CENTRO UNIVERSITÁRIO DA FEI

FILIPE ANTOINE KHATCHADOURIAN

IMPLEMENTAÇÃO EM FPGA DE UM MICROCONTROLADOR 8051 A PARTIR DO CÓDIGO VHDL E GERAÇÃO AUTOMÁTICA DE LEIAUTE DOS BLOCOS ULA E RAM UTILIZANDO AS FERRAMENTAS DE CAD DA *MENTOR GRAPHICS*

São Bernardo do Campo

2014

FILIPE ANTOINE KHATCHADOURIAN

IMPLEMENTAÇÃO EM FPGA DE UM MICROCONTROLADOR 8051 A PARTIR DO CÓDIGO VHDL E GERAÇÃO AUTOMÁTICA DE LEIAUTE DOS BLOCOS ULA E RAM UTILIZANDO AS FERRAMENTAS DE CAD DA *MENTOR GRAPHICS*

Dissertação de Mestrado 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

Khatchadourian, Filipe Antoine.
Implementação em FPGA de um microcontrolador 8051 a partir do código VHDL e geração automática de leiaute dos blocos ULA e RAM utilizando as ferramentas de CAD da Mentor Graphics / Filipe Antoine Khatchadourian. São Bernardo do Campo, 2014.
139 f. : il.

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

1. 8051. 2. Leiautes de circuitos integrados. 3. Mentor Graphics. I. Gimenez, Salvador Pinillos, orient. II. Título.

CDU 681.3

CENTRO UNIVERSITÁRIO DA FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Filipe Antoine Khatchadourian **Matrícula:** 111303-4

Título do Trabalho: Implementação em FPGA de um microcontrolador 8051 a partir do código VHDL e geração automática de leiaute dos blocos ULA e RAM utilizando as ferramentas de CAD da mentor Graphics.

Área de Concentração: Dispositivos Eletrônicos Integrados

Orientador: Prof. Dr. Salvador Pinillos Gimenez

Data da realização da defesa: 08/05/2014

ORIGINAL ASSINADA

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

APROVADO ☒ REPROVADO ☐

São Bernardo do Campo, 08 de Maio de 2014.

MEMBROS DA BANCA EXAMINADORA

Prof. Dr. Salvador Pinillos Gimenez

Ass.: _____

Prof. Dr. Rodrigo Trevisoli Doria

Ass.: _____

Prof. Dr. Saulo Finco

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 alunos e professores do SENAI, instituição que tanto prezo, e aos amigos e familiares que acreditam em meu potencial e torcem pelo meu sucesso.

AGRADECIMENTOS

Agradeço aos amigos e familiares, a quem tive de abrir mão em alguns momentos para dedicar-me nesta etapa tão importante de minha vida.

Aos colegas de trabalho que sempre me apoiaram e tanto contribuíram para mais este passo em meu *curriculum*. Em especial aos meus coordenadores e diretores que viabilizaram muitas vezes a minha dedicação a este trabalho (Prof. Sidnei, Prof. Gomes, Prof. Coppe e Prof. Maurício).

Agradeço muito aos professores do Centro Universitário da FEI do curso de pós-graduação em Engenharia Elétrica, Prof. Dr. Marcelo Antonio Pavanello, Prof. Dra. Michelly de Souza, Prof. Dra. Paula Ghedini Der Agopian, Prof. Dr. Renato Camargo Giacomini, Prof. Dr. Rodrigo Trevisoli Doria, Prof. Dr. Carlos Eduardo Thomaz, e de modo particular meu orientador Prof. Dr. Salvador Pinillos Gimenez, pela ajuda, dedicação e empenho durante este trabalho.

Aos funcionários da instituição, principalmente da secretaria da pós-graduação.

Aos colegas de curso que tive o prazer de conhecer no decorrer das disciplinas e congressos, de modo particular ao amigo Victor Siqueira Martins Braga, por me ajudarem a conquistar os objetivos propostos.

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

À equipe de suporte da *Mentor Graphics*, em particular ao Kurt Liebermann, pela ajuda neste trabalho.

E finalmente à minha noiva Luana, por saber ser paciente e companheira nos momentos mais necessários para que eu conquistasse este sonho tão desejado.

RESUMO

O crescente mercado de *smartphones, tablets* e sensores para automação pessoal, predial e industrial tem impulsionado a utilização de eletrônica embarcada nas diferentes aplicações de Circuitos Integrados (CIs) atuais. O ponto central destas tecnologias são os circuitos integrados digitais. Microcontroladores e microprocessadores têm suas características elétricas e desempenho aprimorados ao mesmo tempo em que novas tecnologias de fabricação de Circuitos Integrados possibilitam um número maior de transistores por unidade área. A demanda por projetos de CIs para aplicações específicas tem crescido e, com ela, a necessidade por técnicas e ferramentas que desenvolvam leiautes com muita precisão e velocidade. Dentro deste contexto, este trabalho tem por objetivo a formação de recursos humanos na área de fabricação de CIs. Utilizando ferramentas em versão de demonstração descreve e implementa um fluxo de projeto para a criação de leiautes de CIs Digitais utilizando como estudo de caso um projeto inédito de leiaute semiautomático de um circuito de Modulação de Largura de Pulso (PWM) com uso da linguagem Verilog e do *software Microwind II*. Apresenta também a técnica de geração automática de leiaute utilizando ferramentas profissionais. A partir de um código VHDL, simulado no *software Quartus* e sintetizado no *software Leonardo Spectrum,* o fluxo apresentado será utilizado na implementação dos blocos de RAM e ULA de um microcontrolador 8051. Para realizar a validação do VHDL usado para a implementação do 8051, utilizou-se uma placa didática de *Field-Programmable Gate Array* (FPGA). A concepção do leiaute de forma automático é efetuado através das ferramentas de automação de projetos eletrônicos da *Mentor Graphics.* Utilizando uma biblioteca de células padrão de acordo com processo de fabricação, posiciona os transistores e realiza as interconexões de modo automático através do *software Pyxis Schematic.* O leiaute dos blocos foi efetuado através do *software Pyxis Layout.*

Palavras-chave: 8051. Leiautes de circuitos integrados. *Mentor Graphics*.

ABSTRACT

The growing market for smartphones, tablets and sensors for automation has impulsed the use of embedded electronics in everyday applications. The main point of these technologies are the integrated digital circuits. Microcontrollers and microprocessors have their electrical characteristics and performance enhanced while new technologies for manufacturing integrated circuits (ICs) enable a larger number of transistors per unit area. The demand for ICs projects for specific applications has grown, and with it, the need for tools and techniques to develop layouts with great precision and speed. This paper describes and implements a design flow for creating layouts of Digital ICs. It demonstrates an unpublished semi-automatic layout design of a PWM circuit using Verilog language and Microwind II software. It also presents the technique of automatic generation of layout from VHDL code simulated in Quartus software and synthesized in Leonardo Spectrum software. The flow showed will be used in the implementation of blocks of RAM and ALU of an 8051 microcontroller. To perform the validation of the VHDL used to implement the 8051 it was utilized a didactic board Field Programmable Gate Array (FPGA). The layout is performed automatically through the electronic design automation tools from Mentor Graphics. Using a library of standard cells according to the manufacturing process, the transistors and interconnects are positioned automatically using the Pyxis Schematic software. The layout of the blocks was done through the Pyxis Layout software.
Keywords: 8051. Digital integrated circuit layouts. *Mentor Graphics*.

LISTA DE FIGURAS

LISTA DE SÍMBOLOS

a, b e c	Entradas do circuito eletrônico digital de exemplo
ALE	*Adress Latch Enable, s*inal de sincronismo do microcontrolador 8051 para a demultiplexação do barramento de endereço
Clk	Sinal de *clock*
CLK1	Primeira entrada digital do circuito de portas lógicas
CLK2	Segunda entrada digital do circuito de portas lógicas
D	Sinal de entrada do *flip-flop* do tipo D
E0, Ent[0]	Primeira entrada digital do circuito PWM
E1, Ent[1]	Segunda entrada digital do circuito PWM
E2, Ent[2]	Terceira entrada digital do circuito PWM
$\overline{EA}$	Habilita a busca das instruções na memória externa
E/S	Entradas e saídas do microcontrolador
f	Frequência
GND	Sinal 0 volt do circuito (terra)
I8051_ALU	Bloco da Unidade Lógica Aritmética do microcontrolador 8051
I8051_CTU	Bloco do núcleo do microcontrolador 8051
I8051_DBG	Bloco da depuração do microcontrolador 8051
I8051_DEC	Bloco da decodificador de instruções do microcontrolador 8051
I8051_RAM	Bloco da memória RAM do microcontrolador 8051
I8051_ROM	Bloco da memória ROM do microcontrolador 8051
P0	Porta 0 de entrada e saída do microcontrolador 8051
p0_in	Porta 0 de entrada do microcontrolador 8051
p0_out	Porta 0 de saída do microcontrolador 8051
P1	Porta 1 de entrada e saída do microcontrolador 8051
p1_in	Porta 1 de entrada do microcontrolador 8051
p1_out	Porta 1 de saída do microcontrolador 8051
P2	Porta 2 de entrada e saída do microcontrolador 8051
p2_in	Porta 2 de entrada do microcontrolador 8051
p2_out	Porta 2 de saída do microcontrolador 8051
P3	Porta 3 de entrada e saída do microcontrolador 8051
p3_in	Porta 3 de entrada do microcontrolador 8051
p3_out	Porta 3 de saída do microcontrolador 8051

$\overline{PSEN}$	Sinal de leitura da memória externa de programa do 8051
Q	Sinal de saída do *flip-flop* do tipo D
Q_INV	Sinal de saída invertida do *flip-flop* do tipo D
$\overline{RD}$	Sinal de leitura da memória de dados externa
RST	Sinal de *reset* do microcontrolador 8051
Sai	Sinal de saída digital do circuito PWM
T	Período
Vcc	Tensão de alimentação positiva dos circuitos eletrônicos
Vdd	Tensão de alimentação positiva dos circuitos eletrônicos
$\overline{WD}$	Sinal de escrita da memória de dados externa do 8051
$\overline{WR}$	Sinal de escrita da memória de dados externa do 8051
x	Saída do circuito eletrônico digital de exemplo
xrm_addr	Barramento de endereços com a memória RAM externa do 8051
xrm_in_data	Barramento de dados de entrada com a memória RAM externa do 8051
xrm_out_data	Barramento de dados de saída com a memória RAM externa do 8051
xrm_rd	Sinal de leitura da memória RAM externa do 8051
xrm_wr	Sinal de escrita da memória RAM externa do 8051
y	Saída do circuito eletrônico digital de exemplo

