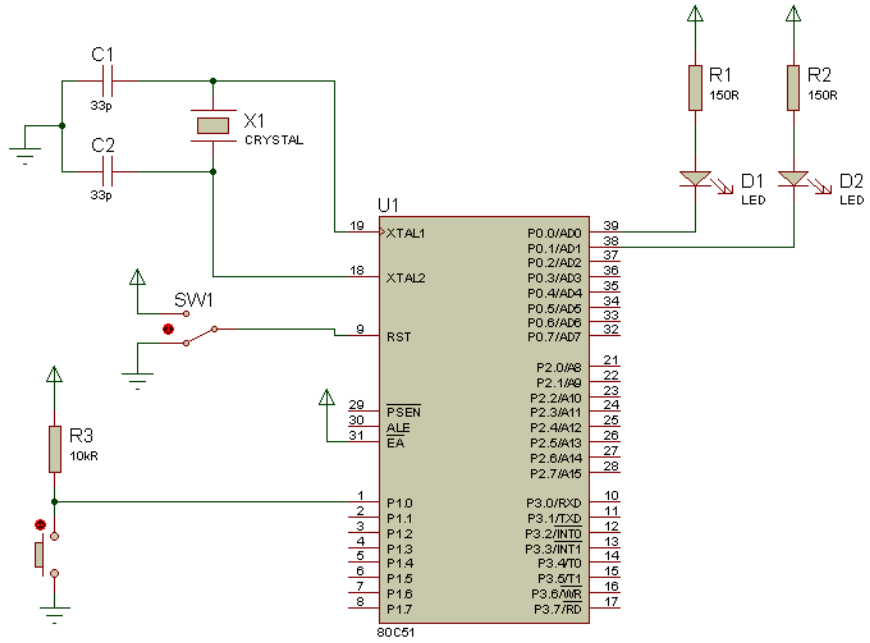


Figura 35- Esquema elétrico de referência para a execução dos programas de teste no 8051 em FPGA.

O esquema eletrônico da Figura 35 é composto por um microcontrolador 8051 (U1) que está conectado a dois leds (D1 e D2), a um botão que está conectado ao bit 0 da porta 1, a um cristal (X1) e ao *reset*.

O primeiro programa testado é simples e tem como intuito acionar e desacionar o led (D1) através do acionamento e desacionamento do botão. O programa teste está descrito da seguinte maneira:

```
$mod51                  ;Diretiva do PEQUI que carrega as bibliotecas
    ORG     0000        ; inicia o programa do endereço 0 da memória
INICIO:
    SETB    P0.0        ; Atribui 1 no pino P0.0
VOLTA:
    JNB     P1.0, LIGA ; Testa se a porta P1.0 é igual a 0.Se for, vai para LIGA
    JMP     INICIO      ;Pula para o endereço INICIO
LIGA:
    CLR     P0.0        ;Atribui 0 no pino P0.0, ligando o LED
    JMP     VOLTA       ;Pula para o endereço VOLTA
END                     ;Finaliza o programa
```

O primeiro programa teste opera de forma contínua, onde sua função é ficar testando a chave conectada no pino P0.0. Enquanto a chave estiver em seu estado natural (nível lógico 1), o led permanece apagado. Caso a chave ser pressionada, o led é acionado, permanecendo neste estado até que a chave seja desacionada.

No segundo programa de teste, dois leds (D1 e D2) ficam alternando seus estados lógicos a cada período de tempo, conforme o programa em *Assembly*:

```
$mod51                  ;Diretiva do PEQUI que carrega as bibliotecas
    ORG     0000H       ;inicia o programa no endereço 0
VOLTA:
    MOV     P1,#0FDH    ;Carrega no Port 1 o valor FDh
    CALL    TEMPO       ;Chama a rotina TEMPO
    MOV     P1,#0FEH    ;Carrega no Port 1 o valor FEh
    CALL    TEMPO       ;Chama a rotina TEMPO
    JMP     VOLTA       ;Pula para rotina VOLTA
TEMPO:
    MOV     R2,#0FFH    ;Carrega o registrador R2 com FFh
SALTO2:
    MOV     R1,#0FFH    ;Carrega o registrador R1 com FFh
SALTO1:
    MOV     R0,#0FFH    ;Carrega o registrador R0 com FFh
    DJNZ    R0,$        ;Decrementa R0 até ele ser zero
    DJNZ    R1,SALTO1   ;Decrementa R1, se o resultado for zero vai pra linha
                        ;de baixo, caso contrario vai para salta1
    DJNZ    R2,SALTO2   ;Decrementa R2, se o resultado for zero vai pra linha
                        ; de baixo, caso contrario vai para salta2
    RET                 ;Retorna da subrotina
END
```

Após a implementação destes programas teste, como já foi dito anteriormanente, eles devem ser incorporados no projeto do VHDL do circuito integrado do 8051, que será realizado com o auxílio do *Quartus* II. A pinagem do *kit* didático com o FPGA da Altera deve ser definida antes do processo de compilação do Quartus II, para definir os bits das portas de entrada/saída que serão usados pelos leds e chave. O mesmo deve ser feito para os pinos de *Reset* e de *Clock*. Realizada a compilação do VHDL do 8051, o arquivo de saída resultante desta compilação deve ser gravado no k*it* didático do FPGA.

No manual do *kit* do FPGA é possível saber todas as conexões entre o dispositivo lógico e o *hardware* disponível no *kit*. A Figura 36 indica os pinos que foram atribuídos aos leds e chave para a realização dos testes do 8051 com os programas de teste [20].

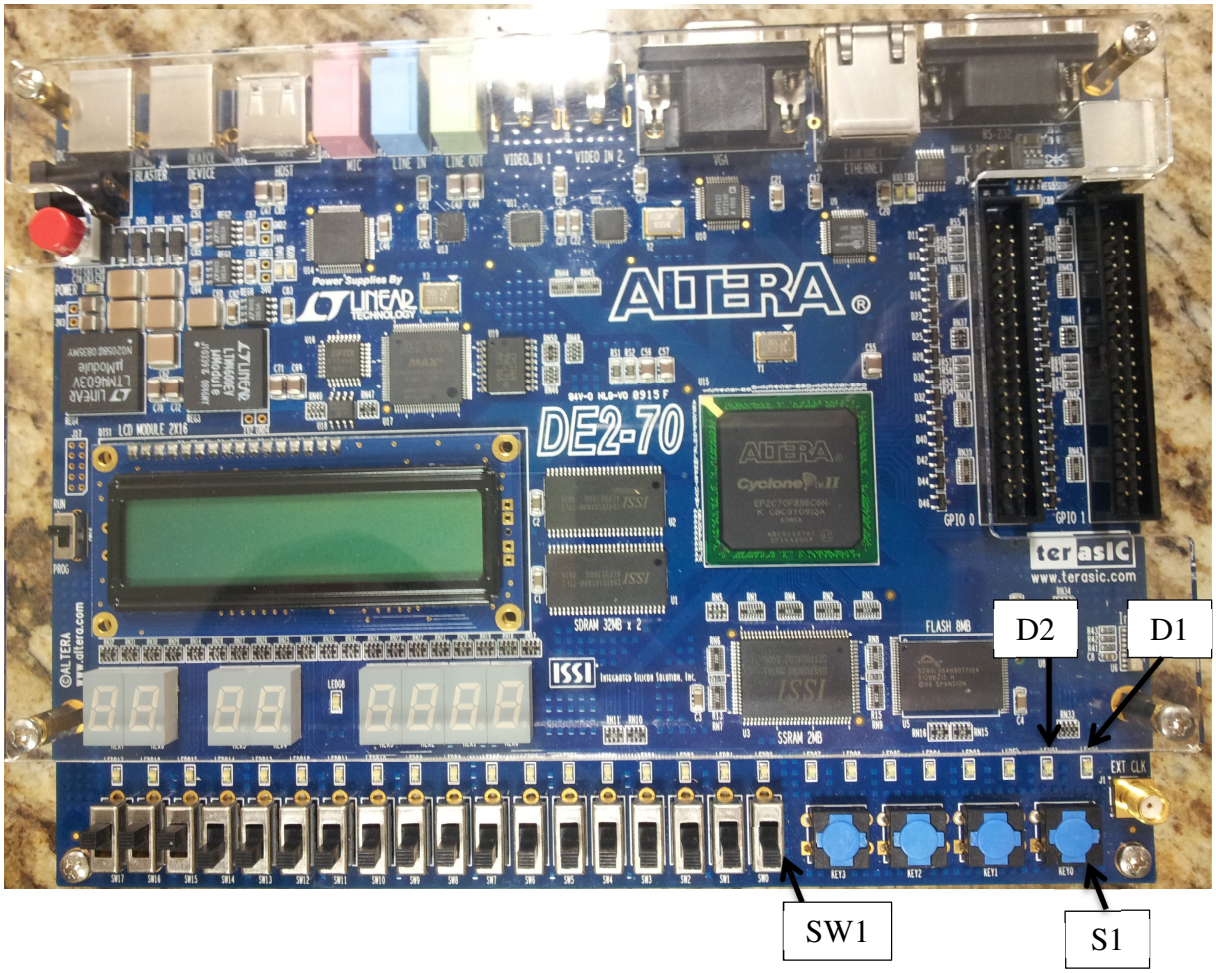


Figura 36- Indicação dos pinos das portas de entrada/saída para serem utilizadas pelos leds e chave no *Kit* didático de FPGA da ALTERA.

Depois da verificação do funcionamento adequado dos programas de testes, o VHDL do 8051 é validado e a próxima etapa é a geração automática do seu leiaute para posterior fabricação do mesmo utilizando-se uma determinada tecnologia CMOS de fabricação de circuitos integrados.

6 GERAÇÃO AUTOMÁTICA DO LEIAUTE DO NÚCLEO DO 8051 UTILIZANDO AS FERRAMENTAS DE CAD DA *MENTOR GRAPHICS*

Com todas as etapas do fluxo de projeto até aqui concluídas, a próxima etapa é a realização do leiaute de forma automática, utilizando-se as ferramentas de CAD disponíveis no centro universitário da FEI, que neste caso é a da *Mentor Graphics*.

Para o teste de verificação do funcionamento do VHDL do 8051, foram utilizados todos os blocos mostrados na Figura 34, executando-se os programas teste, porém para a criação do leiute, por se tratar de um projeto complexo, só foi realizado o leiaute automático do núcleo do 8051.

6.1 Ferramentas profissionais de leiaute da Mentor Graphics

O Centro Universitario da FEI faz parte de um grupo de mais de 1200 universidades parceiras de um Programa de Ensino Superior conhecido como HEP (*High Education Program*) criado pela empresa *Mentor graphics* em 1985, com o intuito de auxiliar as universidades na formação de recursos humanos capacitados nas mais diversas áreas da engenharia elétrica [21].

Para este trabalho foi utilizado um conjunto de ferramentas computacionais focados no projeto ASIC (ASIC *Design Kit*, ADK*),* versão 3.1. Este *design kit* traz todas as informações necessárias para criação de um circuito integrado ASIC como bibliotecas, símbolos, regras de projetos, além de manuais e tutoriais [22].

O *design Kit ADK* estava sendo utilizado junto ao *software ICstation, software* de leiaute da *Mentor,* porém no decorrer do projeto foi necessário a atualização do *software* de leiaute por uma nova versão, que substituiu a anterior. Este novo *software* é chamado *Pyxis Layout*, e neste caso o ADK tornou-se obsoleto, dando lugar ao *Generic Design Kit* (GDK) [23].

Neste novo conjunto de ferramentas computacionais (GDK) foi implementado um novo programa chamado *Project Navigator,* que tem como seu principal objetivo auxiliar no gerenciamento dos arquivos de um projeto, além de auxiliar na utilização das diversas ferramentas de projeto de circuitos integrados [23].

6.2 Sintetização dos arquivos VHDL do núcleo do 8051 utilizando-se o *software Leonardo Spectrum*

O processo de sintetização, como já explicado anteriormente, receberá o código VHDL e realizará sua transcrição para circuitos lógicos que serão capazes de executar as funções especificadas do circuito. Para este processo de sintetização é utilizada a ferramenta *Leonardo Spectrum* da *Mentor Graphics* [24].

No processo de sintetização é possível realizar algumas otimizações do CI, levando-se em consideração alguns parâmentros tais como a velocidade de processamento ou a quantidade de área de silício consumida pelo CI. Esses processos de otimização são realizados através de algoritmos eurísticos, porém em alguns casos estes algorítmicos não conseguem realizar sua função dependendo da descrição VHDL utilizada [24].

A Figura 37 apresenta o diagrama de blocos com algumas etapas do fluxo de projeto de circuito integrados digitais, onde mostra a utilização do *Leornardo spectrum*.

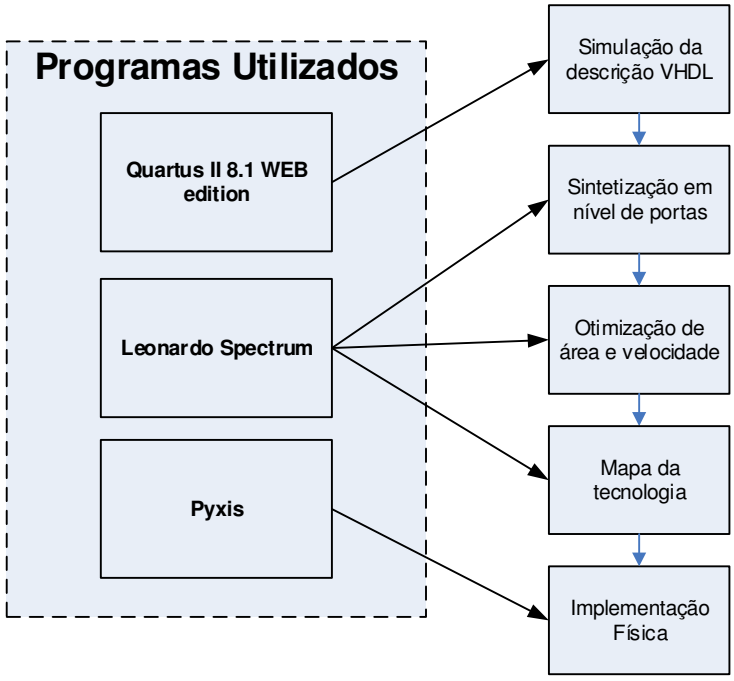


Figura 37- Fluxo de projeto de CI digitais onde mostra a utilização do Leonardo Spectrum
FONTE: Autor [24].

Após a definição do projeto, a geração do código VHDL e a certeza, obtida através da validação em FPGA, que o código está de acordo com as especificações, analisa-se o fluxo de projeto de circuitos integrados digitais, ilustrado na Figura 37, e inicia-se o processo da sintetização que irá receber como entrada o arquivo VHDL. Após as etapas descritas no fluxo da Figura 37, tem-se a descrição de um circuito através de um arquivo na linguagem Verilog.

A primeira etapa do processo de sintetização é a conversão do VHDL numa representação do circuito em nível de portas lógicas. Posteriormente são aplicados algoritmos de otimização que podem dar prioridade a menor utilização de área de silício ou um melhor desempenho em velocidade de processamento. Finalmente, após essa última atividade, é gerado um arquivo com a descrição do circuito escrita em linguagem Verilog. Este arquivo de saída da sintetização do CI é utilizado nos próximos passos do fluxo de projeto de CIs digitais para a obtenção do leiaute.

Na figura 38 é mostrado o conjunto de instruções que são utilizados nos processo de síntese quando se utiliza o *Pyxis Leonardo Spectrum*.