LISTA DE ABREVIATURAS

ASIC	*Application Specific Integrated Circuit* (CIS de aplicação específica)
CAD	*Computer Aided Design* (Projeto assistido por computador)
CI	Circuito Integrado
CISC	*Complex Instruction Set Computer* (Computador com Conjunto Complexo de Instruções)
CMOS	*Complementary Metal-Oxide-Semiconductor* (Semicondutor-Metal-Óxido Complementar)
CPLD	*Complex Programmable Logic Device* (Dispositivo Lógico Programável Complexo)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
DARPA	*Defense Advanced Research Projects Agency* (Departamento de Defesa dos Estados Unidos da América)
DPTR	*Data Pointer* (Ponteiro de Dados)
DRC	*Design Rules Checker* (Verificador de regas de projeto)
DSP	*Digital Signal Processor* (Processador digital de sinal)
EDA	*Electronic Design Automation* (Automação de Projetos Eletrônicos)
FET	*Fiel Effect Transistor* (Transistor de efeito de campo)
FPGA	*Field-Programmable Gate Array* (Matriz de Portas Programável por Campo)
GDSII	*Graphic Data System* (Sistema de banco de dados gráfico)
GUI	*Graphical User Interface* (Interface gráfica do usuário)
HEP	*High Education Program* (Programa de ensino superior)
HS	*High Speed* (Alta velocidade)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Engenheiros Eletricistas e Eletrônicos)
IP	*Intellectual Property* (Propriedade intelectual)
LPLV	*Low Power Low Voltage* (Baixa tensão e baixa potência)
LSI	*Large System Integration* (Sistema de alta integração)
LVS	*Layout Versus Schematic* (Leiaute versus esquemático)
MOSFET	*Metal-Oxide-Semiconductor Field Effect Transistor* (Transistor de efeito de campo MOS)
NRE	*Non-recurring Engineering* (Engenharia não recorrente)
PC	*Program Counter* (Contador de Programa)

PWM	*Pulse-Width Modulation* (Modulação em largura de pulso)
RAM	*Random Access Memory* (Memória de acesso aleatório)
RF	*Radio Frequency* (Rádio Frequência)
ROM	*Read Only Memory* (Memória Apenas de Leitura)
RTL	*Register Transfer Level* (Nível de transferência para os registradores)
SoC	*System on-a-chip* (Sistema em um *chip*)
SDL	*Schematic Driven Layout* (Leiaute Guiado pelo Esquemático)
SFR	*Special Function Register* (Registrador de Funções Especiais)
UART	*Universal Asynchronous Receiver/Transmitter* (Receptor/transmissor universal assíncrono)
ULA	Unidade Lógica Aritmética
VHDL	*Very High Speed Integrated Circuit Hardware Description Language* (Linguagem de descrição de *hardware* de Circuito Integrado de Muito Alta Velocidade)
VHSIC	*Very High Speed Integrated Circuit* (Circuito Integrado de Muita Alta Velocidade)
VLSI	*Very Large System Integration* (Sistema de integração muito alta)

SUMÁRIO

INTRODUÇÃO

Estamos vivendo a era digital. De automóveis a *mainframes*, dispositivos móveis, de telecomunicações, de multimídia e equipamentos eletrônicos sem fio. Todos estes equipamentos, comuns no nosso dia a dia, são constituídos por micro/nanoprocessadores e circuitos integrados (CIs). A popularização dessa tecnologia impulsiona o desenvolvimento da microeletrônica de maneira muito rápida [1] [2] [3] [4].

Década a década, a evolução da eletrônica vem marcando a história, desde os Transistores de efeito de campo Metal-Óxido-Semicondutor (*Metal-Oxide-Semiconductor Field Effect Transistor,* MOSFETs), nos anos 60 [2]. Nos anos 70, surgiram, e viu-se a incrível evolução das memórias não-voláteis e voláteis [2]. Mais tarde, surgiu na década de 80, a integração de larga escala (*Large System Integration*, LSI) dos CIs [2]. Posteriormente, na década de 90, nos deparamos com os fantásticos CIs de baixa tensão e baixa potência (*Low Power Low Voltage*, LPLV) e de alta velocidade (*High Speed*, HS) [2]. Por fim, do início do século XXI até os dias de hoje, utilizamos CIs de alto desempenho elétrico com baixo consumo de energia elétrica, micro/nano-processadores com múltiplos núcleos, a superação da barreira dos 1 GHz de velocidade de processamento e até mesmo a técnica da flutuação da tensão de alimentação durante o processo de escrita em memória volátil [2].

Em 1975, Gordon E. Moore, co-fundador da Intel, vislumbrou a lei de que, a cada 18 meses, o número de transistores dobraria em um circuito integrado (Figura 1-1) [5].

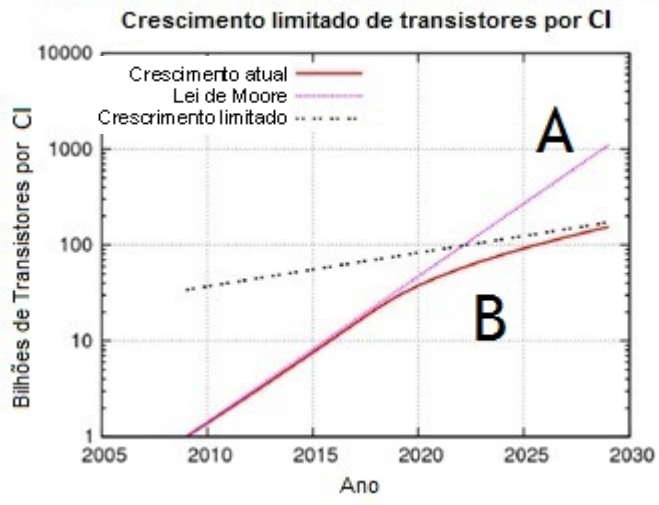


Figura 0-1: Quantidade de transistores por CI em bilhões em função dos anos.
Fonte: [5].

Na Figura 1-1, a curva A, ilustra a previsão da quantidade de transistores em um CI, segundo a Lei de Moore [5]. Na curva B, observa-se que na realidade dos dias atuais, é respeitada a projeção de Moore [5]. Contudo, a dificuldade em colocar mais transistores numa área de núcleo cada vez menor, pode gerar limitações no crescimento do número de dispositivos por CI, como se pode observar por meio da curva tracejada [5].

Assim, novas tecnologias de transistores a cada dia são apresentadas, mantendo-se viva a escalabilidade dos CIs proposta por Moore, como é o caso dos MOSFETs em escalas nanométricas tridimensionais (*nanowires*) [6].

A popularização dos computadores possibilitou esta evolução tecnológica para a criação de microprocessadores que atendessem às necessidades do constante aumento de processamento dos equipamentos portáteis eletrônicos [1]. Entretanto, hoje já não são mais os processadores para computadores que encabeçam os projetos de CIs [1]. A cada dia, surgem projetos dedicados, como os CIs Digitais de Aplicação Específica (*Application-Specific Integrated Circuit*, ASICS), impulsionados pelo conceito de *System on-a-chip* (SoC), que são utilizados para implementar produtos modulares, integrando diferentes blocos eletrônicos especiais de propriedade intelectual (*Intellectual Propreties*, IPs) de funções específicas [1].

Diferente do conceito de um produto para uso geral, atualmente existe uma crescente demanda por CIs de necessidades específicas [1]. Circuitos integrados digitais são projetados especificamente para telefones celulares, *i-pads*, *tablets*, microcomputadores pessoais, etc., utilizados como Processadores Digitais de Sinal (*Digital Signal Processors*, DSPs) e de outras funções [1]. Com este crescimento de demanda por ASICS, e com a individualização na funcionalidade do CI, os projetos de CIs voltados para aplicações em grande escala de integração (*Very Large System Integration*, VLSI) são realizados em dias, ao invés de semanas e meses, como eram feitos há 20 anos atrás [1].

Além disso, novas aplicações de CIs surgem a cada dia trazendo novos desafios, como por exemplo, o uso de CIs em ultra baixas e altas temperaturas, ambientes radiativos para as aplicações médicas e aeroespaciais, contendo blocos analógicos, digitais e de rádio frequência (RF), isto é, os chamados circuitos integrados mistos (*Mixed Integrated Circuits*), além de blocos de processador e memórias [2] [4].

Como a tecnologia de fabricação de CIs nanométricos evolui a cada dia, com o intuito de suportar cada vez mais uma maior densidade de transistores por unidade de área, o projetista de CIs digitais, atualmente, tem à disposição diversos fabricantes e ferramentas de *software*, que permitem desde a concepção dos CIs até geração automática de leiautes.

Uma forma de projetar CIs digitais consiste no uso de linguagens de descrição de *hardware* (Linguagem de descrição de *hardware* de Circuito Integrado de Muito Alta Velocidade - *Very High Speed Integrated Circuit Hardware Description Language* / VHDL, Verilog, etc), que são capazes de descrever um circuito eletrônico, tais como multiplexadores, demultiplexadores, *flip-flops*, *buffers,* portas lógicas, etc [3] [7].

Uma ferramenta de sintetização de CIs digitais tem por objetivo otimizar a função lógica a ser desenvolvida e, usualmente, produz melhores resultados que a realizada de forma manual, pois utiliza uma poderosa heurística para alcançar uma maior integração e menor tempo de processamento [7].

Softwares de Projetos Assistidos por Computador (*Computer Aided Design,* CAD) para desenvolvimento de CIs atuais possuem, como característica fundamental, a função da geração automática de leiaute, pois devido ao grande número de transistores por CI, praticamente seria impossível realizar tal tarefa de forma manual [3]. *Cadence Design System Inc.* [8] e *Mentor Graphics* [9], são referências no mercado por ofertarem esse tipo de CAD, ou seja, a Automação de Projetos Eletrônicos de CIs (*Electronic Design Automation*, EDA) [3].