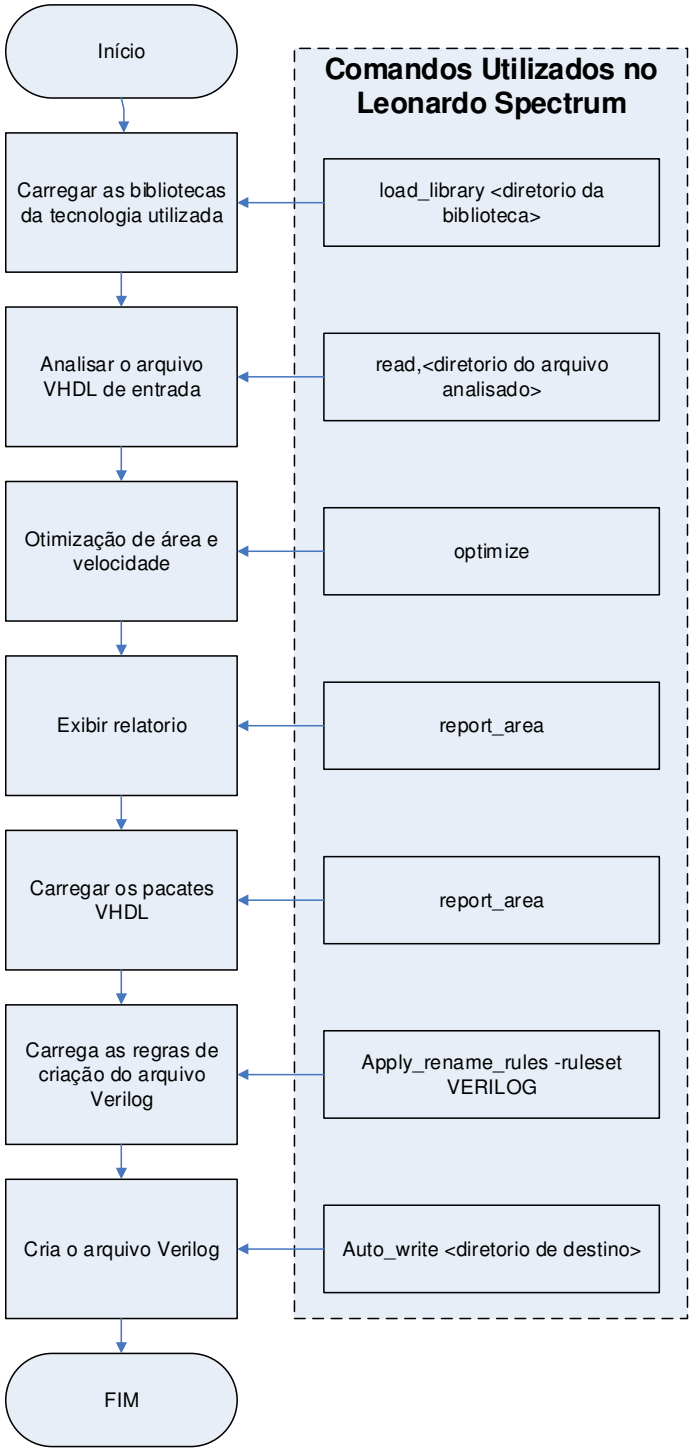


Figura 38- Fluxo e comandos utilizados no processo de sintetização realizado com *Leonardo Spectrum*
FONTE: [24]

No processo de síntese, a primeira etapa é a definição da biblioteca que será utilizada no projeto. Neste projeto, a tecnologia escolhida para a realização automática do leiaute é a tecnologia do fabricante TSMC de 0.35µm. A próxima etapa a ser realizada é a verificação do VHDL. Caso seja encontrada alguma falha neste processo, o processo não irá prosseguir até que a mesma seja eliminada. Depois de analisado, o código VHDL passará por um processo de otimização, que priorizará a velocidade de processamento do sistema ou a redução de área de

silício utilizada. Por padrão, o comando *optimize* realiza a otimização dando prioridade para área de silício gasta pelo CI. O comando report_area é responsável pela visualização dos parâmetros referentes à utilização de silício pelo CI (figura 39). A última etapa deste processo de sintetização é a criação do arquivo Verilog, que será utilizado como arquivo de entrada para a criação do leiaute do núcleo do 8051.

```
*******************************************************

Cell: I8051_CTR    View: BHV    Library: work

*******************************************************

 Number of ports :                      152
 Number of nets :                      4341
 Number of instances :                 4265
 Number of references to this view :      0

Total accumulated area :
 Number of gates :                     5979
 Number of accumulated instances :     4265
Info, Command 'report_area' finished successfully
```

Figura 39- Resultado do comando "report_area" do núcleo do 8051.

6.3 Criação das células padrão e do esquemático do núcleo do 8051 no *software Pyxis Schematic*

Realizada a etapa de sintetização, a próxima etapa do fluxo de desenvolvimento de projetos de CIs digiais, é a representação do circuito integrado através do seu esquemático. Esta representação pode ser realizada de dois modos: pelo modo manual e pelo modo automático. Neste projeto, por se tratar de um circuito integrado com extrema complexidade, optou-se pela utilização da geração automática, através da importação do arquivo Verilog gerado anteriormente.

Para a realização do esquemático será utilizada a ferramenta *Pyxis Schematic,* versão 10.2, que é a sucessora da ferramenta *Design Architect* da própria *Mentor Graphics*.

Para a realização desta etapa, além do arquivo Verilog, é necessário um arquivo de mapeamento que é responsável por realizar a associação entre as células primitivas, que foram detectadas no processo de sintetização, e os circuitos das bibliotecas que serão utilizadas. O arquivo de mapeamento é um arquivo texto que descreve as células utilizadas no projeto

contendo o endereço da localização destas células dentro da biblioteca utilizada. A Figura 40 ilustra o arquivo de mapeamento utilizado no projeto do núcleo do 8051. O arquivo em questão pode-se dividir em duas partes: as células/sinais primitivos e os módulos específicos do projeto.

```
 1 p portin $MGC_IC_GENERIC_LIB/portin X
 2 p portout $MGC_IC_GENERIC_LIB/portout X
 3 p portbi $MGC_IC_GENERIC_LIB/portbi X
 4 p ripper $MGC_IC_GENERIC_LIB/rip/1X1 bundle wire
 5 p pagein $MGC_IC_GENERIC_LIB/offpag.in IN
 6 p pageout $MGC_IC_GENERIC_LIB/offpag.out OUT
 7 p netcon $MGC_IC_GENERIC_LIB/netcon IN OUT
 8 p ground $MGC_IC_GENERIC_LIB/ground VSS
 9 p power $MGC_IC_GENERIC_LIB/vdd VDD
10 //
11 m PadInC $GDKGATES/PadInC DataIn DataInB Pad
12 m PadOut $GDKGATES/PadOut Pad DataOut
13 m dffr $GDKGATES/dffr Q QB D CLK R
14 m mux21_ni $GDKGATES/mux21 Y A0 A1 S0
15 m oai22 $GDKGATES/oai22 Y A0 A1 B0 B1
16 m mux21 $GDKGATES/mux21 Y A0 A1 S0
17 m aoi32 $GDKGATES/aoi32 Y A0 A1 A2 B0 B1
18 m oai21 $GDKGATES/oai21 Y A0 A1 B0
19 m aoi21 $GDKGATES/aoi21 Y A0 A1 B0
20 m nand02 $GDKGATES/nand02 Y A0 A1
21 m nand03 $GDKGATES/nand03 Y A0 A1 A2
22 m nor04 $GDKGATES/nor04 Y A0 A1 A2 A3
23 m nand04 $GDKGATES/nand04 Y A0 A1 A2 A3
24 m oai321 $GDKGATES/oai321 Y A0 A1 A2 B0 B1 C0
25 m nor03_2x $GDKGATES/nor03_2x Y A0 A1 A2
```

Figura 40- Arquivo de mapeamento utilizado no projeto do núcleo do 8051.

As células consideradas primitivas são as células que aparecem em todos os projetos como, por exemplo, o sinal de referência de tensão, visto na oitava linha da figura 40. Esta linha inicia-se com a diretiva “p”, que indica que a célula é uma célula primitiva e os demais itens visualizados seguem os seguintes atributos: <tipo> <primitivo> < localização do símbolo> <sinal (s) conectado (s)>. Todas as demais linhas do arquivo de mapeamento possuem a mesma estrutura.

A segunda parte do mapa apresentado, que compreende as linhas de 11 a 25, tem-se as células iniciadas pela letra “m”, que significa que elas são os módulos. Por exemplo, a linha 22 “m nor04 $GDKGATES/nor04 Y A0 A1 A2 A3” é um módulo que será utilizado no projeto chamado nor04, que indica ser uma porta lógica Não OU (NOR) de 4 entradas. Neste módulo também é indicado onde a porta está localizada na biblioteca e quais são os sinais conectados a ela (A0, A1, A2 e A3 que são as entradas e o sinal Y que é uma saída). Portanto, todos os módulos obedecem a seguinte sintaxe: <tipo> <nome do modulo><endereço do modulo> <sinais>.

Dependendo da biblioteca utilizada, os nomes dados aos módulos pelo arquivo de mapeamento são diferentes da nomenclatura utilizada na biblioteca. Neste caso é necessário ajustá-las para que eles apresentem os mesmos nomes. Caso os nomes não coincidam, ocorerá um erro no momento da geração do esquemático.

A Figura 41 apresenta uma parte do esquemático do núcleo do 8051, que foi gerado pelo Pyxis Schematic.

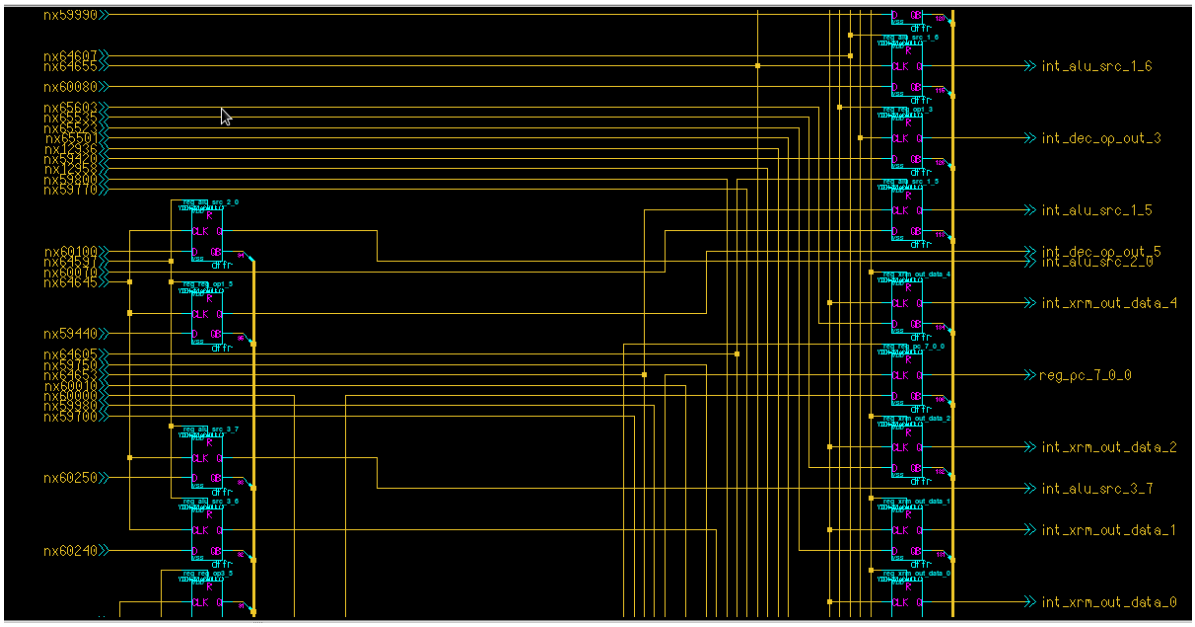


Figura 41- Parte do esquemático do núcleo do 8051.

6.4 Geração automática do leiaute do núcleo do 8051, utilizando-se o *software Pyxis Layout*

A última etapa do fluxo de projeto de circuitos integrados da Mentor é a geração automática do leiaute, que é dividida em seis passos, conforme ilustra o diagrama de blocos da figura 42.

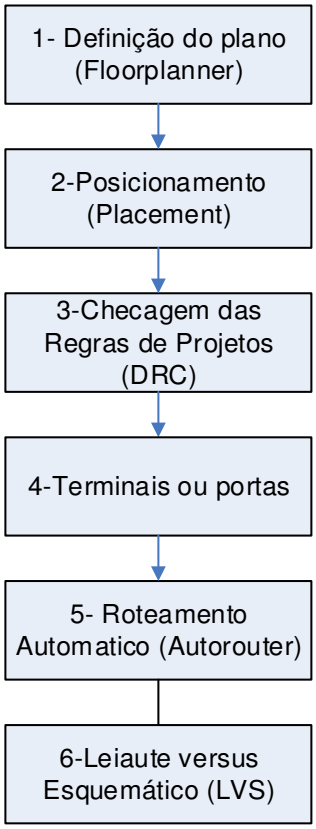


Figura 42- Etapas do fluxo de projeto de CIs digitais da Mentor para a geração automatica de leiaute.

A criação do leiaute inicia-se com a definição do plano (1- *Floorplaner*). Nesta etapa são analisadas todas as informações obtidas nas etapas anteriores e é definido um plano de trabalho onde ficará posicionado o circuito integrado. Além disso, é possível particionar o plano de trabalho e estimar a área necessária para a realização do leiaute. A etapa seguinte é a de *Placement,* que é o posicionamento de cada uma das células no CI. As células que serão posicionadas são aquelas do *netlist* do núcleo do 8051. Após posicionadas todas as células, é realizada uma checagem das regras de projeto do processo de fabricação dos CIs (3-DRC), de acordo com a tecnologia selecionada. Somente quando todas as regras forem atendidas será possível passar para próxima etapa (4-Terminais ou portas), que trata da implementação dos terminais do circuito.

A quinta etapa (5- Autorouter) corresponde à realização das interligações entre as portas. As ferramentas EDA, normalmente oferecem a possibilidade de realizar o roteamento (processo de interligar eletricamente as células do circuito integrado) de maneira manual ou automática. O método manual permite ao usuário realizar todas as interligações da maneira que lhe for conveniente, porém este método apresenta como principal desvantagem o tempo gasto na realização desta etapa. O método automático de roteamento necessita de um tempo

consideravelmente menor que o manual, além de testar as melhores possibilidades de roteamento entre as portas. No projeto do núcleo do 8051, foi utilizado o método automático.

A última etapa do fluxo de desenvolvimento de projetos de CIs digitais é a etapa de verificação entre o leiaute e o esquemático (6-LVS), onde é feito uma análise que verifica se todos os itens e conexões do esquemático foram implementados no leiaute. Após todas essas etapas, o leiaute pode ser alterado da maneira que o projetista julgar necessário, como por exemplo, pode-se adicionar outros dispositivos que aumentem a confiabilidade do processo de fabricação, como os dispositivos *dummy,* etc.

Para que o leiaute possa ser enviado para a fabricação, ele deve ser convertido em arquivos no formato GDSII.

Finalizadas as etapas de desenvolvimento de projeto de CIs digitais com os procedimentos da *Mentor Graphics,* é possível obter o leiaute do núcleo do 8051 (figura 43).

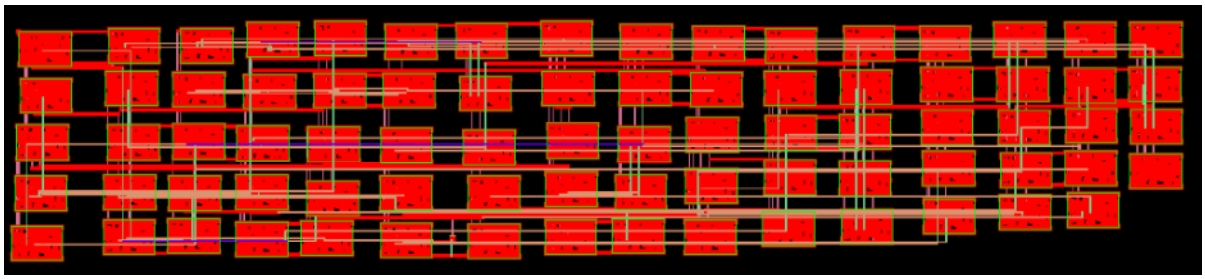


Figura 43- Leiaute do núcleo do 8051 gerado de forma automática pelas ferramentas da *Mentor Graphics*.

Este leiaute possui uma grande quantidade de componentes, cada retângulo representa um bloco lógico que realiza uma determinada função. A figura 44 ilustra, em detalhes, um destes blocos lógicos.