Dentro desse contexto, este trabalho tem como objetivos: 1 – A implementação de um fluxo de projeto para a criação de leiautes semi-automatizados para a formação de recursos humanos na fabricação de circuitos integrados digitais não complexos, utilizando ferramentas em versão demonstração que estão acessíveis para qualquer instituição de educação para projetos de fins acadêmicos; 2 – O desenvolvimento de projetos de CIs digitais complexos utilizando-se como veículo de aplicação a geração automática de parte do leiaute de um circuito integrado digital, neste caso, o consagrado microcontrolador da família 8051, com ferramentas computacionais profissionais.

Os leiautes desenvolvidos no Centro Universitário da FEI são criados, hoje, todos manualmente. Assim, este trabalho tem também por objetivo gerar leiautes de circuitos digitais de forma automática utilizando as ferramentas da *Mentor Graphics* para projetos de circuitos integrados digitais, a serem desenvolvidos pela FEI.

A estrutura deste trabalho traz, na seção dois, os conceitos fundamentais para o seu entendimento, a linguagem de descrição VHDL, utilizada nos projetos de circuitos integrados digitais, o microcontrolador 8051, sua arquitetura e características construtivas e elétricas, e o projeto de CIs digitais de aplicação específica, onde se detalham os passos necessários para a sua realização e também são descritas as suas principais características.

A seção três traz a geração semiautomática do leiaute (utilização do Verilog na descrição do circuito) de um circuito de Modulação em Largura de Pulso (*Pulse Width Modulation,* PWM), que foi implementado com as ferramentas *Quartus* [10], *Circuit Wizard* [11] e *Microwind II* [12] com o objetivo de dar a possibilidade aos projetistas de CI que não possuem ferramentas profissionais na área de CIs, semelhantes aos da *Mentor Graphics*.

Na seção quatro, é implementado um microcontrolador 8051 em um Arranjo de Portas Programável em Campo (*Field-Programmable Gate Array,* FPGA), utilizando o *Quartus II* da Altera. É realizada a criação dos blocos Memória de acesso aleatório (*Random Access Memory*, RAM) e Unidade Lógica Aritmética - ULA em VHDL do 8051, e os testes de funcionamento para a validação dos códigos VHDL. Relata passo a passo, como gerar um leiaute automático, utilizando as ferramentas *Mentor Graphics.* A geração automática do leiaute utilizando o *Leonardo Spectrum* [13] para a sintetização lógica do VHDL, a criação das células padrão e o esquemático através do *Pyxis Schematic* [14], e a concepção do leiaute através do *Pyxis Layout* [15], aplicado na concepção dos blocos ULA e RAM do 8051.

A quinta e última seção apresenta as principais conclusões deste trabalho e as sugestões de trabalhos futuros.

CONCEITOS FUNDAMENTAIS

Esta seção tem por objetivo definir a técnica de controle de cargas PWM, citar os tipos de simulações de circuitos digitais, descrever alguns aspectos da linguagem de programação VHDL e as suas características básicas, além de alguns exemplos práticos. Descreve-se também os conceitos básicos e a arquitetura de um microcontrolador 8051. Também será descrito o fluxo de projetos de leiaute de CIs digitais ASIC.

1.1 Modulação por Largura de Pulso (*Pulse Width Modulation,* PWM)

A modulação por largura de pulso, geralmente chamada pela sigla PWM, é uma técnica utilizada no controle de cargas de corrente contínua. Consiste em ativar ou desativar o funcionamento da carga utilizando-se de dispositivos eletrônicos de chaveamento, como os MOSFETs de potência [16].

Durante o período em que a carga é controlada o chaveamento pode receber um lógico ou zero. Se um lógico for recebido durante todo o período a carga estará sempre acionada, produzindo trabalho sempre. Este tempo que a carga é acionada é denominado ciclo de trabalho ou *Duty Cicle.* Se o chaveamento ocorre durante todo o período, o *Duty Cicle* é de 100%. A medida que o chaveamento desativa a carga durante o período, o valor do ciclo de trabalho é reduzido proporcionalmente. Por exemplo: se uma carga é acionada metade do tempo do período, o *Duty Cicle* será de 50% [16].

Este controle do ciclo de trabalho faz com que haja uma variação controlada do funcionamento da carga. Assim pode-se controlar a velocidade de motores, a luminosidade de lâmpadas, etc [16].

Um circuito PWM digital é aquele que controla o *Duty Cicle* por meio de um controle por bits. Por exemplo: utilizando três bits para o controle de uma carga, divide-se um período em oito partes ($2^3 = 8$ possibilidades). A combinação binária determinará qual será o valor do ciclo de trabalho, ou por quanto tempo dentro do período a carga permanecerá em estado lógico um [16].

1.2 Tipos de simulação de circuitos digitais

O objetivo da simulação de um projeto de circuito integrado digital é verificar se o circuito corresponde às características de sua especificação. Existem dois tipos: simulação funcional e simulação temporizada [17].

A simulação funcional verifica apenas a condição lógica do circuito digital. Basicamente verifica as equações *booleanas* e as máquinas de estado lógico. Não considera o atraso que os dispositivos podem apresentar, ou considera os atrasos igual a zero. É uma simulação simples e rápida que visa identificar erros de projeto muito rapidamente [17].

A simulação temporizada além da condição lógica do circuito, verifica o funcionamento com a propagação de sinais dos dispositivos, que operam com atrasos reais que podem gerar resultados diferentes do esperado. É uma simulação mais próxima da realidade e sendo assim, mais demorada [17].

1.3 Definição e histórico da linguagem VHDL (*Very High Speed Integrated Circuit Hardware Description Language)*

Define-se VHDL como uma Linguagem de Descrição de *Hardware* para Circuitos Integrados de Alta Velocidade [18]. É utilizada para a descrição de sistemas digitais eletrônicos. É padronizada pelo Instituto de Engenheiros Eletricistas e Eletrônicos (*Institute of Electrical and Electronics Engineers,* IEEE) e foi criada na década de 80 pelo Departamento de Defesa dos Estados Unidos da América (*Defense Advanced Research Projects Agency,* DARPA) [18].

Primeiramente, esta linguagem permite a descrição da estrutura do circuito eletrônico, que pode ser composta por sub-circuitos e por suas interligações (como eles são interligados entre si) [18]. Depois, uma especificação da funcionalidade de cada um dos sub-circuitos, que pode ser realizada utilizando-se um formato específico da linguagem de descrição de *hardware* [18]. Por fim, utilizando a linguagem VHDL para uma simulação dos circuitos eletrônicos antes de sua produção possibilita rápidas comparações de desempenho elétrico e de funcionalidade entre os diferentes sub-circuitos eletrônicos. Além disso, é possível realizar as correções e alterações nesses circuitos eletrônicos na descrição VHDL, sem qualquer custo [18].

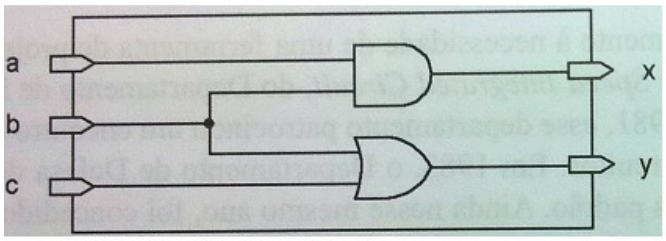
Após a criação da primeira linguagem de descrição de *hardware* em 1987, uma nova versão foi aprovada em 1993 pelo IEEE [19]. Não há grandes mudanças em relação à síntese

de circuitos eletrônicos, mas existiram grandes mudanças com relação ao tratamento dos diferentes arquivos que compõem o projeto como um todo [19]. Muitas vezes os códigos podem ser compilados sem a preocupação com a versão do VHDL utilizado [19].

Para a representação dos circuitos eletrônicos, utilizam-se os chamados esquemáticos, com simbologias padronizadas [20]. Esta técnica gráfica facilita a visualização da funcionalidade e dos componentes utilizados [20]. Contudo, se necessitarmos representar um circuito integrado complexo, com alta densidade de componentes, o método gráfico pode dificultar a visualização do todo, principalmente das ligações dos diversos blocos construtivos e dos componentes eletrônicos [20].

O intercâmbio de informações referentes ao comportamento do circuito eletrônico, entre ferramentas de *software*, sistemas, fabricantes e a documentação do projeto, são fatores que também justificaram a criação de uma linguagem textual que descrevesse o esquemático do *hardware* de um circuito integrado eletrônico a ser implementado [20].

Com o passar do tempo, foi criada uma linguagem de descrição de *hardware,* que permitisse um método mais rápido e simplificado de criar esquemáticos de circuitos eletrônicos que possuem um grande número de componentes [20]. Além disso, elas permitem a realização de alterações para customizar e reprogramar esses circuitos eletrônicos mais facilmente [20]. Um exemplo do uso desta linguagem está apresentado na Figura 2-1.



(a)

```
1  ENTITY portas IS
2    PORT (a, b, c : IN  BIT;
3          x, y    : OUT BIT);
4  END portas;
5
6  ARCHITECTURE teste OF portas IS
7  BEGIN
8    x <= a AND b;
9    y <= c OR  b;
10 END teste;
```

(b)

Figura 0-1: Exemplo de um esquema elétrico de um circuito eletrônico digital (a) e do respectivo código VHDL (b).
Fonte: [19].

Na Figura 2-1 (a), *a, b* e *c* são as entradas de um exemplo de um circuito digital, que é composto pelas portas lógicas E e OU, e *x* e *y* são as saídas do circuito. A Figura 2-1 (b) apresenta o código VHDL do circuito. Na linha 1, a entidade "portas" é declarada. Na linha 2 são definidas as entradas binárias *a, b* e *c*, e na linha 3, são definidas as saídas binárias *x* e *y*. Na linha 6, a arquitetura "teste" é definida. Na linha 8 uma porta lógica do tipo E foi definida,

tendo como entradas *a* e *b*, e a saída *x*. Na linha 9, uma porta lógica do tipo OU foi definida, tendo com entradas *c* e *b*, e a saída *y*. A linha 10 finaliza a arquitetura.

1.3.1 Estrutura da descrição VHDL

Existem dois modos de estruturar um código VHDL: estrutural e comportamental. O modo estrutural consiste da descrição da interligação de descrições menores, enquanto que o modo comportamental consiste em descrever o comportamento esperado do circuito eletrônico [21].

A descrição de um circuito eletrônico em VHDL começa com a declaração da entidade. Depois, deve-se descrever a arquitetura da entidade, que define o circuito eletrônico interno com suas conexões [20]. Para tal, é realizada uma declaração da lógica utilizada e as conexões de suas entradas e saídas [20].

A Figura 2-2 apresenta um exemplo de arquitetura no modo estrutural de um contador de 2 *bits*.

```
-- Descrição Estrutural do Contador de 2 bits --
1  ARCHITECTURE estrutural OF contador IS
2              SIGNAL ff0, ff1, inv_ff0:STD_LOGIC;
3  BEGIN
4     bit_0: entity work. Tflip_flop
5              port map (ck=>clock, q=>ff0);
6     inv:    entity work. Inversor
7              port map (a=>ff0, y=>inv_ff0);
8     bit_1: entity work. Tflip_flop
9              port map (ck=>inv_ff0, q=>ff1);
10     q0 <=ff0;
11     q1 <=ff1;
12    END estrutural
```

Figura 0-2: Exemplo do código VHDL da arquitetura estrutural de um contador.
Fonte: [21].

Na Figura 2-2, a descrição estrutural do contador inicia declarando na linha 1, a entidade (interface que contém parâmetros de entradas e saídas e conexões) do contador. Na

linha 2, a declaração dos sinais que interligam as várias entidades que serão utilizadas. No caso do contador, são utilizadas duas entidades denominadas de *flip-flop* (linhas 4 e 8) e uma entidade denominada de inversor (linha 6). Ao final, são declaradas as saídas do contador, recebendo o sinal das variáveis de interligação (linhas 10 e 11).

A Figura 2-3 traz um exemplo do modo comportamental do mesmo contador.

```
-- Descrição Comportamental do Contador de 2 bits --
1  ARCHITECTURE comportamental OF contador IS
2              SIGNAL q : std_logic_vector (1 dowto 0);
3  BEGIN
4     count_up : process (clock)
5              variable count_value : integer :=0;
6  BEGIN
7  IF clock = '1' THEN
8     count_value := (count_value + 1) mod4;
9     q <= conv_std_logic_vector (count_value,2) after prop_delay;
10 END IF;
11 END process count_up;
12 q0 <= q(0);
13 q1 <= q(1);
14 END comportamental;
```

Figura 0-3: Exemplo do código VHDL comportamental de um contador.
Fonte: [21].

O modo comportamental descreve a funcionalidade de um contador através de linhas de código. Na linha 1, a declaração da entidade. São declaradas nas linhas 4 e 5, as variáveis *count_up*, *count_value*. É utilizada uma função condicional "se", declarada na linha 7. Dentro da função "se", realiza-se o incremento das variáveis (na linha 8) e, fora do processo, a alteração do estado das saídas, nas linhas 12 e 13.

O nome da entidade é muito importante, pois atrelado a ele está o nome do projeto. Nesta descrição também estão definidos as interfaces e os nomes e tipos de portas de entradas e saídas [20].

1.3.2 Sintaxe da linguagem de descrição VHDL

A ordem dos comandos é irrelevante em relação ao comportamento do circuito eletrônico. A execução não é sequencial, mas concorrente [19]. Contudo, podem-se delimitar regiões onde o código é executado sequencialmente [19].

A linguagem permite a utilização de funções e subprogramas [19]. Não há distinção entre maiúsculas e minúsculas [19]. Pode-se, ainda, criar códigos VHDL para a realização de teste de uma estrutura, que simula valores de entradas e saídas [20]. A sintaxe das linhas de código segue o padrão das linguagens convencionais de descrição de *hardware*, basicamente combinando os operadores primários [20].

Comentários na linha de programação devem ser precedidos de hífen (--) [18].

1.3.3 Pacotes e bibliotecas da linguagem VHDL

Ferramentas muito utilizadas na linguagem VHDL são os pacotes e bibliotecas [19]. Pacotes são, na realidade, pequenos códigos, ou partes de códigos e linhas de comandos. A sua utilização é efetuada para evitar repetições de linhas de código, ou ainda quando um comando não existe na biblioteca padrão e precisa ser utilizado num circuito. [19]. Já as bibliotecas armazenam informações compiladas, como funções básicas. Ambos, ou seja, os pacotes e as bibliotecas, devem ser declarados durante a elaboração da descrição de um circuito eletrônico [19].

Existem, ainda, dois tipos de bibliotecas especiais. O primeiro tipo, invocado através da declaração *work,* define as bibliotecas a serem utilizadas no projeto que será implementado em VHDL pelo projetista [19]. Além das bibliotecas criadas pelos próprios programadores, normalmente, pode-se utilizar as bibliotecas do padrão IEEE 1076.3 ou IEEE 1164, que também selecionam a versão da linguagem VHDL utilizada [19]. O segundo tipo de bibliotecas, chamadas de especiais, é denominado *std* e contêm funções e operações básicas da linguagem VHDL [19].

1.4 Microcontrolador 8051

1.4.1 Introdução

O 8051 é um dos mais populares dos microcontroladores, podendo operar como microcontrolador ou microprocessador [16]. É membro da família MCS-51 da Intel, que foi criada na década de 80. Ganhou muita popularidade e até hoje é muito utilizado em sistemas embarcados (sistemas com capacidade de processamento, dedicados a tarefas específicas [22]) [16]. Em sua forma básica, o núcleo é acompanhado de periféricos, como temporizadores, contadores, memórias e portas de entrada e saída [16]. Este microcontrolador é comercializado por diversos fabricantes, tais como Texas Instruments, Phillips, Analog Devices, Siemens, entre outros. Chamam atenção suas características de projeto de alta estabilidade elétrica, robustez e confiabilidade [16]. A Figura 2-4 ilustra os diversos blocos que compõem a arquitetura interna do microcontrolador 8051.

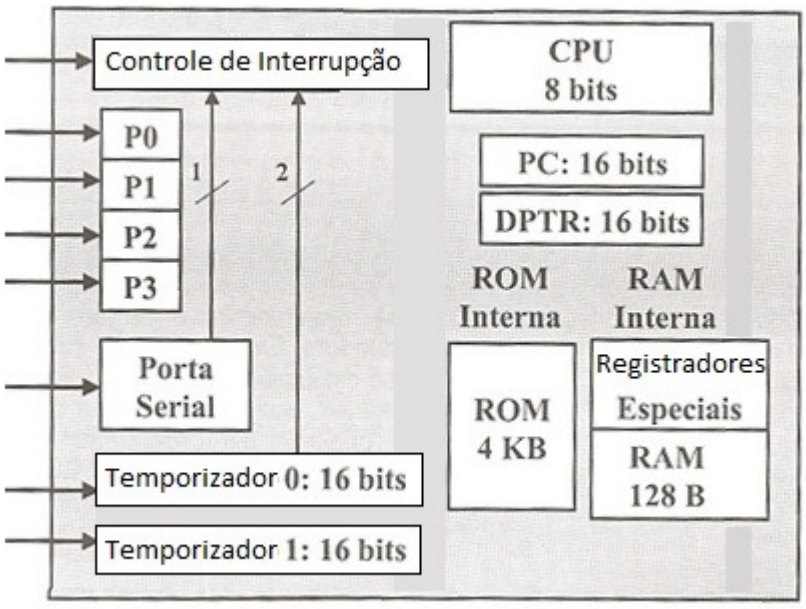


Figura 0-4: Arquitetura interna do 8051.
Fonte: [23].

Na Figura 2-4, são mostradas as partes que compõem a arquitetura básica do microcontrolador 8051: os dois temporizadores/contadores, sendo que um deles é utilizado também no controle de interrupção, a Unidade Central de Processamento (*Central Processing Unit,* CPU), os blocos de memórias, as portas de entrada e saída P0, P1, P2 e P3, e os diversos

registradores de função especiais, tais como o Contador de Programa (*Program Counter*, PC) e o Ponteiro de Dados (*Data Pointer*, DPTR), entre outros.

São algumas características gerais do microcontrolador 8051 [16]:

a) CPU de 8 *bits* com alta capacidade de processamento *booleano*;
b) Memória interna de programa de 4 k*bytes*;
c) Memória interna RAM de 128 *bytes*;
d) Quatro registradores de 8 *bits* correspondente às portas de entrada ou saída (P0, P1, P2 e P3);
e) Um canal de comunicação serial assíncrono *full-duplex*;
f) Dois temporizadores ou contadores programáveis de 16 *bits* [16];

1.4.2 Arquitetura interna

O microcontrolador 8051 é de arquitetura CISC (*Complex Instruction Set Computer*), ou seja, sua arquitetura é de um computador com um conjunto complexo de instruções, que possui um conjunto de instruções de 111 tipos, num total de 255 instruções [24]. As instruções são divididas por grupos, sendo: 24 aritméticas, 25 lógicas, 28 de transferência de dados, 17 *booleanas* e 17 de saltos [24].

Não é baseado na máquina de Von Neumann (armazena dados junto com programa), pois é capaz de endereçar separadamente a memória de dados da memória de programa, conforme está ilustrado na Figura 2-5, seguindo a arquitetura *Harvard,* mais nova e que permite maior velocidade, pois busca umas instrução enquanto executa outra [23].

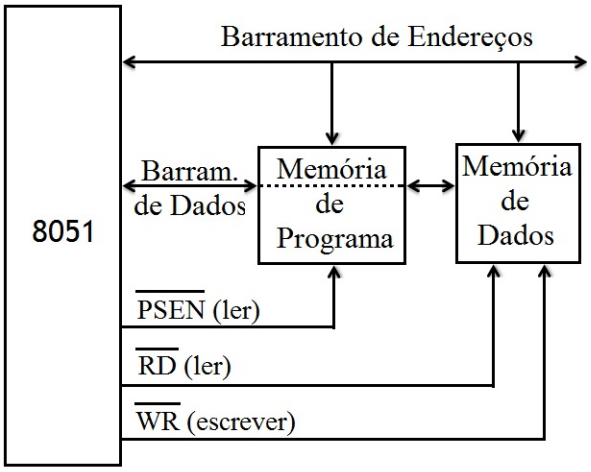


Figura 0-5: Típica arquitetura do 8051.
Fonte [23].

Na Figura 2-5, $\overline{PSEN}$ é o sinal de leitura da memória de programa externa, $\overline{RD}$ é o sinal de leitura para a memória de dados externa e $\overline{WR}$ é o sinal para escrita da memória de dados externa [23].

Cada ciclo de máquina do 8051 é composto por seis estágios [16]. Cada estágio tem duração de dois períodos de *clock* [16]. Existem instruções que precisam apenas de um ciclo de máquina para serem executadas, porém outras, como instruções aritméticas, podem precisar de dois ou mais ciclos de máquina [16].