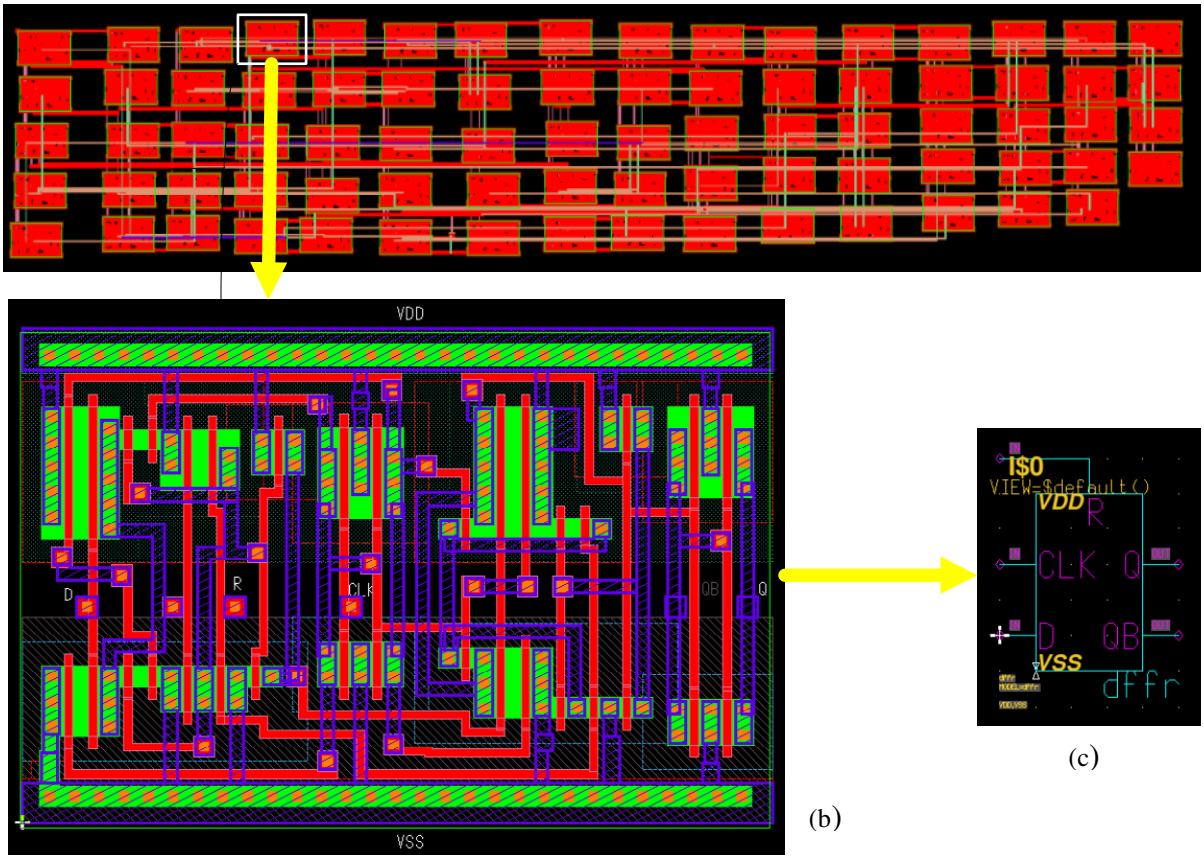
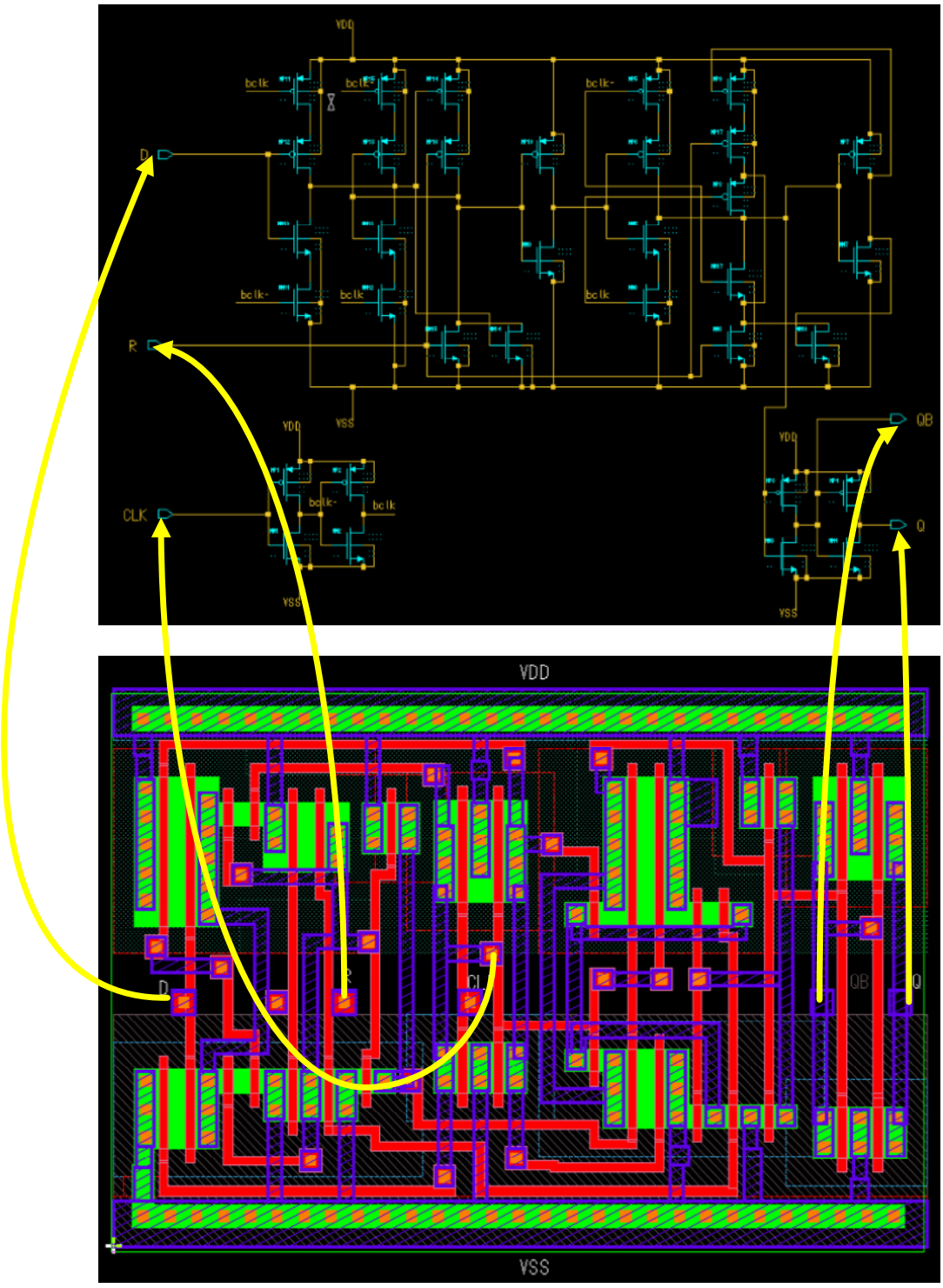


Figura 44- Leiaute completo do núcleo do 8051 (a), leiaute de um bloco que foi colocado em destaque (b) e representação elétrica do bloco em destaque (c).

Na Figura 44a é possível visualizar o leiaute completo do núcleo do 8051, onde é destacado um dos blocos, onde é detalhado o seu leiaute para fins de verifição se realmente ele foi gerado de acordo com o circuito elétrico, com as máscaras e com as regras de projeto da tecnologia CMOS de fabricação utilzada. Este bloco detalhado possui o leiaute apresentado na Figura 44b que apresenta uma grande quantidade de transistores e interconexões. Cada máscara utilizada esta destacada em cores diferentes. E, para finalizar, na figura 44c é apresentado o símbolo correspondente ao leiaute da Figura 44b. O símbolo trata-se de um *flip-flop* do tipo D que apresenta os sinais : D, CLK, R, Q e QB. A verificação do leiaute de acordo com o descrito acima foi feita com sucesso para este caso.

O *flip-flop* tipo D é um circuito eletrônico que realiza a função básica de memória, onde a cada sinal de *clock* gerado, o sinal que está na entrada D é transferido para a saída Q, onde permanece até que outro sinal de *clock* seja gerado. Além destas funções, caso a entrada de *reset* seja acionada, a saída Q é colocada em nível lógico 0 e a sua correspondendente saída complementar (QB), é colocada em nível lógico 1. Para que essas funções sejam realizadas, é necessário gerar um circuito eletrônico conforme apresenta a Figura 45.

(a)



(b)

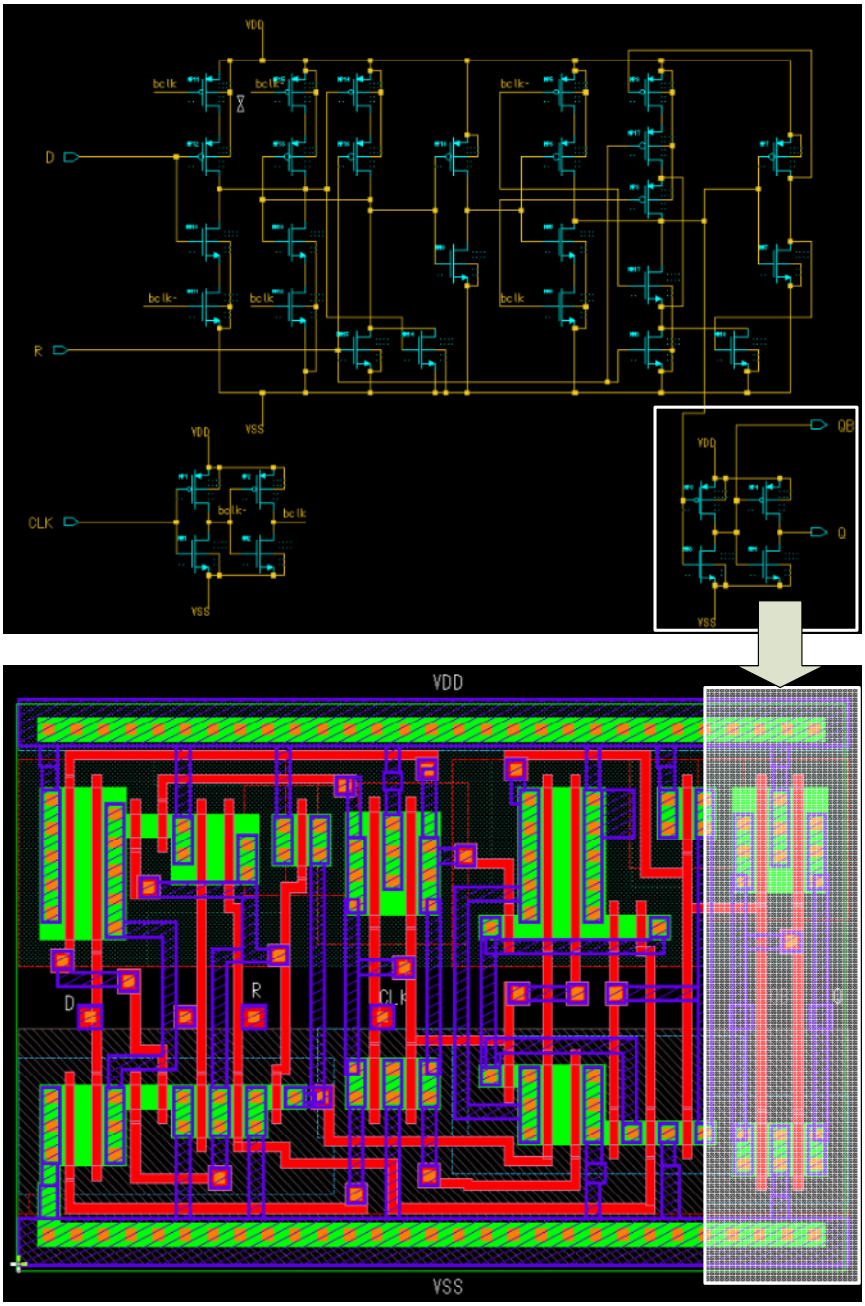
Figura 45- Esquemático do *flip-flop* do tipo D (a) e o correspondente leiaute com suas interconexões (b).

O circuito elétrico do *flip-flop* do tipo D, como apresentado na figura 45a, é composto por diversos transistores. Pode-se perceber que os mesmos sinais já descritos anteriormente na simbologia do *flip-flop* do tipo D (D, CLK, R, Q e QB) encontram-se tanto no circuito

esquemático (figura 45a) quanto no leiaute (figura 45b). Isso deve ser checado pelo projetista e neste caso foi feito com sucesso.

O leiaute, por vezes, apresenta dificuldades em sua análise pois utiliza de artifícios para obter uma economia de área em silício, o que o torna ainda mais complexo. Na Figura 46 são ilustrados o esquemático e o seu correspondente leiaute.

(a)



(b)

Figura 46- Esquemático do *flip-flop* do tipo D com estágio de saída em destaque (a), leiaute do *flip-flop* do tipo D com estágio de saída em destaque (b).

Na Figura 46a é possivel observar o esquemático do *flip-flop* do tipo D, onde estão em destaque quatro transitores que formam duas portas inversoras, que estão conectadas entre si, e são responsáveis pelas saídas do circuito (Q e QB). Na figura 46b estão em destaque, no leiaute, os mesmos quatro transistores do estágio de saída. Estes transistores são apresentados na Figura 47.

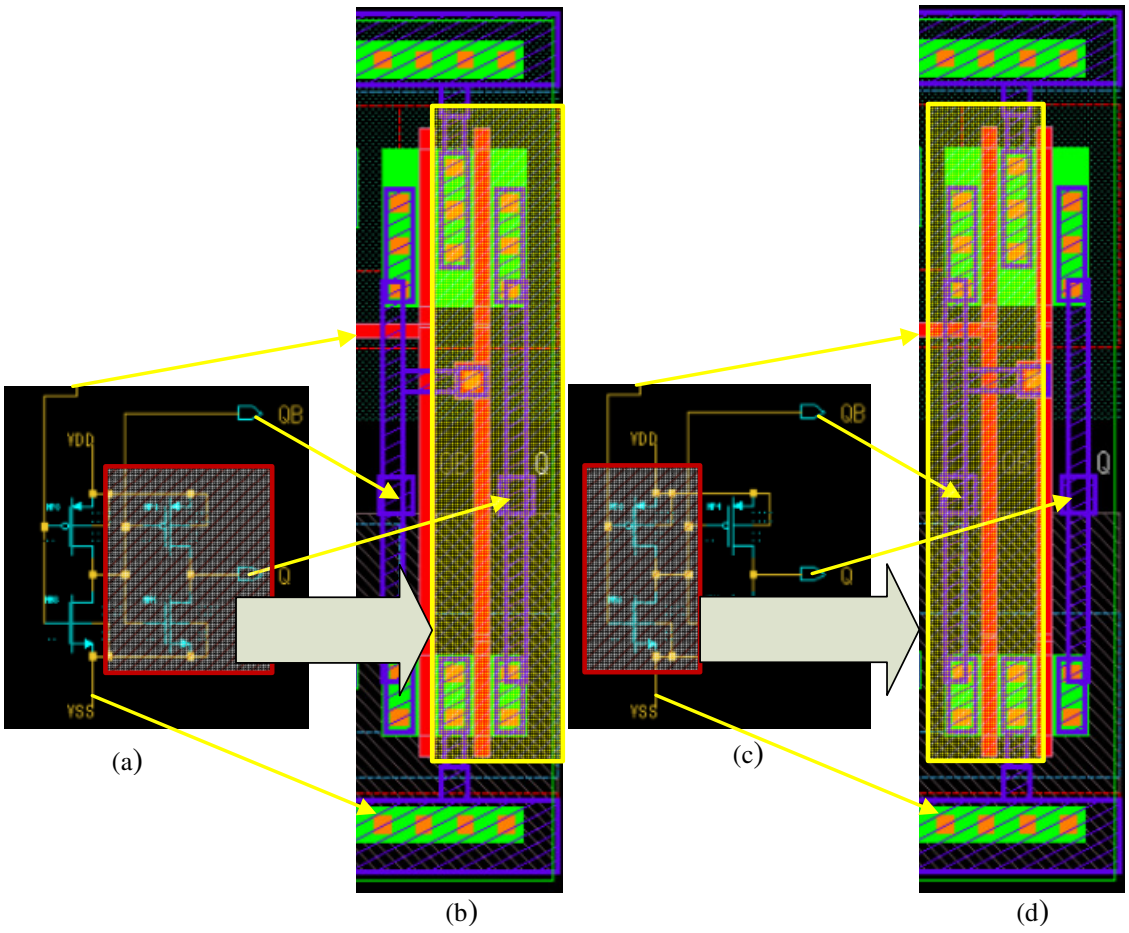
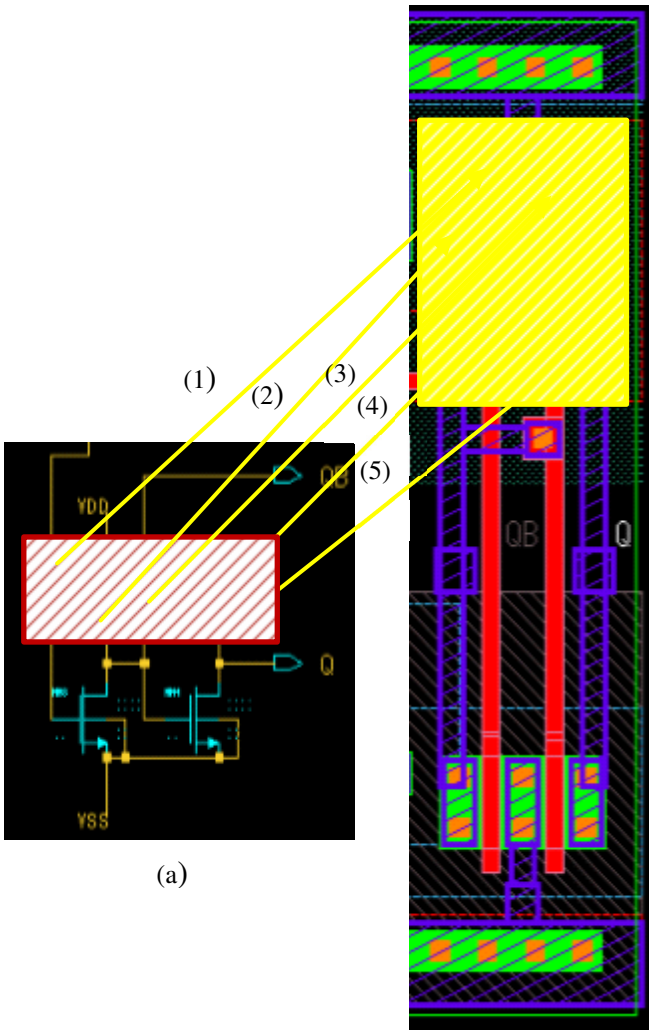


Figura 47- Esquemático do circuito inversor da saída Q (a) e o seu leiaute (b), esquemático do circuito inversor da saída QB (c) e o seu leiaute(d).

Para formar um circuito inversor são necessários dois transistores, sendo um deles do tipo PMOSFET e um do tipo NMOSFET. O transistor PMOSFET é responsável por fornecer o nível lógico 1 para a saída do inversor e o transistor NMOSFET é responsável por fornecer o nível lógico 0 para a saída do inversor. Na figura 5-11a estão em destaque dois transistores que formam um circuito inversor que é o responsável pelo sinal da saída Q. A figura 5-11b destaca o leiaute onde encontram-se esses transistores.

Na figura 5-11c temos os dois transistores que formam a segunda porta inversora, responsável pela saída QB, e seu respectivo leiaute com o posicionamento dos sinais são ilustrados na figura 5-11d.

Finalizando a análise do leiaute, na figura 5-12 está ilustrada a posição, no leiaute, de cada terminal do transistor.



(a)

(b)

Figura 48- Esquemático de dois circuitos inversores (a) e o leiaute de dois circuitos inversores (b).

Na figura 5-12a é apresentado um esquemático de um circuito composto de duas portas inversoras. Neste esquemático estão em destaque dois transistores PMOSFET, enquanto a figura 5-12b está ilustrado o leiaute destacando-se os mesmos transistores PMOSFET. Entre estas duas figuras estão indicados 5 pontos do esquemático e o seu respectivo posicionamento no leiaute. O primeiro ponto mostra o posicionamento do terminal de porta (*gate*) do primeiro transistor, onde no leiaute esta porta é constituída pelo polisilício, que está representado pela máscara de cor vermelha. O segundo indicador aponta para o terminal de dreno (*drain*) do primeiro transistor. Este terminal corresponde à saída QB do circuito, que no leiaute ele é representado pela máscara azul, que representa a máscara de metal 1, que é utilizada para realizar algumas interligações dentro do circuito integrado. O dreno deste mesmo transistor

além de ser o sinal da saída, também é utilizado como a entrada do segundo transistor (indicador 3).

O quarto indicador (4) é o terminal de fonte (*source*) de ambos os transistores, que no leiaute possui apenas um contato sendo compartilhado entre os dois transitores. Este compartilhamento de contatos entre transistores resulta numa maior economia de área de silício.

O último indicador (5) apresenta o posicionamento do terminal de dreno (*drain*) do segundo transistor, que é responsável pelo sinal de saída Q.

Este detalhamento pode ser realizado em todo o leiaute do núcleo do microcontrolador 8051, porém por se tratar de um leiaute com grande número de blocos lógicos é importante o projetista fazer tal verificação em pelo menos alguns blocos. Caso todos eles estejam de acordo com o esquemático e com as máscaras e as regras de projeto correspondente ao processo de fabricação CMOS de circuito integrados escolhido, a possibilidade dos demais estarem corretos são muito altas. Caso for encontrado algum problemas em um dos leiautes verificados, uma conferência completa do leiaute completo deve ser realizada, com o objetivo de detectar qual foi a causa raiz do problema. Com as informações obtidas das falhas ocorridas, o projeto deve ser revisto por completo.

6.5 Conclusão

No circuito utilizando ferramentas computacionais em versão profissional, não foi realizada a simulação do código VHDL, pois o código adotado no projeto foi disponibilizado pela Universidade da Califórnia. Para verificar as funcionalidades do projeto, foi realizada uma verificação física através de um *kit* didático.

Como o *kit* didático, disponibilizado pela FEI, é da fabricante Altera, o *software* utilizado para realizar todas as configurações necessárias para o teste foi o *Quartus II*, que disponibilizou uma enorme quantidades de recursos.

Para a sintetização do núcleo do 8051 foi utilizada a ferramenta *Leonardo Spectrum,* da fabricante *Mentor Graphics,* a versão utilizada não possuía interface gráfica, todos os comandos eram realizados via texto. O *Leornardo Spectrum* permite definir critérios de otimização do circuito e o resultado do processo de síntese é um arquivo em linguagem Verilog, que foi utilizado para a geração do esquemático.

Algumas dificuldades encontradas em relação à utilização do *Leornardo* é a escassez de material técnico e literaturas, principalmente no que diz respeito aos comandos utilizados.

O resultado do processo de síntese (um arquivo Verilog) é utilizado para a criação do esquemático do circuito, o *software* utilizado é o *Pyxis Schematic,* toda esta etapa é feita automaticamente, o projetista deve apenas criar o arquivo de mapeamento como já descrito nesta dissertação.

Com o esquemático pronto, inicia-se a etapa de leiaute, utilizando-se o *Pyxis Layout,* este *software* possui diversas ferramentas e recursos de edição de leiaute. O posicionamento de uma porta ou bloco lógico pode ser definido de maneira manual ou automática.

Após o leiaute ser realizado, o projetista pode alterá-lo conforme sua necessidade, o leiaute de cada projeto muda de acordo a biblioteca especificada no início do mesmo, pois cada biblioteca possui padrões de leiaute diferentes.

7 CONCLUSÕES

A área de projetos de circuitos integrados digitais passa por constantes mudanças a cada dia, através do uso de novos métodos de desenvolvimento de projetos de circuitos integrados digitais, através da constante criação de novas ferramentas computacionais para ajudar o projetista a desenvolver projetos de circuitos integrados digitais de forma mais rápida, eficiente e confiável.

Atualmente existem diversas ferramentas EDA disponível no mercado, que facilitam o trabalho de se desenvolver complexos projetos de circuitos integrados digitais desde a sua concepção até a implementação automática do seu leiaute.

Essa dissertação de mestrado focou em primeiro lugar na proposição de um fluxo de projeto semi-automático de desenvolvimento de projetos de circuitos integrados digitais por meio do uso de ferramentas computacionais (EDA) em suas versões de demonstração, principalmente através das ferramentas computacionais Quartus e Microwind. O objetivo maior dessa proposta é a formação de recursos humanos na área de projetos de circuitos integrados digitais para alunos de Instituições de ensino (a nível técnico, de graduação e de pos-graduação), que não possuem acesso às ferramentas computacionais (EDA) profissionais, que são bastante caras. Isso permite a esses estudantes terem acesso a tecnologia de desenvolvimento de circuitos integrados digitais desde a sua concepção até a implementação do leiaute, que define o conjunto de máscaras do processo de fabricação CMOS de circuitos integrados escolhido, isto é, permite que esses alunos também possam ter a possibilidade de fabricar circuitos integrados digitais.

Além disso, este trabalho implementou uma versão do microcontrolador 8051 a partir do seu código VHDL em FPGA. Dois programas teste foram implemtandos no 8051 em FPGA com sucesso, validando o VHDL do 8051 utilizado. Depois disso, o objetivo foi o de usar ferramentas computacionais profissionais da Mentor para a geração automática do seu leiaute a partir do VHDL do 8051, usando-se o fluxo de projetos de circuitos integrados da Mentor Graphics.

Com esses trabalhos realizados nessa dissertação de mestrado, hoje na FEI é possível conceber, gerar o VHDL correspondente a um determinado circuito integrado digital, validar o VHDL e obter o leiaute de forma automática com o uso das ferramentas da Mentor Graphics e fabricar circuitos integrados digitais complexos.

Como possíveis trabalhos futuros, pode-se recomendar:

I. Para o fluxo de projetos de circuitos integrados digitais utilizando ferramentas computacionais em versão de demonstração:
   a) A automatização do processo de geração do arquivo Verilog de entrada do *Microwind*, através de um *software* externo ou da inclusão de diferentes bibliotecas no *Quartus II*;
   b) A realização do estudo de novas ferramentas para a simulação pós-leiaute do circuito integrado digital.

II. Para o fluxo de projetos de circuitos integrados digitais utilizando ferramentas computacionais em versão profissional:
   a) O estudo das ferramentas *Mentor* de simulação pós-leiaute do circuito integrado digital;
   b) A realização de um projeto com o fluxo proposto nesta dissetação que contemple a fabricação do circuito integrado.

III. Para ambos os fluxos de projeto de circuitos integrados diitais:
   a) A fabricação de um circuito integrado digital com esses fluxos de projetos de circuitos integrados digitais.

REFERÊNCIAS

1 ITOH, K. A Historical Review of Low-Power, Low-Voltage Digital MOS Circuits Development. **IEEE SOLID-STATE CIRCUITS MAGAZINE**, 4 Fevereiro 2013.

2 BASS, M.; CHRISTENSEN, C. The Future of the Microprocessor Business. **IEEE SPECTRUM**, ABRIL 2002.

3 PEDRONI, V. **Eletrônica digital moderna com VHDL**. Rio de Janeiro: Elsevier, 2010.

4 MELO, W. R. D. **Estudo do Fluxo de Projeto de Circuitos Integrados Digitais de Aplicação Específica (ASICS) Aplicado a um CI Monitor de Velocidade**. 2004. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual de Campinas.

5 ROCHA, J.; MARTINS, J.; FERNANDES, M. **Projecto de Circuitos Integrados Digitais utilizando VHDL**. Instituto Superior de Engenharia de Lisboa. [S.l.].

6 GIELEN, G. G. E.; RUTENBAR, R. A. Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits. **PROCEEDINGS OF THE IEEE**, v. 88, December 2000.

7 TIRI, K.; VERBAUWHEDE, I. A Digital Design Flow for Secure Integrated Circuits. **IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS**, v. 25, n. 7, Julho 2006.

8 TANG, Y. et al. Vertical Integration of System-on-Chip Concepts in the Digital Design Curriculum. **IEEE TRANSACTIONS ON EDUCATION**, v. 54, Maio 2011.

9 LYONS, E. et al. Full-Custom Design Project for Digital VLSI and IC Design Courses using Synopsys Generic 90nm CMOS Library. **IEEE**, 2009. 4.

10 GIMENEZ, S. P. **Microcontroladores 8051:** teoria e prática. São Paulo: Érica, 2010.

11 AMORE, R. D. **VHDL:** descrição e síntese de circuitos digitais. Rio de Janeiro: LTC, 2005.

12 IEEE Standard VHDL Language Reference Manual, Disponivel em < ieeexplore.ieee.org> Acesso em: 04 nov 2014.

13 ITAJUBÁ, U. F. D. **Tutorial VHDL**. UNIFEI. [S.l.].

14 NICOLOSI, D. E. C. **Microcontrolador 8051 Detalhado**. São Paulo: Érica, 2000.

15 MUSSOLINI, T. P. **Desenvolviento de um 8051 com linguagem VHDL**. UNIFEI. Itajubá. 2011.

16 SICARD, E. Microwind. Disponivel em: <www.microwind.org>. Acesso em: 05 abril 2014.

17 ALTERA. Download Center. **Site da Altera**. Disponivel em: <https://www.altera.com/download/sw/dnl-sw-index.jsp>. Acesso em: Setembro 2012.

18 VAHID, F. Dalton Projetc. **University of California**, 2001. Disponivel em: <http://www.cs.ucr.edu/~dalton/i8051/i8051syn/>. Acesso em: 17 Setembro 2012.

19 CHAVES, C. R.; CANTERO, R. J.; BORGES, R. A. Sobre o projeto PEQui. **Site do PEQui Universidade Federal de Goias**, 2005. Disponivel em: <http://www.geocities.ws/sim8051/>. Acesso em: 25 novembro 2012.

20 TECHNOLOGIES, T. **DE2-70 User Manual**. Taiwan: [s.n.], 2009.

21 CORPORATION, M. G. Site da Mentor Graphics. **High Education Program**. Disponivel em: <http://mentor.com/company/higher_ed/>. Acesso em: 23 maio 2013.

22 GRAPHICS, M. **Mentor Graphics ASIC Design Kit 3.1**. Oregon - Estados Unidos: [s.n.], 2005.

23 CORPORATION, M. G. **Generic Design Kit (GDK)**. Oregon - Estados Unidos: [s.n.], 2013.

24 GRAPHICS, M. **LeonardoSpectrum HDL Synthesis Manual**. Oregon - Estados Unidos: [s.n.], 2012.

APÊNDICE A – TUTORIAL DE UTILIZAÇÃO DO SOFTWARE *QUARTUS II* PARA CRIAÇÃO DE BLOCOS VHDL, SINTETIZAÇÃO E SIMULAÇÃO DE PROJETOS DIGITAIS

O primeiro passo, para criação do código VHDL do projeto digital é abrir o *software* Quartus, previamente instalado. Ao iniciar o *software* Quartus será aberta uma tela conforme a demonstrada na Figura 49.

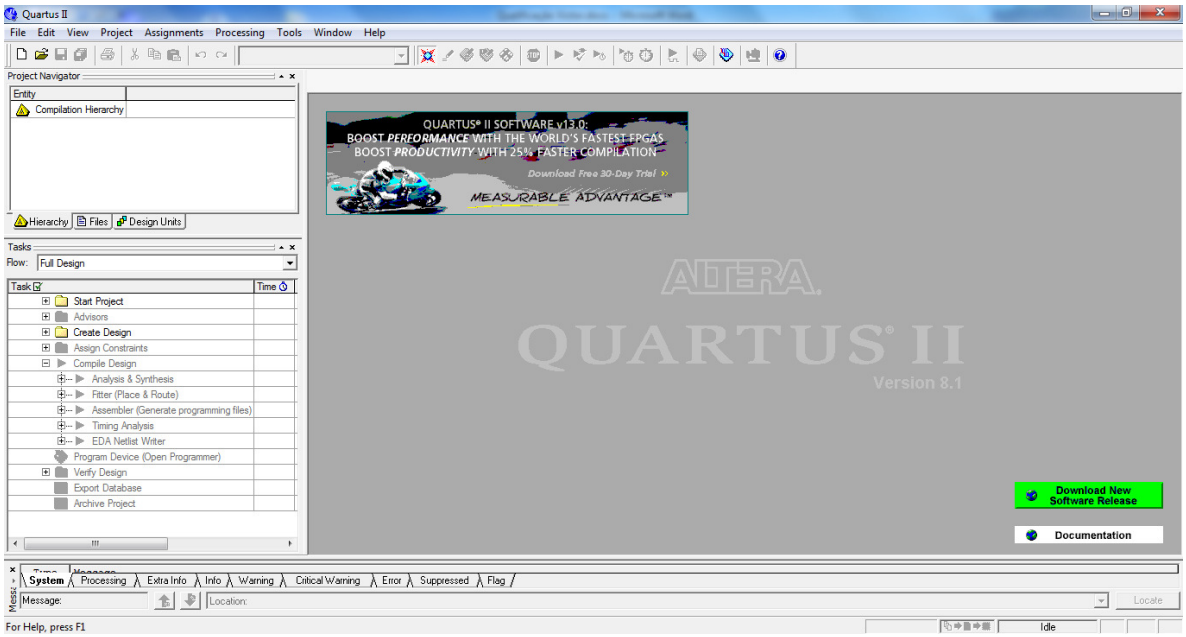


Figura 49- Tela inicial do Software Quartus.

A tela inicial do Quartus é composta por diversas barras de menus e sub-janelas, similar a um *software* que siga o padrão do sistema operacional *Windows.* Para iniciar a descrição do código VHDL, como primeiro passo é necessário à criação de um projeto. Portanto na barra de menu “File” , deve-se clicar em “New Project Wizard”.

Ao iniciar a criação do projeto, é aberto uma sucessão de janelas, que serão necessárias para configuração inicial do projeto. A primeira tela que se abre (Figura 50), é a tela onde se definirá o local onde este projeto será armazenado e o nome de seu projeto. Um detalhe bastante importante é que, como regra, o nome da entidade de seu código VHDL, tenha o mesmo nome do seu projeto, por isso o Quartus define de maneira automática, o mesmo nome da entidade do código VHDL.

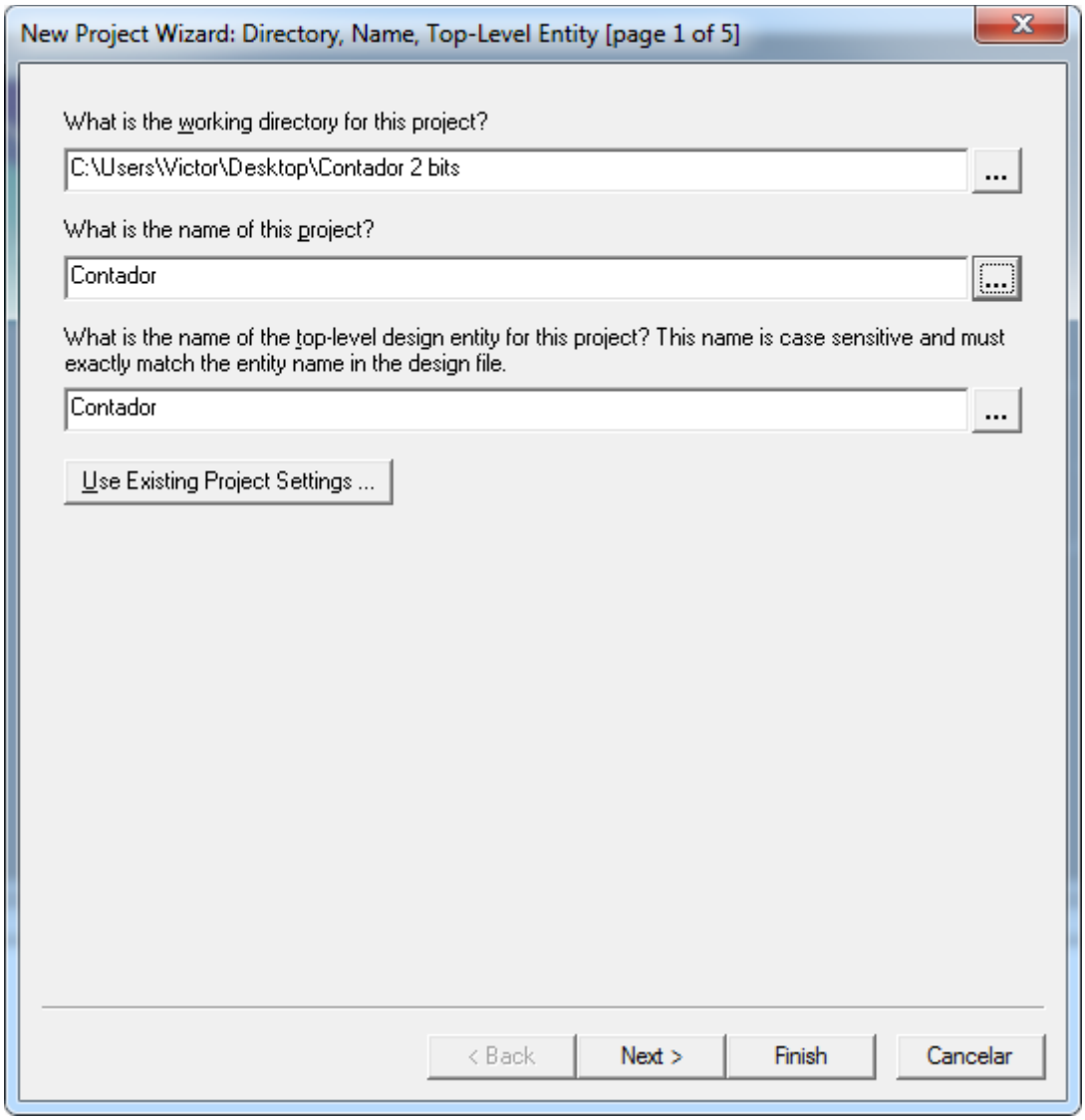


Figura 50- Tela de configuração de projeto.