Existem cinco fontes de interrupção no 8051: uma para a comunicação serial de dados do receptor/transmissor universal assíncrono (*Universal Asynchronous Receiver/Transmitter,* UART), duas para as interrupções externas, e duas para os dois temporizadores/contadores. Os níveis de prioridade obedecem à sequência que está definida na Tabela 2-1.

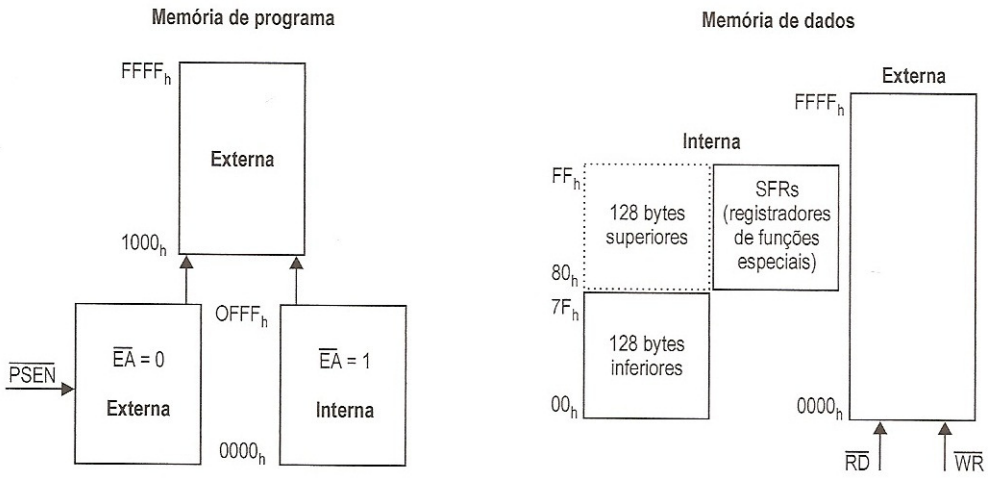
Tabela 2-1: Ordem de prioridade das interrupções do microcontrolador 8051.

Interrupções	
Prioridade	**Tipo**
	Reset
1º	Externo 0
2º	Temporizador 0
3º	Externo 1
4º	Temporizador 1
5º	Serial

1.4.2.1 Estrutura das memórias do 8051

As memórias do 8051 são organizadas por endereçamentos separados de Memória Apenas de Leitura (*Read Only Memory,* ROM) e Memória de Acesso Aleatório (*Random Access Memory,* RAM) [16]. Estas memórias internas são de 4 *kbytes* para a de programa e 128 *bytes* para a de dados. Tem ainda a capacidade de endereçamento externo de memória de até 64 *kbytes*, tanto para memória de programa quanto para memória de dados [16].

Na memória de programa, que pode ser interna ou externa, podem-se armazenar constantes ou o *firmware* (programa ou *software*, que define a funcionalidade do equipamento) [22]. Já a memória de dados pode armazenar as variáveis de um projeto qualquer, dados e também programas [16]. A Figura 2-6 apresenta a organização da memória da família de microcontroladores MCS-51 da Intel [16].



(a) (b)

Figura 0-6: Organização da memória da família de microcontroladores MCS-51 da Intel: endereçamento da memória de programa (a); endereçamento da memória de dados (b).
Fonte: [16].

Na Figura 2-6 (a), $\overline{EA}$ habilita ou não a busca de instruções e a execução (ciclo de instrução) na memória de programa externa. Na figura 2-6 (b), $\overline{RD}$ *e* $\overline{WR}$ são os sinais de entrada responsáveis por realizar a leitura e escrita na memória de dados externa. A Figura 2-6 mostra ainda quais as faixas de endereços para o acesso às memórias internas e externas, de programa e de dados, respectivamente.

O acesso à memória de dados pode ser realizado através de endereços de 8 *bits* pela CPU. Há ainda a possibilidade de acesso com a memória de dados externa de 16 *bits*, que se dá por meio do Ponteiro de Dados (DPTR).

O endereçamento da memória externa, seja de dados ou de programa, é realizada através das Portas P0 e P2, conforme ilustrado nas Figuras 2-7 e 2-8 [16].

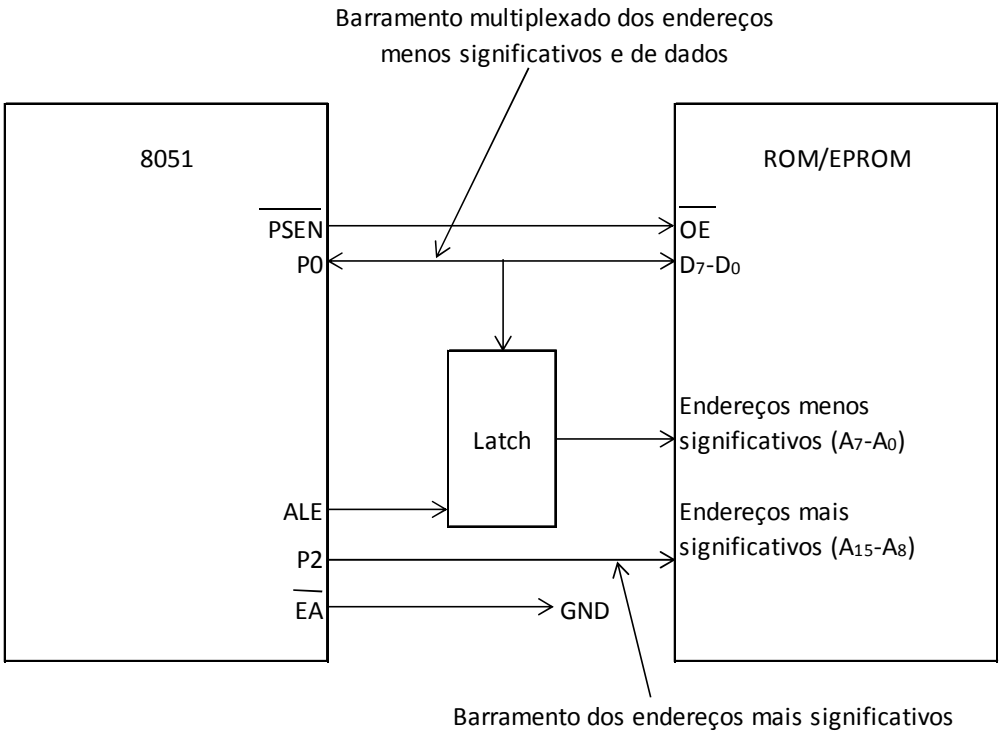


Figura 0-7: Exemplo de interfaceamento entre o 8051 e a memória de programa externa.
Fonte: [16].

Na Figura 2-7, as portas P0 e P2 funcionam como barramento de dados e endereços para a memória externa, ALE é um sinal de sincronismo gerado pelo microcontrolador para que seja possível fazer a demultiplexação do barramento de endereço menos significativo, que está multiplexado com o barramento de dados, $\overline{PSEN}$ é o sinal de leitura da memória de programa externa, A0 a A15 são os 16 *bits* do endereçamento da memória externa, D0 a D7 são os *bits* que formam junto à P0, o barramento de dados e $\overline{OE}$ habilita os *bits* D0 a D7 para a leitura do microcontrolador.

Na Figura 2-8, a porta P0 funciona como barramento dos endereços menos significativos e dados para a memória de programa e dados externa e P2 é a porta para paginação da RAM de 256 em 256 *bytes*, ALE é um sinal de sincronismo gerado pelo microcontrolador para que seja possível fazer a demultiplexação do barramento de endereço, menos significativo, que está multiplexado com o barramento de dados e $\overline{RD}$ *e* $\overline{WD}$ são os sinais utilizados para leitura e escrita para acesso da memória RAM externa, por meio de sinal gerado pela CPU.

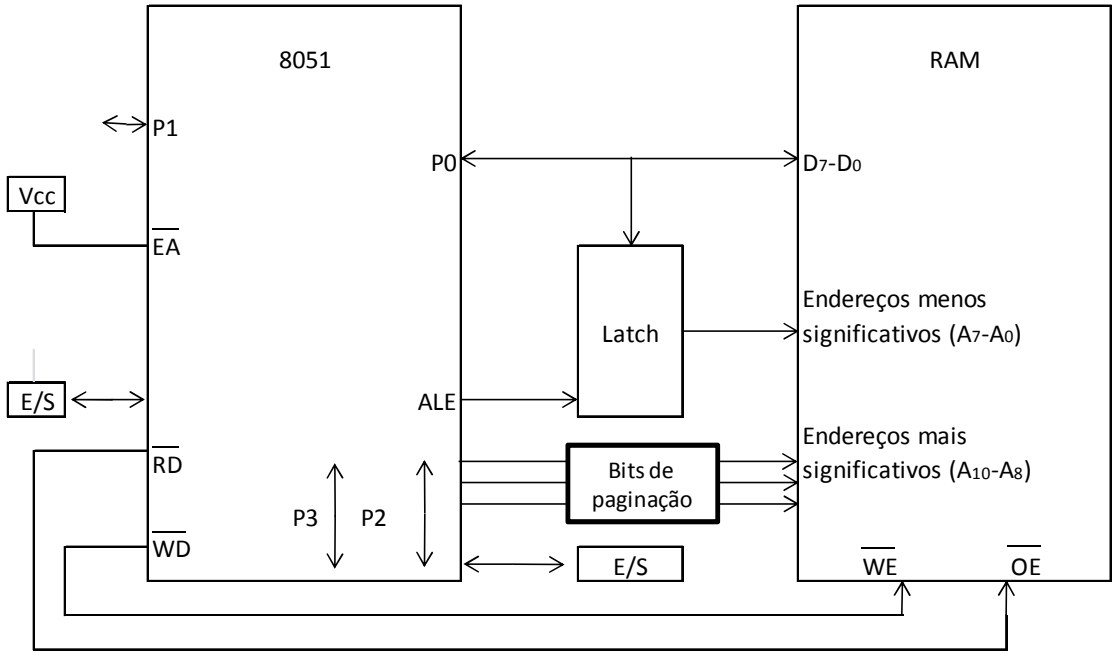


Figura 0-8: Exemplo de interfaceamento entre o 8051 e a memória de dados externa.
Fonte: [16].

1.4.2.2 O Núcleo do 8051

O núcleo ou também conhecido como *Core* ou microprocessador, contém uma CPU, que incorpora uma Unidade Lógica Aritmética (ULA) [16].

A CPU é otimizada para aplicações de controle, e opera com 8 *bits*. Realiza também processamento *booleano*, *bit* a *bit* [16].

Internamente, a CPU possui uma série de registradores com funções específicas. Esses registradores são definidos como Registradores de Funções Especiais (*Special Function Registers*, SFRs), já outros são de função geral, sendo esta arquitetura ilustrada na Figura 2-9 [16].

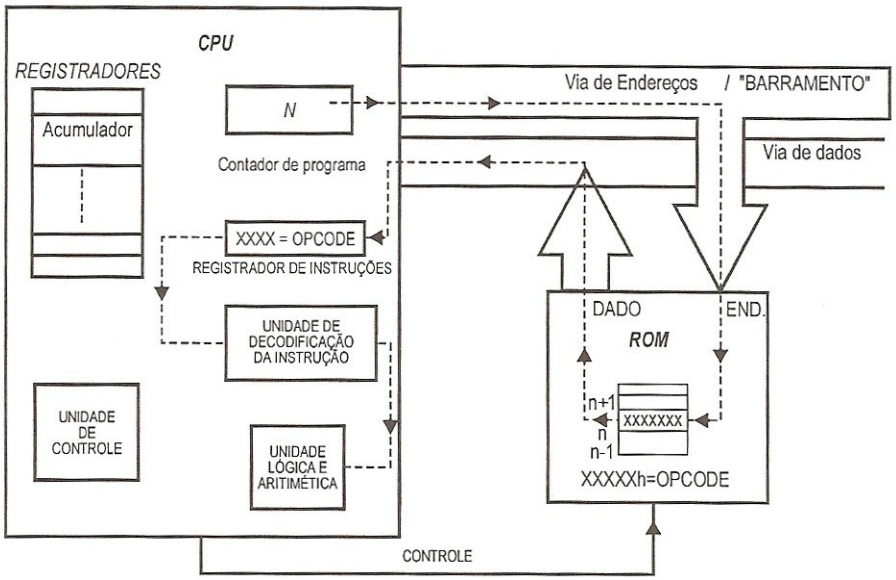


Figura 0-9: Arquitetura da CPU e o ciclo de busca da instrução.
Fonte: [24].

A Figura 2-9 apresenta um exemplo de um ciclo de busca de uma instrução, que através do barramento de endereços e dados internos, acessa a memória ROM, o registrador de instruções e a ULA. A ULA é responsável pela realização das operações lógicas, aritméticas e de comparação [16].

Este ciclo de busca de uma instrução funciona assim: o contador de programa aponta um endereço a ser buscado na memória externa. O *byte* lido na memória externa é armazenado no registrador de instruções. A unidade de decodificação da instrução interpreta a instrução lida do *byte* e envia esta informação para a ULA realizar a execução da instrução.

1.4.3 Sinal de *Clock* para o 8051

O sinal de *clock* é uma forma de onda do tipo "trem de pulsos" com período T e frequência f, que é inversamente proporcional a T e tem fundamental importância para o funcionamento do microcontrolador, pois define a velocidade do processamento, através do ciclo de máquina. O 8051 utiliza um oscilador externo com frequência máxima de 12 MHz na sua versão original [16].

1.4.4 As portas de entrada e saída do 8051

As portas de entrada e saída são as interfaces entre o 8051 e o mundo exterior [16]. São 32 *bits* que podem ser programados como entradas ou saídas individualmente, ou seja, elas são bidirecionais e podem também ser acionadas individualmente. Sua arquitetura apresenta 4 portas de 8 *bits* de registradores de função especiais [16]. São utilizadas para a interação com outros dispositivos e circuitos externos periféricos, como outros microcontroladores, memórias externas, chaves mecânicas, sensores, *displays*, entre outros [16].

Fisicamente, as saídas dessas portas são providas de *drivers* de corrente [16]. Geralmente, utilizam-se outros dispositivos externos para o chaveamento de cargas (transistores, tiristores, etc.). Já as entradas destas portas são providas de *buffers*. Há também um *latch* através de 8 *flip-flops* tipo D para armazenar a informação de um *byte* [16].

As quatro portas são denominadas de P0, P1, P2 e P3, respectivamente. Cada porta pode ter os seus *bits* identificados individualmente como por exemplo: o bit 5 da porta 0 é denominado P0.5 [16].

Algumas portas, além das funções bidirecionais, podem ser utilizadas para outras funções especiais, como a entrada de um sinal de *clock* externo, para os temporizadores e contadores, etc. Estas funções são acionadas através da configuração do *latch* de cada *bit* [16].

1.4.5 Programação em *Assembly*

Existem 3 níveis de abstração de linguagem de programação: a de baixo nível (*Assembly*), médio nível (linguagem C) e alto nível (Java, Basic, etc).

A linguagem *Assembly* é semelhante às instruções de máquina que o microcontolador executa. O seu conjunto de instruções é definido pelo fabricante. Isto permite que o programador controle toda a memória e os registradores internos. [16].

Se houver uma restrição de memória e velocidade de processamento no projeto, certamente a linguagem ideal a ser utilizada é a linguagem *Assembly* [16]. Ela possibilita um programa menor, e assim tem-se uma maior velocidade de processamento. Para programar em *Assembly* é necessário o conhecimento do *hardware* e da arquitetura interna do microcontrolador [16].

A geração do código de máquina se dá após a edição e a compilação de um programa [23]. As instruções, representadas por *bytes* ou conjunto de *bytes*, são comandos que são

executados pelo microcontrolador [23]. Quando um programa é implementado em uma determinada linguagem, deverá ser compilado para, então, ser transformado em linguagem de máquina, que pode ser simulado e gravado na memória de programa do *hardware* do equipamento, que executará as funções programadas [23].

Um exemplo de programação *Assembly* é mostrado na Figura 2-10.

```
MOV     A,R0
ADD     A,R3
ADD     A,55h
MOV     R6,A
```

Figura 0-10: Exemplo de um programa-fonte em *Assembly* de uma soma dos conteúdos dos registradores R0 e R3.
Fonte: [16].

Na Figura 2-10, o programa em *Assembly* executa uma adição (comando ADD) dos conteúdos dos registradores R0, R3 e do conteúdo da posição de memória 55h. Os conteúdos de R0, R3 e 55h são copiados (MOV) para o acumulador e somador. O resultado é copiado para R6.

1.5 Projeto de CIs digitais ASIC

Um projeto ASIC caracteriza-se pela integração de várias funções no mesmo CI [25]. A realização de um projeto ASIC pode ser motivada por oportunidades tais como: um novo mercado tecnológico, novas funcionalidades para um CI, circuitos integrados que proporcionem redução de custos, CIs com redução no consumo de energia elétrica e CIs com um melhor desempenho elétrico que outro em uso [25].

Ao iniciar um projeto, pode ocorrer uma dúvida entre optar por um projeto ASIC, ou a utilização de Dispositivos de Lógica Programável (*Complex Programmable Logic Device* - CPLD) [25]. Num projeto ASIC, você customiza o seu CI de acordo com as necessidades do projeto, criando um produto dedicado a esta aplicação. Já uma estrutura CPLD, pode receber a lógica de seu projeto, integrando-se fisicamente as portas lógicas internas e por fim apresentando o resultado desejado, podendo ainda ser modificado ou reusado [25].

O projeto ASIC apresenta vantagens em relação à velocidade de processamento, a potência dissipada, a produção em massa, a integração do leiaute e a dificuldade de realização da engenharia reversa (melhoria da segurança do projeto) [25]. Em contrapartida, tem como

desvantagens as várias semanas para fabricação dos CIs, o maior custo se for produzido em pequena escala e a impossibilidade de ser modificado de forma rápida, como é feito utilizando-se CPLD [25].

Os ASICs podem utilizar os IPs criados em outros projetos [25]. A utilização destes IPs geram custos de Engenharia Não Recorrente (*non-recurring engineering* - NRE), que são os custos de projeto e fabricação, ou taxas NRE, que serão pagas ao fabricante de CI. Além disso, altos investimentos em bons computadores e ferramentas CAD são necessários para o desenvolvimento e fabricação desse tipo de CI. ASICs são justificados somente para altos volumes de produção [25].

Com o aumento da demanda no mercado de CIs SoC, o fluxo de projetos de ASIC fica evidenciado, e assim os fabricantes de ferramentas de EDA disputam o mercado de forma a oferecer uma melhor funcionalidade e integração de suas ferramentas computacionais, proporcionando aos usuários uma maior facilidade de uso.

A seguir, é apresentado o fluxo de projeto típico de um CI ASIC, que qualquer projetista digital pode realizar a partir de uma necessidade qualquer.

1.5.1 O Fluxo de projeto de um ASIC típico

Basicamente, o fluxo de projeto de um circuito integrado digital compreende das seguintes etapas:

1 – especificação das funções do circuito integrado digital;

2 – descrição funcional;

3 – descrição estrutural;

4 – esquemático;

5 – simulação;

6 – leiaute;

7 – fabricação;

8 – testes [26].

Existem duas abordagens para o início do fluxo do projeto. A primeira é a chamada *Botton-up,* que parte do esquemático do circuito eletrônico. A outra é chamada de *Top-down*, que gera o leiaute a partir de uma descrição lógica, utilizando-se VHDL ou Verilog [27].

A abordagem *Bottom-up*, também conhecida como Projeto VLSI Personalizado (*Custom VLSI design*), permite o absoluto controle do leiaute, mas gera um alto esforço de projeto e dificuldade na verificação do funcionamento do CI devido a complexidade do

esquemático [25]. É indicada em casos de leiautes de alta performance e alto volume de produção [27].

Alternativamente, a abordagem *Top-down* utiliza as ferramentas EDA, que proporcionam maior velocidade na geração do leiaute do CI digital [25]. Estas ferramentas trazem automação ao processo, simplificando a geração de leiaute. Estes *softwares* tem funcionalidades para a geração de leiaute automático que proporcionam excelentes características de automação, e tem trazido muita qualidade ao fluxo de projeto [27]. A indústria vem adotando esta metodologia para seus projetos de CIs [27].

A Figura 2-11 apresenta o diagrama de blocos que mostra as duas diferentes formas de projetar um CI ASIC.