Após esta etapa inicial, clicando-se no botão “ Next >”, o *software* abre uma nova janela (Figura 51), onde possibilita ao usuário incluir a este projeto, arquivos já existentes, visando agilizar a criação do projeto digital. No caso do Contador, não é incluído nenhum arquivo nesta etapa.

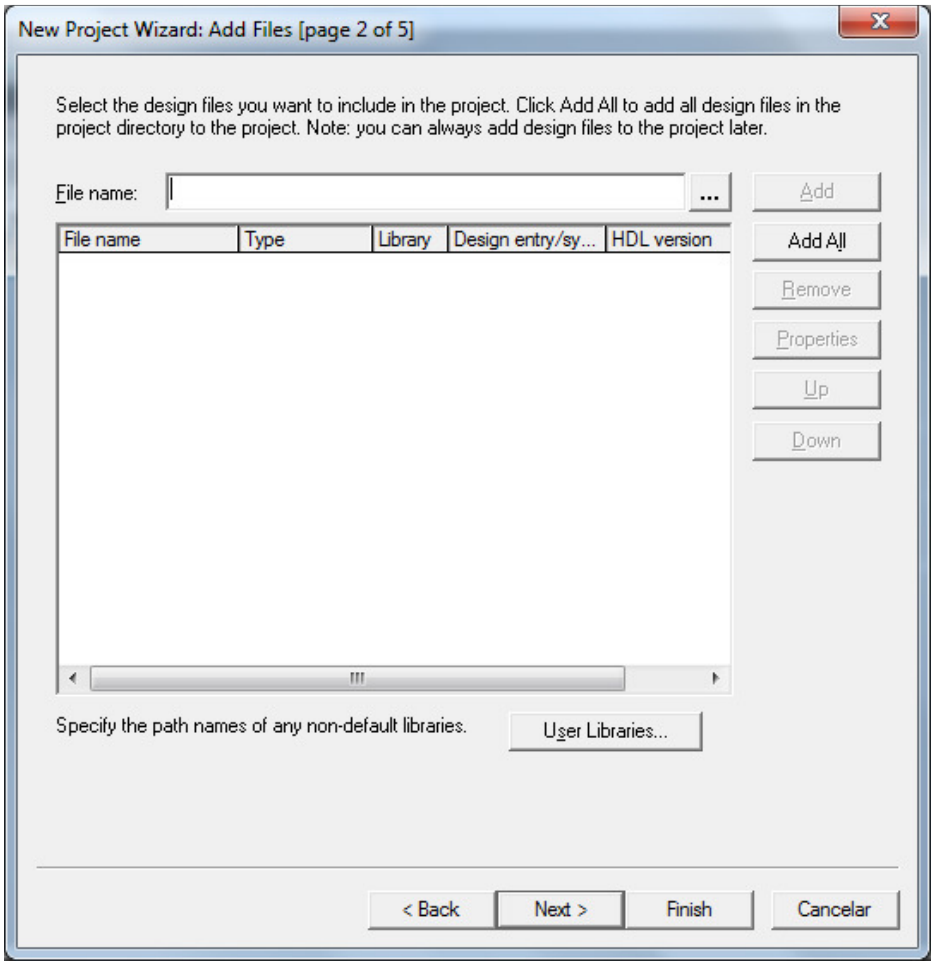


Figura 51- Tela de inclusão de arquivos de projetos.

Posteriormente é necessário fazer a escolha da família de FPGA a ser utilizada (Figura 52). Este projeto não é testado em *hardware*, por esse motivo foi selecionando um FPGA com grande capacidade de dispositivos, que supere a real necessidade do Contador. Caso nesta etapa seja escolhida uma família de FPGA, que não suporte o projeto em questão, será gerado um erro no momento da compilação.

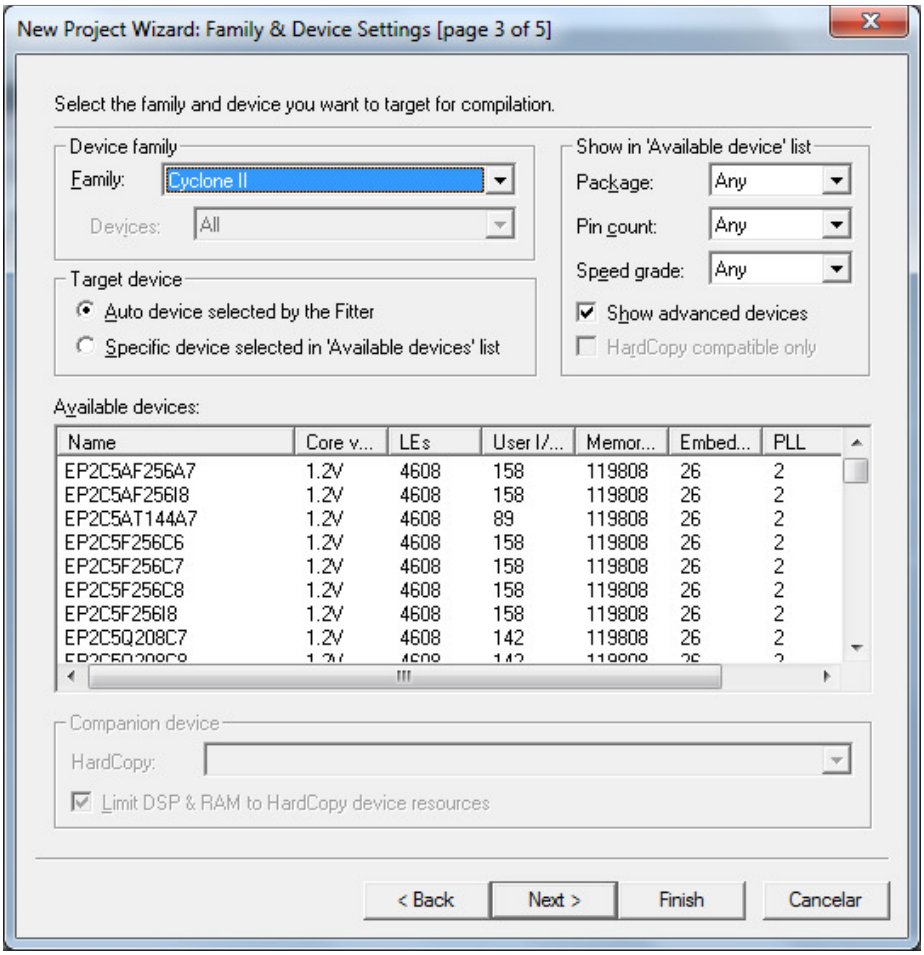


Figura 52- Seleção de Familia de FPGA.

Para este projeto, a etapa de seleção da família do FPGA será a etapa final da criação de projeto no Quartus e portanto após realizar esta seleção, pode-se clicar no botão “Finish” , para concluir a criação do projeto.

Após o projeto ter sido criado, o próximo passo, é a criação do arquivo VHDL, onde será descrito o projeto. Para a criação do arquivo VHDL, pode-se clicar na barra de menu principal em “File” => “New”, que resultará na abertura de uma nova janela (Figura 53), onde pode-se selecionar qualquer tipo de arquivo suportado pelo Quartus. Neste momento será selecionado o arquivo “VHDL File”.

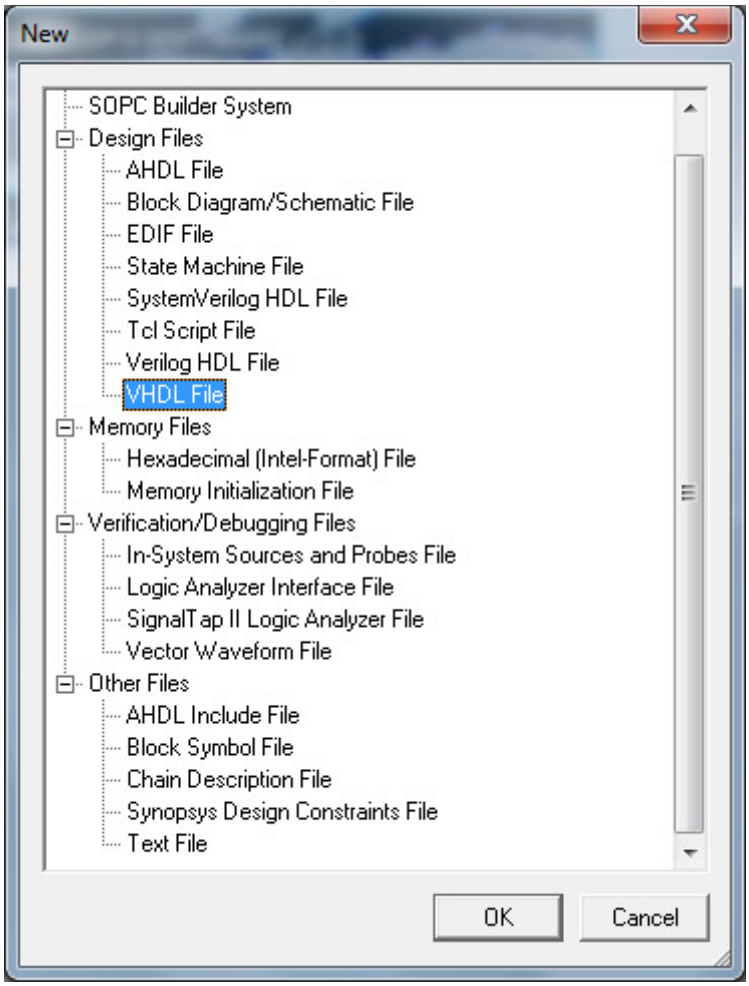


Figura 53- Criação de um novo arquivo

Após selecionar o tipo de arquivo desejado, basta clicar em “OK”, para que o arquivo selecionado seja aberto, na Figura 54 pode-se ver como deve ficar a tela principal do Quartus, para que possa ser realizada a programação VHDL.

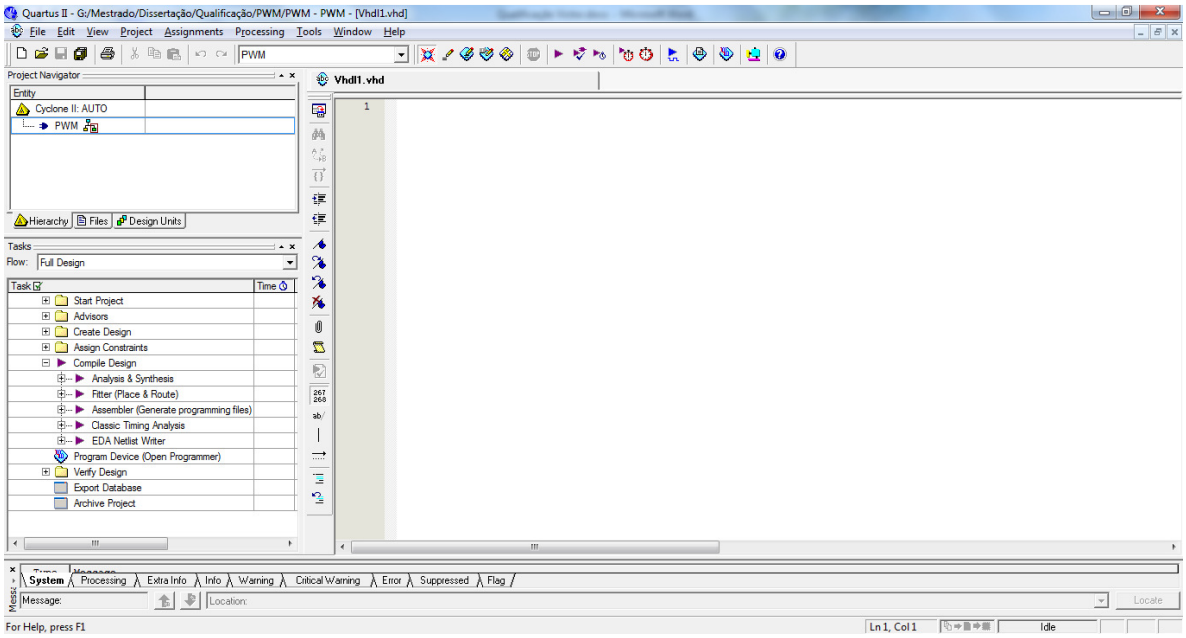


Figura 54- Tela e criação do VHDL.

Repare que foi aberta na tela principal, uma página em branco, que será utilizada para editar o código VHDL.