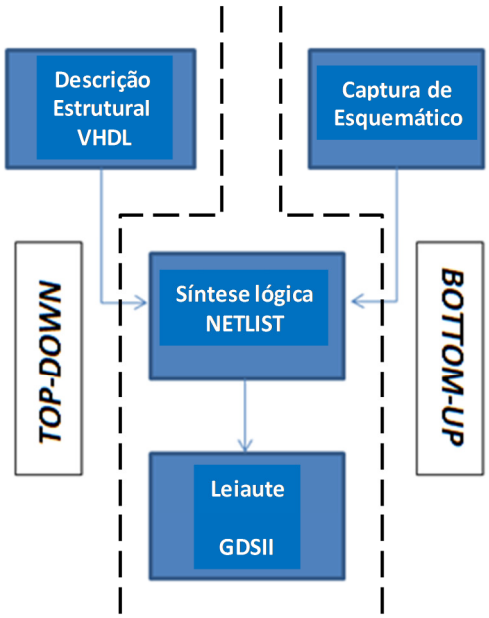


Figura 0-11: Diagrama de blocos das diferentes maneiras de se implementar um de CI ASIC.

Na Figura 2-11, verifica-se duas possibilidades de se projetar um CI ASIC. A metodologia *Top-down* parte do código em linguagem VHDL do CI e a metodologia *Botton-up* do seu esquemático. Posteriormente deve-se realizar a síntese lógica do CI (geração de um arquivo chamado *Netlist* que contém a descrição do CI em nível de portas lógicas e suas interligações). Por fim, deve-se gerar o leiaute do CI no formato *Graphic Data System* (GDSII), que contém as máscaras da tecnologia de fabricação do CI [28].

O projeto do *front-end* de um CI é definido como o desenvolvimento da lógica de entrada do projeto (descrição em VHDL ou Verilog, ou da captura de um esquemático), a compilação, a simulação e o teste do circuito integrado [28]. Existem muitas ferramentas de

front-end, e não necessariamente específicas para projetos ASIC. Geralmente, aplicações em FPGA ou em CPLD usam tais ferramentas, como por exemplo, o *Quartus*, da fabricante Altera [10].

Realizada a etapa de *Front-end*, existem dois métodos de geração de leiaute: 1 - *Full Custom,* que implementa de forma manual as geometrias dos dispositivos; 2 – Geração de Leiaute Automático, com ferramentas que automatizam o processo de geração de leiaute [28].

Leiautes de dispositivos implementados com polígonos de forma manual (*Full Custom)* são mais compactos, enquanto que leiautes gerados de forma automática (Projeto Automático), com células padrão, apesar de menos eficientes, apresentam uma maior velocidade na geração do leiaute do CI digital [28].

A maioria das ferramentas EDA permite dois modos de operação para geração de leiaute automático: 1 - o modo de Interface Gráfica do Usuário (*Graphical User Interface* - GUI), mais amigável e que segue o fluxo de projeto passo a passo; 2 - Modo de *Scripts,* que é definido por linhas de comandos, proporcionando maior velocidade e muito indicado em casos de repetitividade de blocos básicos que formam os CIs digitais [27].

1.5.2 Etapas do fluxo de projeto de um CI ASIC

O projeto ASIC pode ser dividido em duas etapas, independentemente do fabricante de ferramentas EDA escolhido, são elas: o projeto lógico e a implementação física [25].

A Figura 2-12 apresenta os passos necessários para cada uma das etapas, dentro do fluxo padrão de projetos de um CI ASIC comum a todas as ferramentas EDA [25].

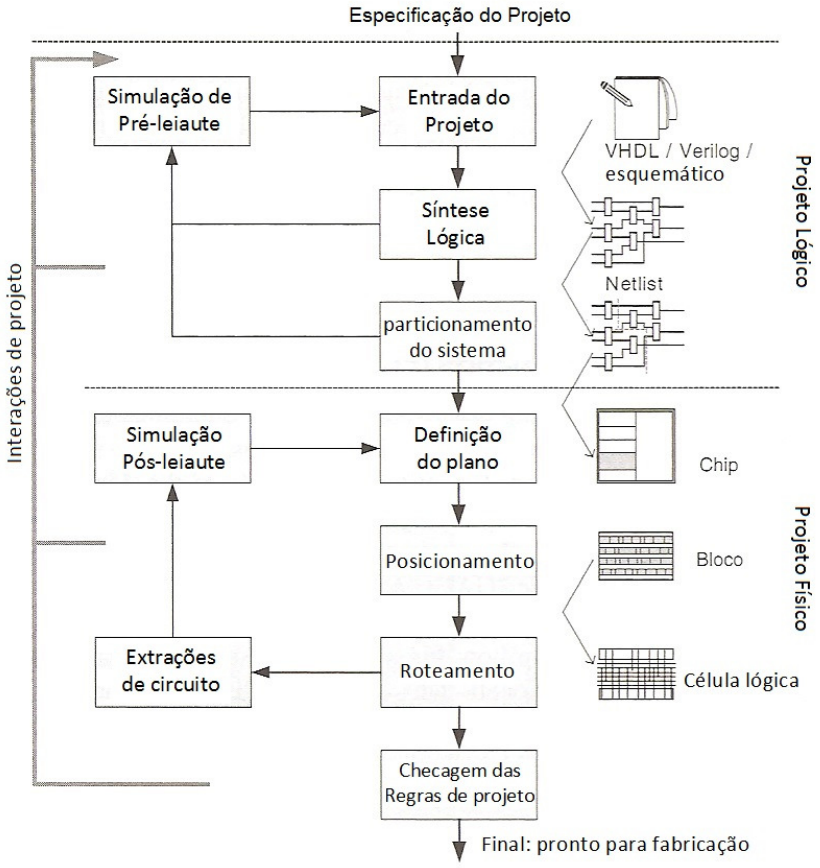


Figura 0-12: Fluxo do projeto de um CI ASIC.
Fonte: [25].

1.5.2.1 Projeto Lógico

Definidas as especificações e descrições do projeto, a entrada do projeto pode ser realizada de duas formas: 1 - captura de um esquemático; 2 - descrição de *hardware*, tipicamente através de VHDL ou Verilog [25].

O uso de entrada por meio de esquemáticos sempre foi uma técnica tradicional de projetos de CIs digitais. Contudo, a facilidade na descrição de um circuito integrado através das linguagens de descrição de *hardware*, apresenta-se como o método mais adequado aos dias atuais, devido à grande complexidade dos CIs [25].

Como a visualização do projeto é muito mais facilitada através do esquemático, muitas vezes a descrição lógica é utilizada como método de entrada do projeto, e transformada em esquemático para a compreensão do funcionamento e para a verificação de erros lógicos na descrição VHDL, dependendo do grau de complexidade e do tipo de circuito eletrônico [25].

A escolha do método de entrada em nada interfere no fluxo, ficando portanto como escolha do projetista, levando em consideração apenas o seu domínio por cada tipo [25].

Pode-se acrescentar nesta etapa uma simulação que o *software* realiza independente do fabricante, seja no esquemático, seja no VHDL, com o objetivo de detectar erros, curto-circuitos, etc [25].

O próximo passo do projeto lógico é o processo de síntese lógica, responsável por converter o esquemático na descrição de Transferência de Dados entre os Registradores (*Register Transfer Level* – RTL), uma descrição do circuito de baixo nível, que especifica como os dados fluem entre os blocos funcionais (registradores ou conjunto de registradores) presentes no circuito eletrônico. O processo de síntese simplifica as expressões booleanas equivalentes ao circuito, permitindo a redução da área do CI e otimizando o funcionamento (reduzindo o atraso na propagação de sinais, por exemplo). A partir da década de 80, as ferramentas de síntese têm evoluído e proporcionado maior qualidade e eficiência do projeto do CI digital [26].

O processo de síntese lógica resulta na geração de um *Netlist*, uma descrição das interligações entre os transistores em nível de porta lógica e registradores [26].

O código RTL pode ser descrito sem definir uma tecnologia determinada de fabricação de CI [26]. Assim, pode-se utilizar um mesmo projeto em outras diferentes tecnologias [26].

1.5.2.2 Projeto físico de um CI ASIC

A partir do código RTL gerado no projeto lógico, será iniciado o projeto físico ou implementação física do CI, para a criação do leiaute do CI [26].

O processo de criação do leiaute automático do circuito integrado utiliza algumas etapas, chamadas de definição do plano, posicionamento e roteamento de blocos e geração de células, que compõem o CI a partir das informações definidas no *Netlist* [28].

A definição do plano é o processo em que se estima a região do *chip* que será utilizada para cada bloco ou célula padrão do projeto [28]. É uma estrutura topológica que utiliza linhas e formas geométricas para guiar o posicionamento das células do projeto [28].

O posicionamento é o processo que posiciona as células padrão ou blocos nos planos definidos na etapa de definição de plano [28]. Este posicionamento requer cuidados, pois pode causar problemas principalmente na velocidade de *clock* [29]. Um mal posicionamento pode ocasionar atrasos nos sinais ao longo de todo o projeto [29]. Algumas técnicas de

posicionamento, em projetos mais específicos, podem ser utilizadas para minimizar ou solucionar problemas, como por exemplo posicionar o *clock* próximo às entradas [29].

Por fim, o roteamento é o processo de posicionamento e conexão dos caminhos dos sinais e alimentação dos blocos e das células padrão, ou seja, a interligação entre os blocos [28].

Opcionalmente, alguns projetos podem, ainda, utilizar uma etapa chamada de compactação, que consiste na minimização e otimização do tamanho do leiaute completo [28].

A verificação física é a etapa final da geração automática do leiaute. Esta etapa é divida em três partes: 1 - checagem das regras de projeto (*Design Rules Checker,* DRC), 2 – confronto do leiaute a partir do esquemático (*Layout Versus Schematic,* LVS) e 3 - extração das capacitâncias e resistências parasitas do leiaute do CI para posterior simulação final do CI [28].

A verificação física consiste em certificar-se que o leiaute, que está sendo gerado, está de acordo com os parâmetros estabelecidos pelo fabricante de circuitos integrados. [28].

A primeira parte da verificação física consiste na checagem das regras de projetos definidas pelo fabricante, através de um arquivo que possui as restrições de leiaute para a tecnologia selecionada [15]. Geralmente são restrições com relação às dimensões mínimas das regiões de porta, fonte e dreno, o espaçamento entre as geometrias, e a sobreposição das máscaras [15].