O código VHDL que descreve o projeto do Contador é apresentado na Figura 55. Este programa deve ser digitado nesta tela, que foi aberta no Quartus, com o intuito de integrar o projeto.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE ieee.std_logic_unsigned.all ;
4
5   ENTITY Contador IS      --define Contador com a entidade
6
7       PORT ( Clock, Reset: IN STD_LOGIC ; --define Clock e Reset com entrada
8
9           Q : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)) ; --define Q com uma vetor de saída composto de
10                                                   --dois bits
11      END Contador ;
12  ARCHITECTURE Projeto OF Contador IS
13
14      SIGNAL Count : STD_LOGIC_VECTOR (1 DOWNTO 0) ; --define Count como uma variavel que
15                                                     --trabalhará com o os valores de 0 a 2
16          BEGIN
17              PROCESS ( Clock, Reset )
18              BEGIN
19                  IF Reset = '0' THEN     -- Se o pino de Reset estiver em nível lógico 0
20                      Count <= "00" ; -- Count receberá o valor 00
21              ELSIF (Clock'EVENT AND Clock = '1') THEN --Senão, a cada evento de borda de subida do Clock
22                      Count <= Count + 1 ;    -- Count é incrmentado
23              END IF ;
24              END PROCESS ;
25          Q <= Count ; -- A saída Q recebe o valor de Count
26  END Projeto ;
27
```

Figura 55- Código VHDL comentado do Circuito Contador.

A próxima etapa a ser realizada é a compilação do código VHDL, mas antes de realizar a compilação, é necessário integrar o código VHDL ao projeto, primeira etapa para este

procedimento é salvar o arquivo VHDL clicando-se no menu “File => Save As”, escolhe-se um diretório, onde será salvo o arquivo e depois finaliza-se a operação.

Tendo o arquivo salvo, automaticamente ele integrará o seu projeto. Para ter certeza que ele faz parte do seu projeto, pode-se visualizá-lo na aba lateral (*Project Navigator*), se o arquivo VHDL está incluso. Na Figura 56 podemos ver que o Arquivo Contador1.vhd faz parte do projeto.

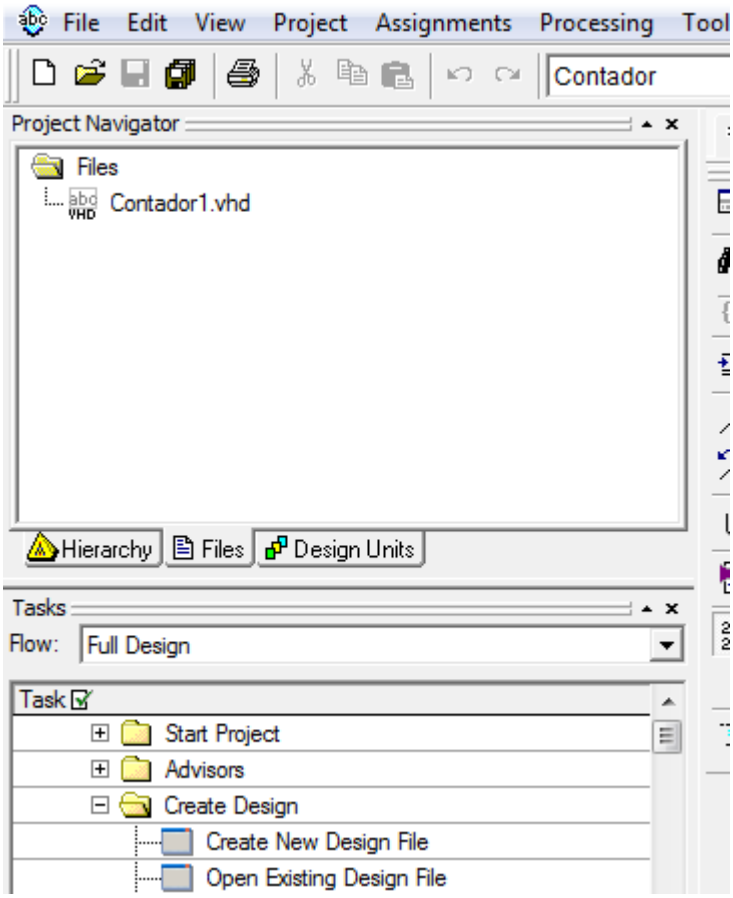


Figura 56- Project Navigator.

Com o código que acabou de ser incluso ao projeto, pode-se realizar a compilação, que tem como uma de suas funções, a análise sintática do código VHDL. Caso alguma instrução não esteja de acordo com o padrão VHDL, o compilador acusa um erro, e para o processo, que só prosseguirá após sua correção. Quando todos os erros forem corrigidos, o compilador mostra uma mensagem de sucesso e permitirá a realização da próxima etapa.

Para iniciar a compilação, basta clicar no menu “Processing => Start Compilation” (Figura 57).

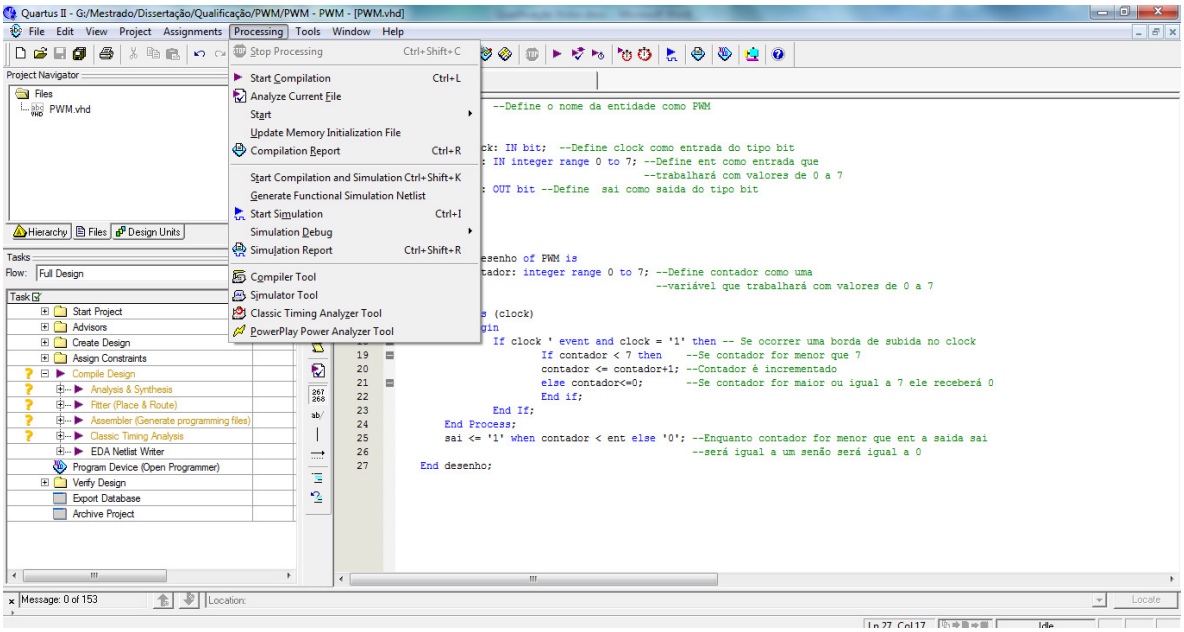


Figura 57- Iniciando a Compilação.

Neste instante, o Quartus começa analisar o seu Código VHDL. Caso a compilação seja realizada completamente, e não seja acusado nenhum erro, a mensagem de compilação realizada com sucesso (Figura 58) irá ser mostrada. Agora, como já foi dito anteriormente, caso seja detectado algum erro, o processo de compilação será encerrado, o erro deve ser visualizado e corrigido e a etapa de compilação deve ser reiniciada. Quando todos os erros forem solucionandos e a compilação for realizada com sucesso, esta etapa do fluxo de projeto estará concluída.

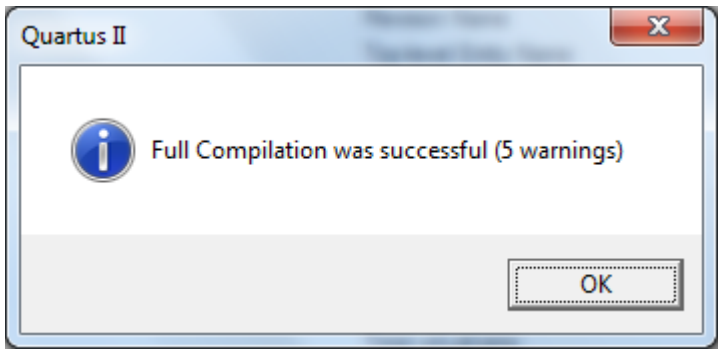


Figura 58- Mensagem de compilação realizada com sucesso

Após ser gerado e compilado o código VHDL, é iniciado o processo de síntese. O processo de síntese consiste numa tradução de um código VHDL para uma estrutura de *hardware*, que por sua vez deve realizar todas as funcionalidades do projeto [3]. Para a realização da síntese, foi utilizado o próprio Quartus II.

Para iniciar o processo de síntese, no menu principal, deve-se clicar nesta sequência de comandos, “File => Create/Update => Create Symbol Files for Current File” (Figura 59).

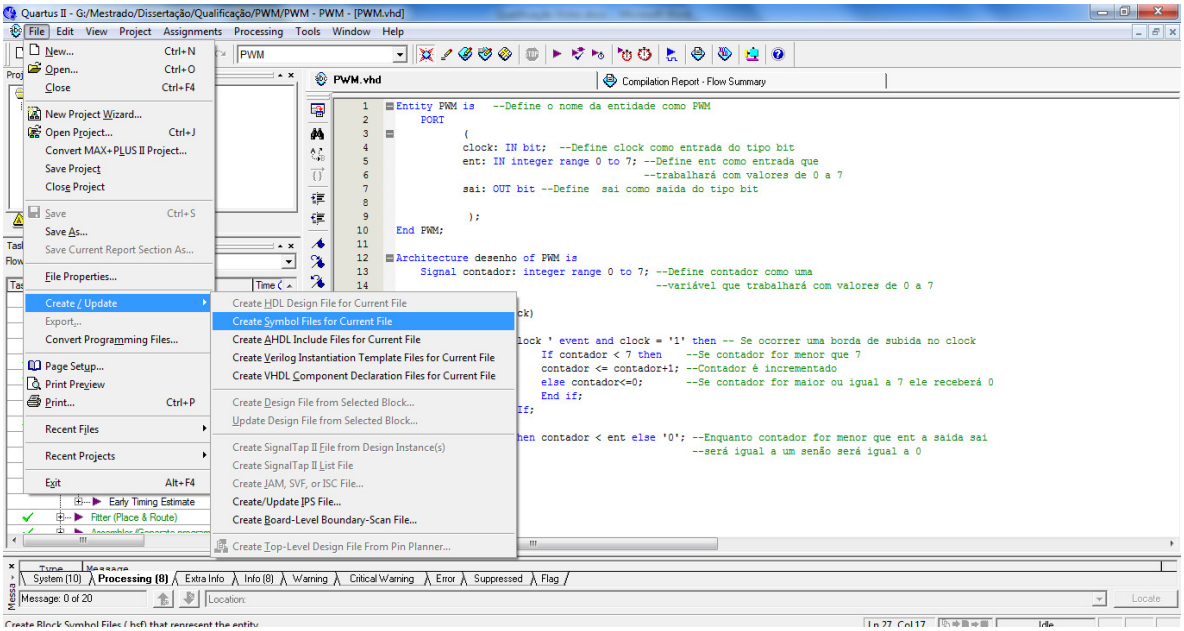


Figura 59- Criação do Arquivo de síntese.

Este procedimento tem como função criar o circuito lógico digital que descrevera o projeto do contador de 2 bits. Depois de alguns instantes surge, a mensagem que o arquivo foi criado com sucesso (Figura 60).

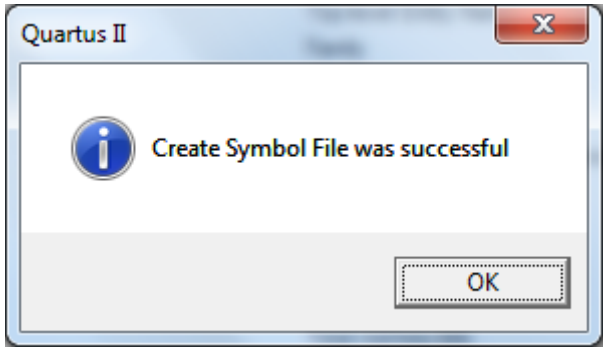


Figura 60- Mensagem de criação do arquivo de símbolos.

Depois de criado este arquivo, é preciso visualizá-lo. Para que seja possível a visualização do circuito lógico digital, é necessário abrir o arquivo RTL Viewer através do Menu “Tools => NetList Viewer => RTL Viewer ” ,conforme mostrado na Figura 61.

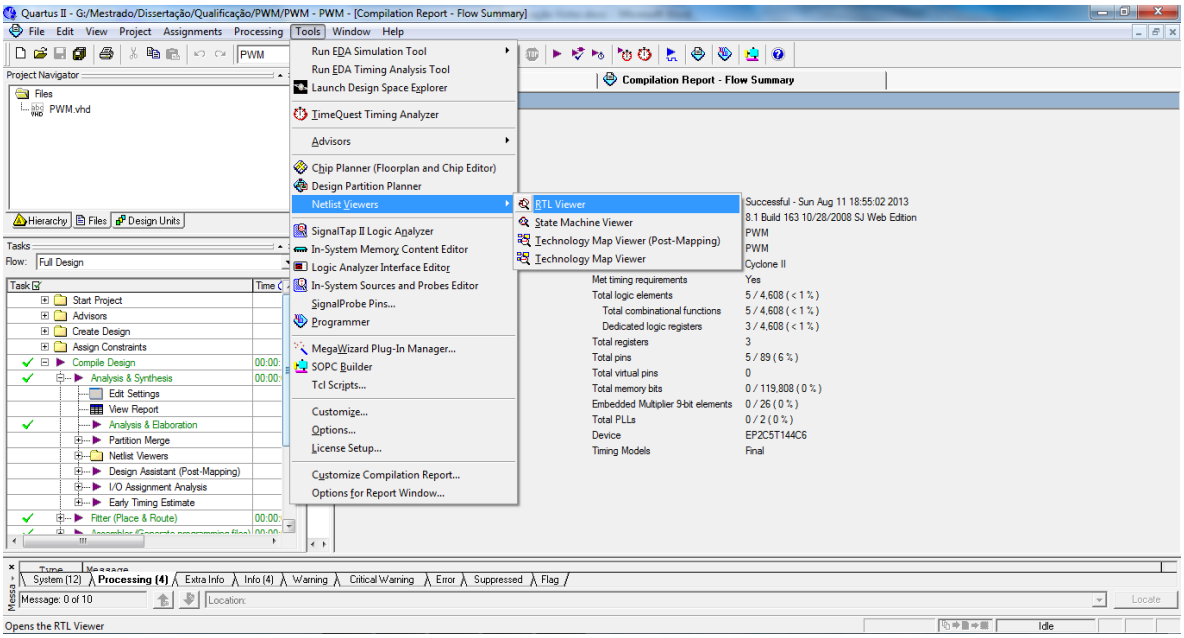


Figura 61- RTL Viewer.

A Figura 62 apresenta o resultado do processo de síntese, ou seja, o circuito digital do Contador desejado.

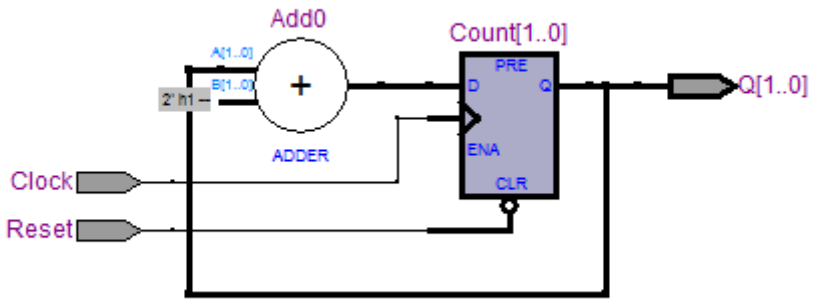


Figura 62- Hardware do Contador

A ferramenta Quartus nos permite fragmentar alguns destes blocos.

Para fragmentar os blocos lógicos, em blocos mais detalhados, é necessário selecionar o bloco lógico, como por exemplo o bloco do contador, e dessa forma basta clicar sobre o ele, e com o botão direito do *mouse,* selecionar a função “Ungroup Selected Nodes” (Figura 63), um detalhe importante é que, sempre que for possível realizar esta descompactação, esta função estará habilitada sobre o bloco lógico, caso ela não esteja, significa que o bloco não pode ser descompactado.

Para uma melhor visualização do circuito do contador de pulsos esse processo deve ser feito e a Figura 64 ilustra o diagrama de blocos com maiores detalhes.

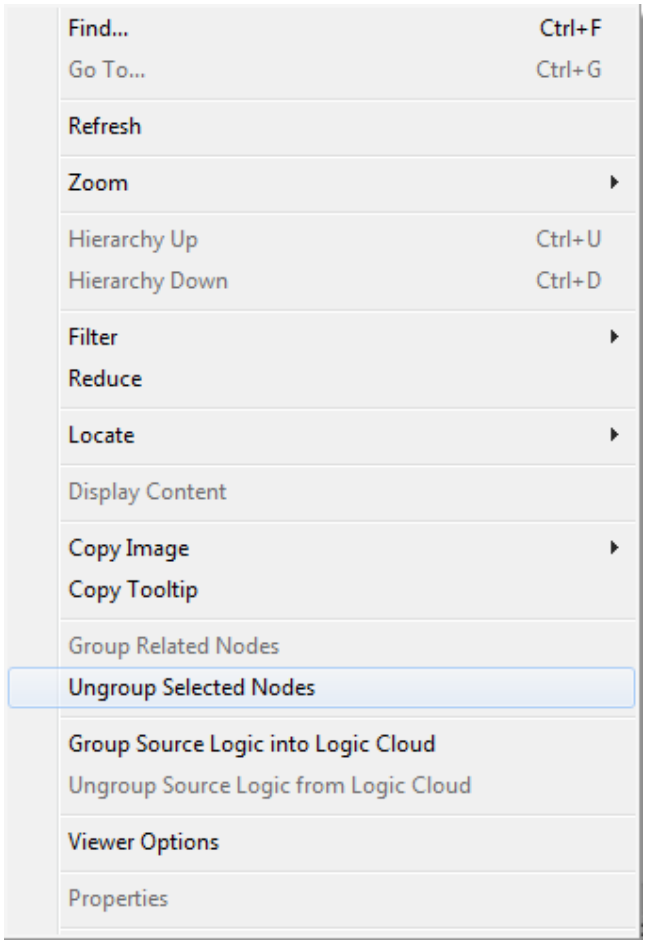


Figura 63- Descompactação do Bloco Lógico.

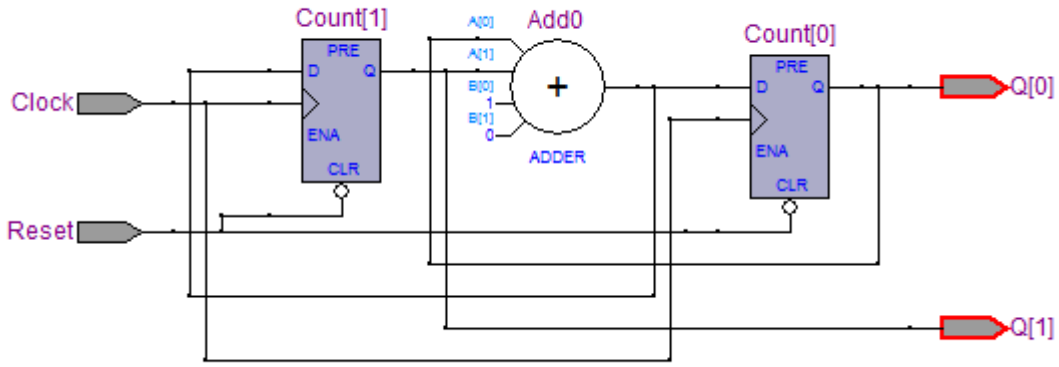


Figura 64- Diagrama de bloco detalhado do cicuito Contador de Pulsos.

Neste diagrama de bloco, que é apresentado na Figura 64 pode-se ver, de uma maneira mais detalhada, as conexões entre cada bloco, o que facilita a confecção do esquema elétrico final. Outra função que foi utilizada no Quartus, é a geração das expressões booleanas que define o projeto do circuito, clicando no menu “Processing” =>“Start” => “Start Equation Writer” conforme é mostrado a figura 65.

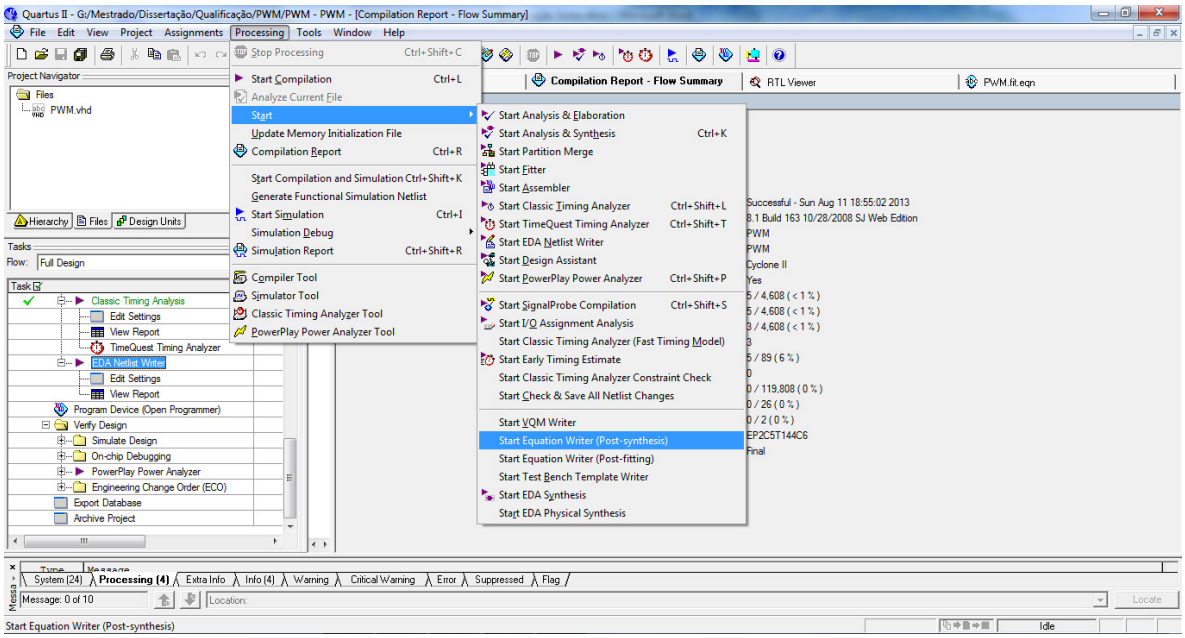


Figura 65- Geração das equações booleanas.

Ao término desta ação, será apresentado uma mensagem, indicando que as equações foram geradas com sucesso. Para visualiza-las, basta abrir o arquivo com a terminação “.eqn”,

que estará localizado na pasta em que foi definido o projeto do circuito Contador. As expressões apresentadas do projeto do Contador são mostradas na Figura 66.

```
Count[0] = DFFEAS(A1L5, GLOBAL(A1L2), GLOBAL(A1L12),  ,  ,  ,  ,  ,  );


--Count[1] is Count[1] at LCFF_X1_Y26_N23
Count[1] = DFFEAS(A1L7, GLOBAL(A1L2), GLOBAL(A1L12),  ,  ,  ,  ,  ,  );


--A1L7 is Count[1]~9 at LCCOMB_X1_Y26_N22
A1L7 = !Count[1] & Count[0] # Count[1] & !Count[0];


--A1L5 is Count[0]~10 at LCCOMB_X1_Y26_N0
A1L5 = !Count[0];


--Q[0] is Q[0] at PIN_D18
--operation mode is output

Q[0] = OUTPUT(Count[0]);


--Q[1] is Q[1] at PIN_E20
--operation mode is output

Q[1] = OUTPUT(Count[1]);


--Clock is Clock at PIN_N20
--operation mode is input

Clock = INPUT();


--Reset is Reset at PIN_M21
--operation mode is input

Reset = INPUT();
```

Figura 66- Expressões Booleanas do circuito contador.

Estas expressões remetem a nomes que foram utilizados nos diagramas de blocos mostrados anteriormente, e definem novas relações entre os sinais.

Realizando-se a interpretação destas expressões, juntamente com as conexões já estabelecidas pelos diagramas de blocos, é possível definir o circuito do Contador de uma maneira geral. Após esta análise, foi confeccionado um circuito que demonstrasse em nível de porta lógica e *flip-flops,* toda a estrutura do circuito. Para facilitar a visualização foi criado,

utilizando um *software* de simulação eletrônica, um esquemático geral do projeto que serve como base para as próximas etapas. Este diagrama é apresentado na Figura 67.

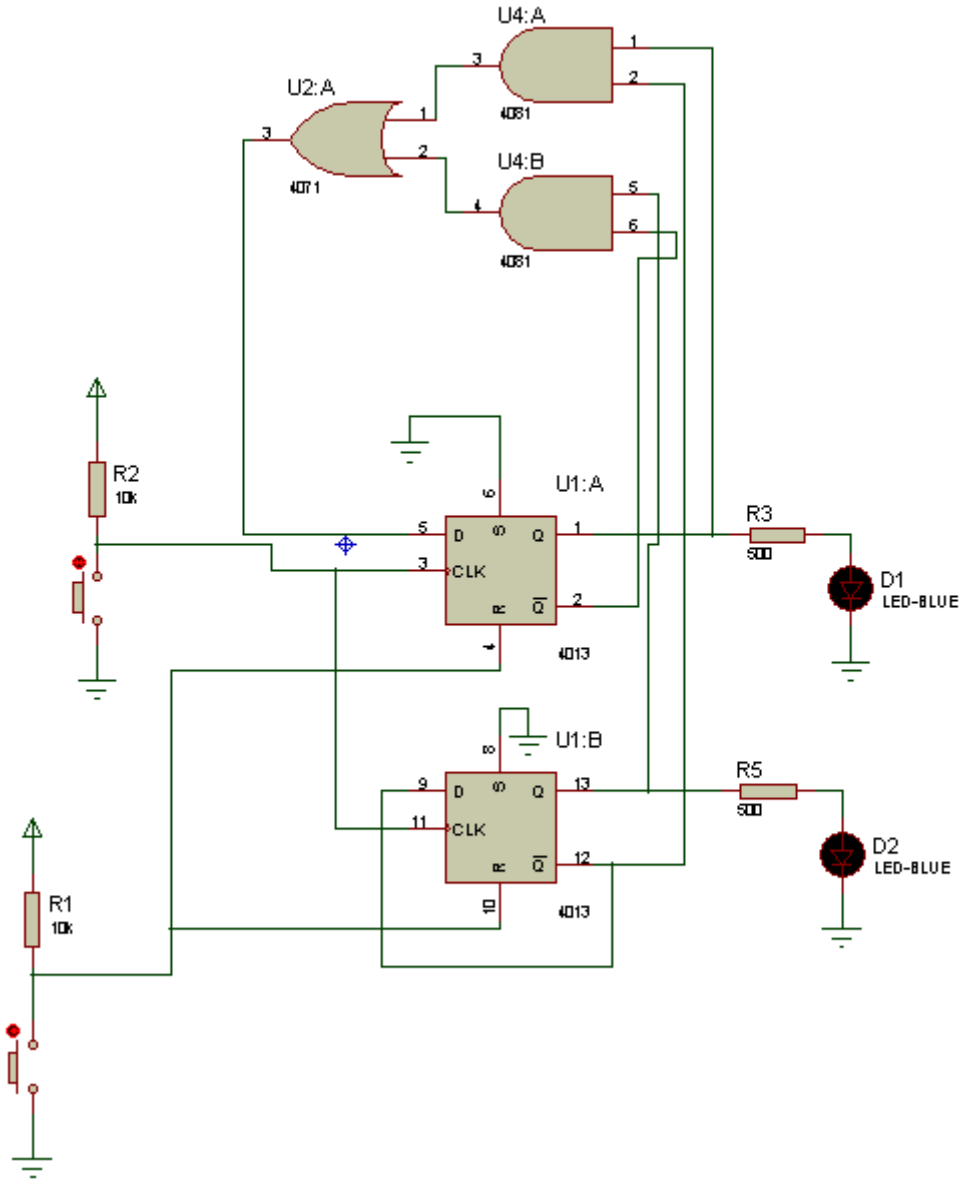


Figura 67- Circuito eletrônico do Contador.

Após toda a etapa de síntese, inicia-se a primeira etapa de simulação, onde será checado se a especificação do projeto foi atendida pela descrição VHDL. Esta simulação tem um caráter funcional, ou seja, será analisado se o funcionamento do projeto está adequado as suas especificações.

Para iniciar a simulação, abriremos a ferramenta *Simulator Tool,* localizada no menu "Processing", conforme a Figura 68.

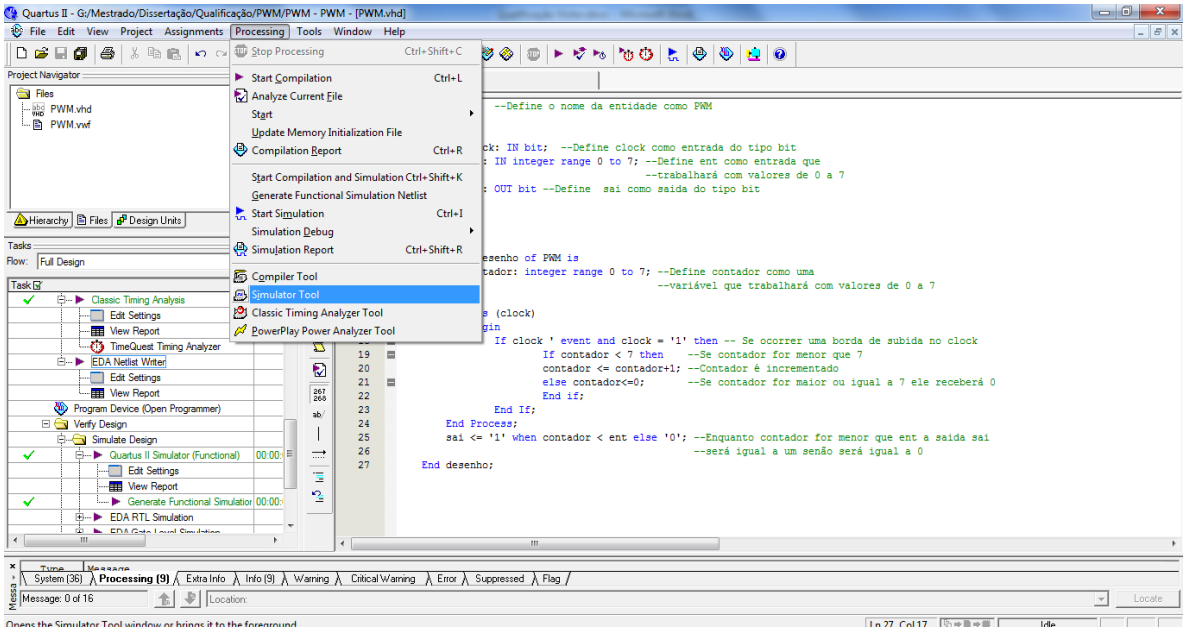


Figura 68- Iniciando a ferramenta de simulação.

Esta ferramenta abrirá a janela mostrada na Figura 69.

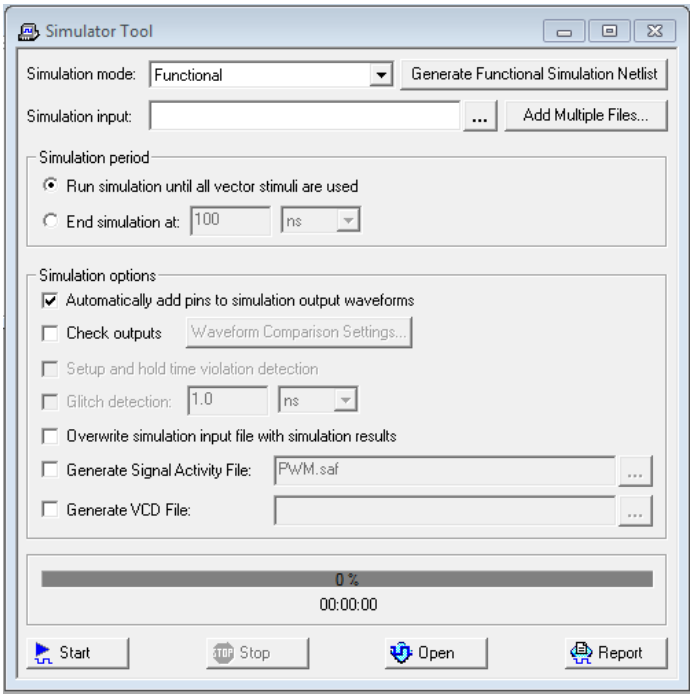


Figura 69- Simulator Tool.

Nesta tela da ferramenta de simulação, pode-se selecionar dois tipos de simulação: a simulação funcional ou a de tempo. No projeto do Contador, a simulação escolhida será a

funcional. Depois de selecionado este tipo de simulação na caixa "*Simulation mode*" deve-se criar o arquivo de *Netlist* da simulação funcional, apenas clicando no botão "*Generate Functional Simulation Netlist*".

Para que a simulação ocorra, é necessário ter um arquivo de estímulos, que não é nada mais que, um arquivo onde estará especificado os sinais de entrada e seus respectivos valores, e os sinais de saída. Caso este arquivo já tivesse sido criado, ele poderia ser selecionado na caixa *Simulation input*, neste projeto, este arquivo ainda será gerado, por esse motivo este item não precisa ser selecionado.

A criação deste arquivo de estímulos pode ser feita clicando no botão "Open", na parte inferior da tela. Ao clicar neste botão se abre um arquivo de *waveform,* onde poderão ser incluídas as formas de onda que serão utilizadas na simulação, seja ela uma forma de onda de entrada ou saída. Esta tela é mostrada na Figura 70.

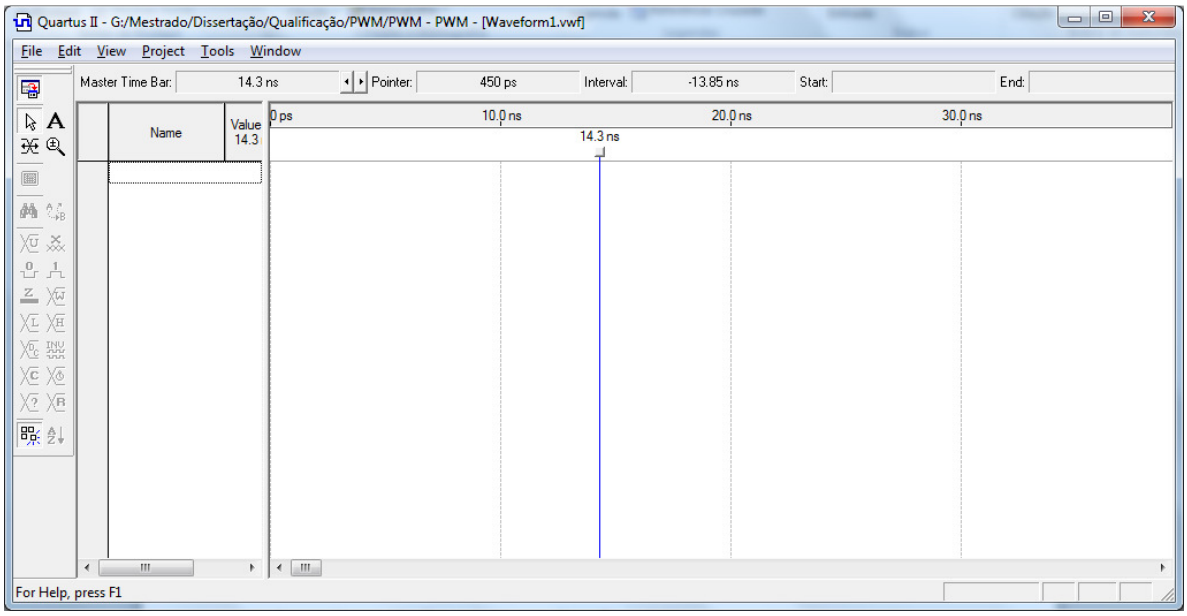


Figura 70- Janela dos sinais de estímulos para a realização da simulação.

Nesta tela, conforme ilustrada na Figura 70, não existe nenhuma forma de onda, nem sinal ainda incluído no projeto. Para realizar a inclusão desses sinais, basta dar um duplo *click* com o *mouse* sobre aba esquerda, logo abaixo da coluna *Name*. Realizando este procedimento abre-se uma janela, conforme é mostrado na Figura 71, onde pode-se incluir os sinais de entrada e saída.

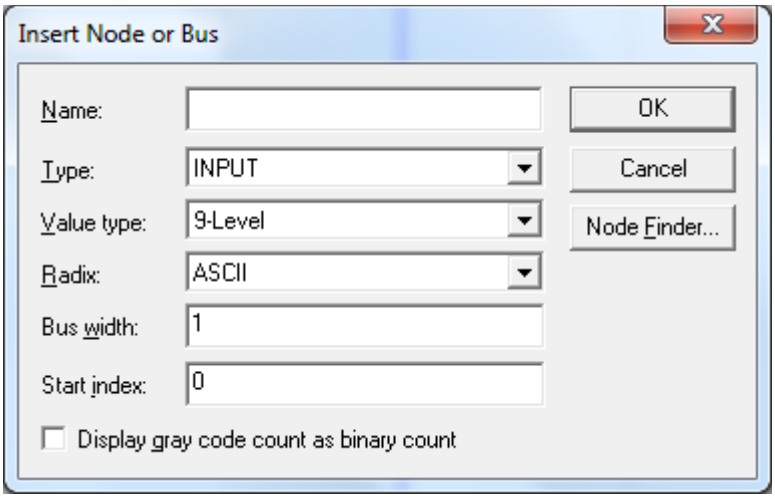


Figura 71- Tela de inserção de sinais.

É possível realizar a inserção dos sinais de maneira manual, escrevendo-se o nome dos sinais, o seu tipo e as demais informações conforme estão apresentadas na janela da Figura 71, ou pode-se realizar uma procura pelos sinais de seu projeto e a sua inclusão de uma maneira automática, bastando para isso clicar no botão *Node Finder*. Fazendo isso, abre-se a janela da Figura 72, onde é possível fazer uma busca dos sinais relacionados a entidade de seu projeto, onde todas as informações serão importadas do seu código-fonte VHDL.

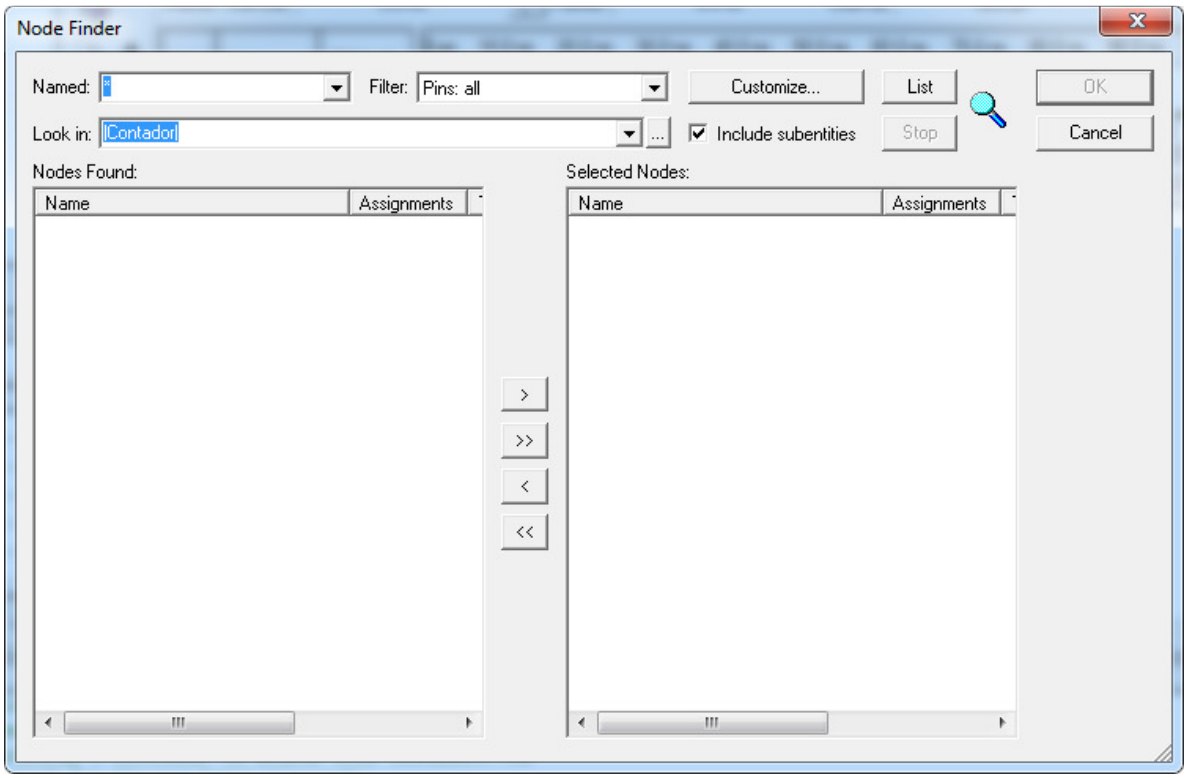


Figura 72- Tela de inclusão de sinais para simulação.

Pode-se notar que nesta janela já esta selecionada a entidade Contador (definida no código VHDL), deste modo, basta clicar no botão “List”, para que o *software* resgate todos os sinais presentes no código VHDL, e apresente-os de maneira que possa ser selecionado, no arquivo de simulação. Este procedimento resulta na imagem mostrada Figura 73.

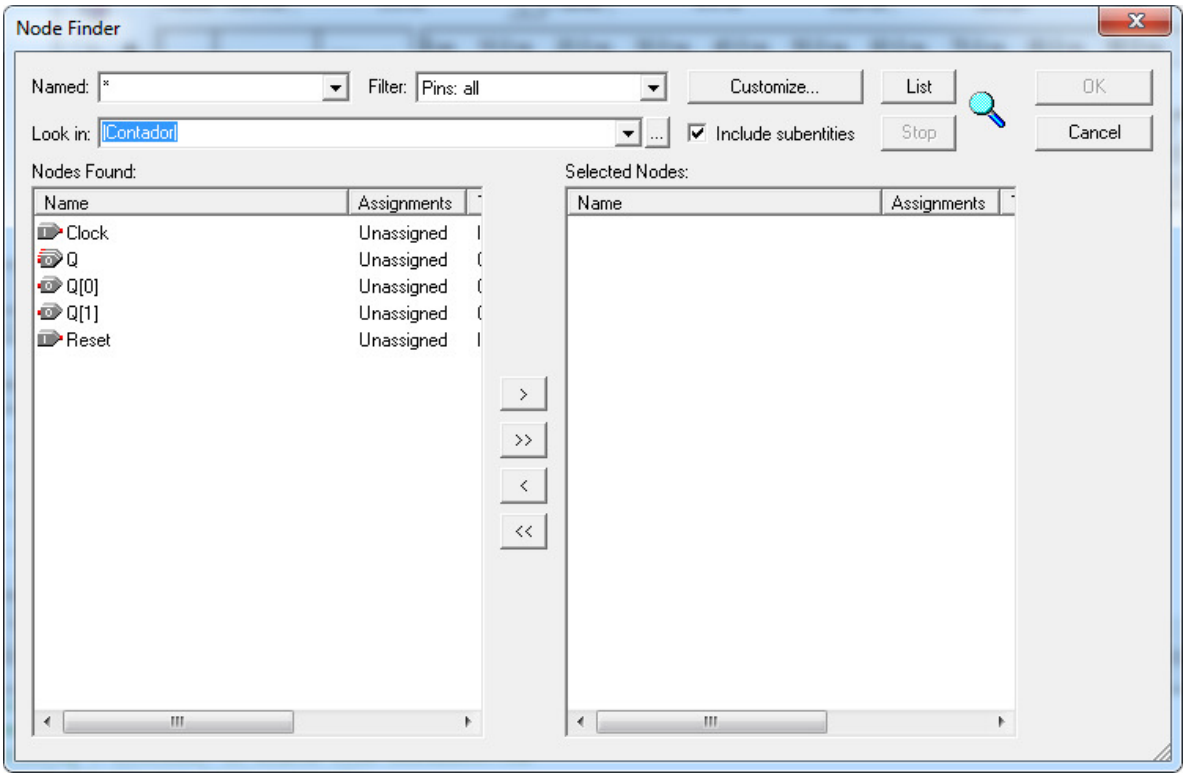


Figura 73- Sinais encontrados no código VHDL.

Vários sinais podem ser selecionados ao mesmo tempo e também podem ser transferidos para o arquivo de simulação ao mesmo tempo, para isso basta que seja selecionado o sinal e pressionado o botão “>”, fará com que o sinal seja copiado para este o espaço *“Selected Nodes*”. Na simulação deste projeto foram selecionados todos os sinais com a exceção do “Q”, que nada mais é do que a junção dos outros dois sinais de entrada (Q[0] e Q[1]).

Após esta seleção, a ação pode ser concluída clicando em “ok” das duas telas anteriores.

Neste momento o arquivo de *waveform* está com os sinais selecionados, porém sem os valores definidos, conforme é apresentado na Figura 74.

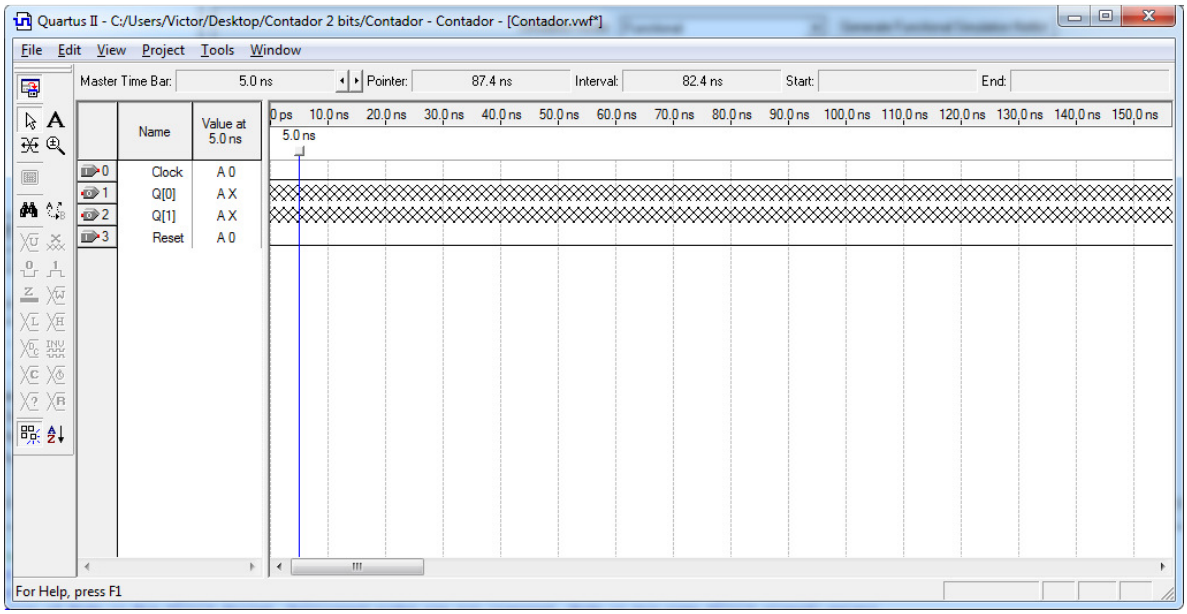


Figura 74- Arquivo Waveform com sinais já selecionados.

Para se atribuir os valores para cada sinal, basta clicar com o botão direito do *mouse* em cada sinal e com a aba *Value,* selecionar o tipo de valor necessitado, conforme mostra a Figura 75.

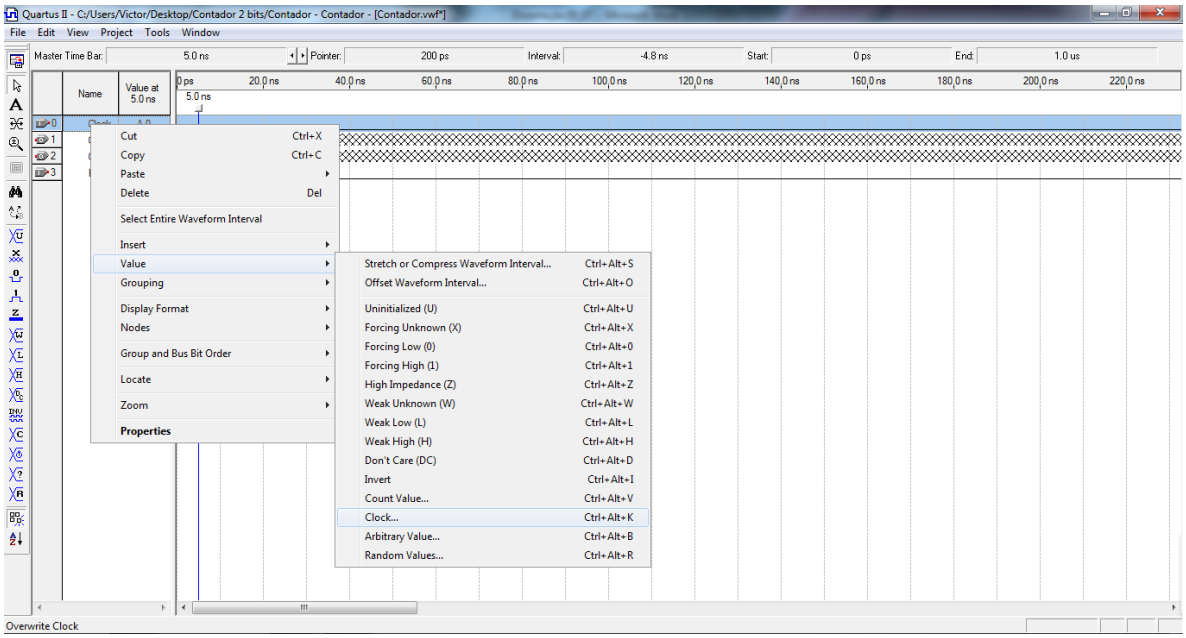


Figura 75- Inserindo valor a cada sinal.

Este procedimento deve ser repetido com cada sinal, sempre atribuindo os valores de acordo com aquilo que se espera simular.

A Figura 76 apresenta os sinais com seus respectivos valores.

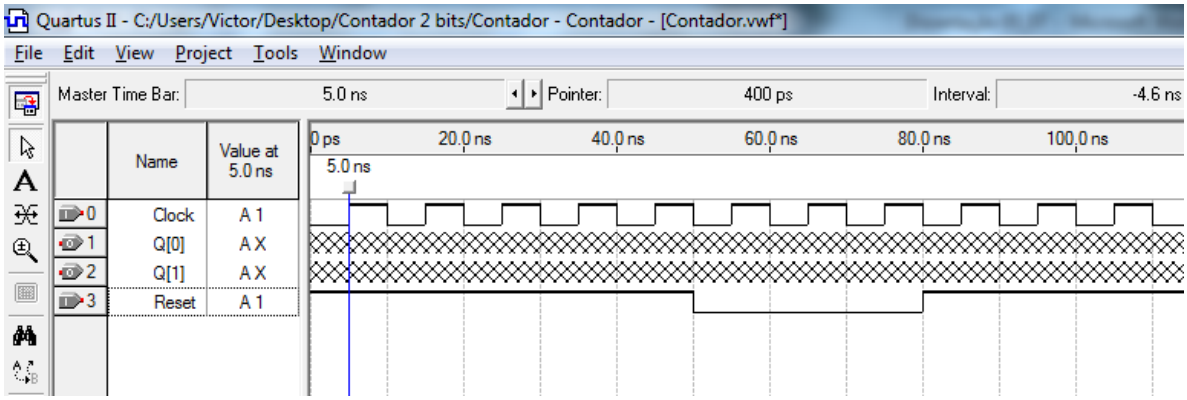


Figura 76- Arquivo de Waveform com sinais definidos.

Definido os sinais, a simulação pode ser iniciada. Para isso, primeiro deve-se salvar o arquivo de *waveform*, depois basta clicar no menu "Processing => Start Simulation", conforme mostrado na Figura 77.

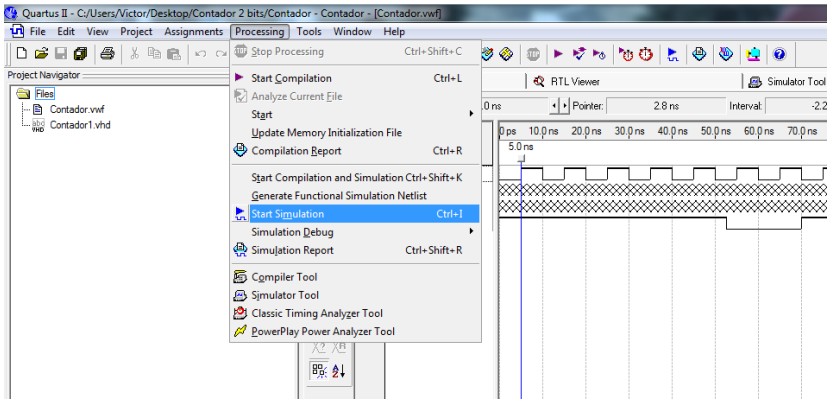


Figura 77- Iniciando a simulação.

Quando terminado o processo de simulação, o Quartus mostra a mensagem que a simulação foi realizada com sucesso (Figura 78).

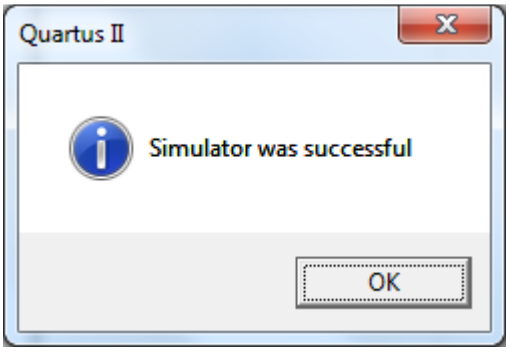


Figura 78- Mensagem de simulação ralizada com sucesso

Para analisar o arquivo de saída, basta retorna a tela “*Simulator Tool*” (Figura 69), que normalmente permanece aberto nas abas superiores do Quartus II, e clicar no botão “*Report*”, que abre uma janela semelhante a da *waveform*, porém com a resposta do sinal de saída, como mostrado na Figura 79.

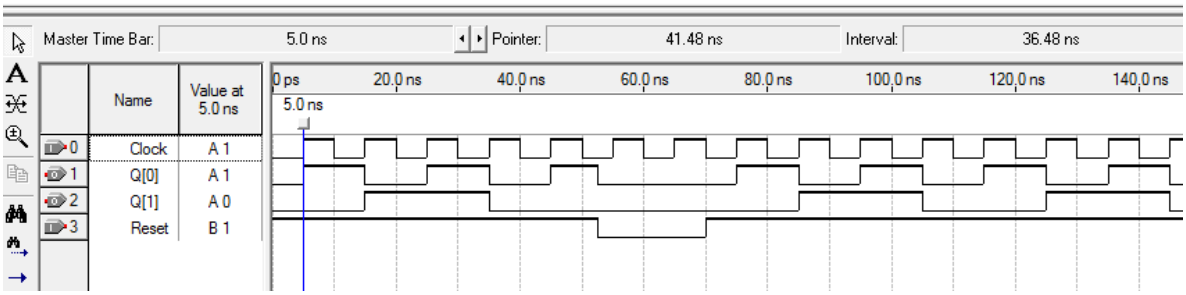


Figura 79- Janela da reposta da simulação realizada.

Agora basta analisar a resposta da simulação.