Estas informações são disponibilizadas no manual da tecnologia de fabricação de CI fornecido pelo fabricante. Um arquivo contendo as regras de projeto é fornecido pelo fabricante para ser carregado na ferramenta CAD, configurando a ferramenta para estar apta a realizar a verificação das regras de projetos no circuito digital implementado [15].

Desta forma, é possível executar uma verificação que confronta o leiaute com as regras de projetos de fabricação do CI. Toda vez que alguma parte do desenho estiver em desacordo, uma mensagem de erro é gerada, especificando qual regra não está sendo respeitada. A mensagem de erro facilita a compreensão do leiautista que, manualmente, corrige o leiaute [15].

Após a checagem das regras de projeto, realiza-se o processo de verificação, isto é, se o leiaute está de acordo com o esquemático. Nesta etapa da verificação física, será confrontado se o leiaute obtido da geração automática está em acordo com as ligações elétricas especificadas pelo esquemático do projeto [15].

Quando a criação do leiaute é realizada pelo método de geração automática, o fluxo de projeto já considera as restrições do projeto, minimizando muito a incidência de falhas detectadas pelas verificações de regras de projeto e confronto do leiaute a partir do esquemático [15].

A última etapa do processo de verificação consiste na extração das capacitâncias e resistências parasitas do CI. Conhecendo as dimensões das interconexões, são calculadas as resistências e capacitâncias parasitas. Posteriormente, podem-se realizar simulações com estes valores adicionados às entradas e saídas dos transistores que formam o CI. O resultado desta simulação avalia atrasos indesejados. Assim, pode-se otimizar o leiaute de forma a corrigir tal degradação. Isto é fundamental para que um circuito integrado seja fabricado de maneira adequada em relação às suas especificações [15].

1.5.2.3 Geração dos arquivos GDSII de leiaute do CI

Quando a criação do leiaute for finalizada e o leiaute estiver pronto para ser enviado ao fabricante de circuitos integrados, faz-se necessária a criação dos arquivos GDSII do leiaute do CI. Esses arquivos são o produto final do processo de geração automática do leiaute.

Estes arquivos, também conhecidos como arquivos de máscaras do CI, representam em formato binário as máscaras dos diferentes dispositivos (transistores, etc.), que são utilizadas no processo de fabricação do CI [26].

O arquivo de leiaute do CI obedece a um formato padrão, para que todos os fabricantes de circuitos integrados e também as ferramentas de CAD das diferentes empresas, possam abrir, entender e interpretar o leiaute gerado [15].

De acordo com a tecnologia selecionada, um mapeamento das máscaras será criado nos arquivos GDSII, que define uma ordem sequencial das máscaras que serão utilizadas para a fabricação dos CIs [15].

PROPOSTA DE UM FLUXO DE DESENVOLVIMENTO DE PROJETO DE CIS DIGITAIS NÃO COMPLEXOS, UTILIZANDO-SE FERRAMENTAS DE DEMONSTRAÇÃO COMPUTACIONAIS DE SIMULAÇÃO (*QUARTUS II*) E DE LEIAUTE DE CI (*MICROWIND II*)

Será apresentado neste capítulo, uma proposta inédita de simulação e geração de leiaute semiautomática de CIs, utilizando ferramentas computacionais de versão de demonstração, que possibilitem a realização de projetos de CIs.

Essa inovadora proposta é aplicada a um circuito digital de modulação de largura de pulso (*Pulse-Width Modulation* – PWM) de 3 entradas, como um primeiro estudo de caso. Serão percorridas todas as fases da metodologia proposta: concepção do projeto, criação e edição, compilação e simulação de um código VHDL, criação e simulação de um esquemático, criação de um arquivo Verilog, geração, edição e simulação de um leiaute e a geração dos arquivos para manufatura do Circuito Integrado Digital, utilizando as ferramentas de versão demonstração.

1.6 Proposta de um fluxograma da metodologia de desenvolvimento de um CI digital utilizando ferramentas de versão demonstração

A Figura 3-1 apresenta o fluxograma de uma proposta de desenvolvimento de um CI digital e as ferramentas computacionais de simulação e geração de leiaute utilizados em cada uma das etapas do projeto.

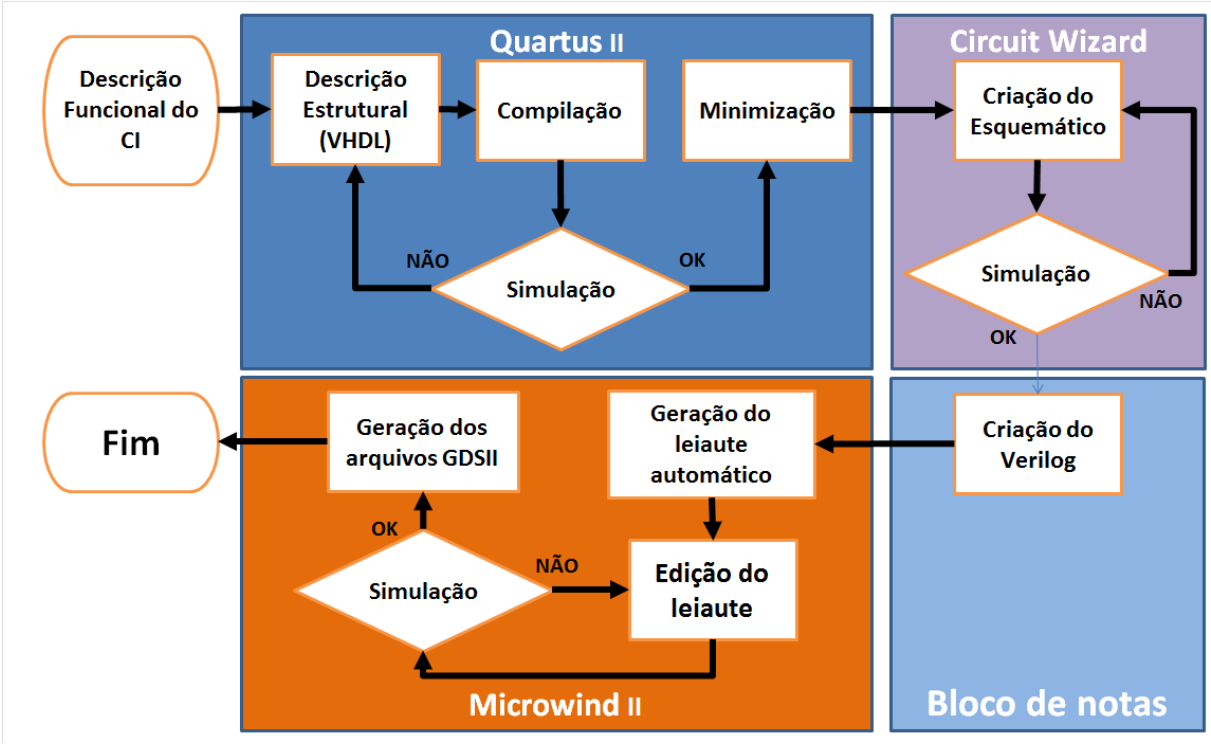


Figura 0-1: Fluxograma de projeto de CIs digitais básicos com ferramentas computacionais de demonstração.

A proposta consiste, inicialmente, na definição das especificações de projeto. De posse das especificações, deve-se gerar o correspondente VHDL desse CI através do *software Quartus II*. Com o VHDL deve-se gerar o esquemático e simular o seu funcionamento utilizando o *software Circuit Wizard*. Caso o resultado da simulação não esteja em acordo com as especificações do projeto, deve-se editar o código VHDL e realizar novamente a simulação até o correto funcionamento. Com o esquemático validado pela simulação, desenvolve-se o código Verilog equivalente utilizando qualquer editor de textos, pois o *Quartus II* não gera o Verilog compatível com o *Microwind II*. Este código Verilog será utilizado no *software Microwind II* para a geração do leiaute. Ainda no *Microwind II*, será realizada uma simulação elétrica do leiaute gerado. Caso o resultado da geração do leiaute ou a simulação não sejam satisfatórios ao funcionamento do circuito do CI, deve-se editar o código Verilog e repetem-se as etapas de geração e simulação de leiaute.

1.7 Aplicação da metodologia proposta para o desenvolvimento de um CI digital PWM

A metodologia proposta na Figura 3-1 será utilizada para o desenvolvimento de CI digital PWM como estudo de caso. As ferramentas computacionais utilizadas (*Quartus II, Circuit Wizard* e *Microwind II*) são gratuitas.

A seguir, cada uma das etapas da metodologia proposta será explicada através de exemplos de aplicação do projeto de CI digital PWM.

1.7.1 Especificação do PWM

O projeto consiste no desenvolvimento de um circuito integrado modulador de largura de pulso (PWM), que seja capaz de gerar oito larguras diferentes de pulso em sua saída, a partir de três entradas digitais (E0, E1 e E2).

Um contador binário de 0 a 7 é incrementado a cada borda de subida do *clock*. A saída permanece em 1 lógico enquanto o valor selecionado pelas entradas (E0, E1 e E2) for maior que o valor do contador. Quando esses valores forem iguais, o contador será zerado, portanto, a seleção realizada através das entradas E1, E2 e E3, especificará o tamanho da largura de pulso do sinal de saída, que é em função do *clock* do circuito. Quanto maior o valor binário (0 a 7) das entradas, maior é o tempo em que a saída permanecerá em 1 lógico.

A descrição funcional do circuito está ilustrada pelo diagrama de blocos apresentado na Figura 3-2 e a forma de onda de saída do PWM é apresentada em função do tempo para todos os estados possíveis das entradas E0, E1 e E2 na Figura 3-3.

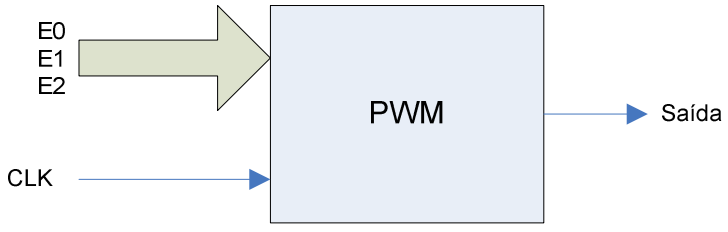


Figura 0-2: Diagrama de blocos do projeto de um CI PWM.

Na Figura 3-2, as entradas digitais E0, E1 e E2 selecionam a largura de pulso que será apresentada na saída do circuito PWM: a menor largura de pulso possível será dada pelo período do *clock* do circuito.

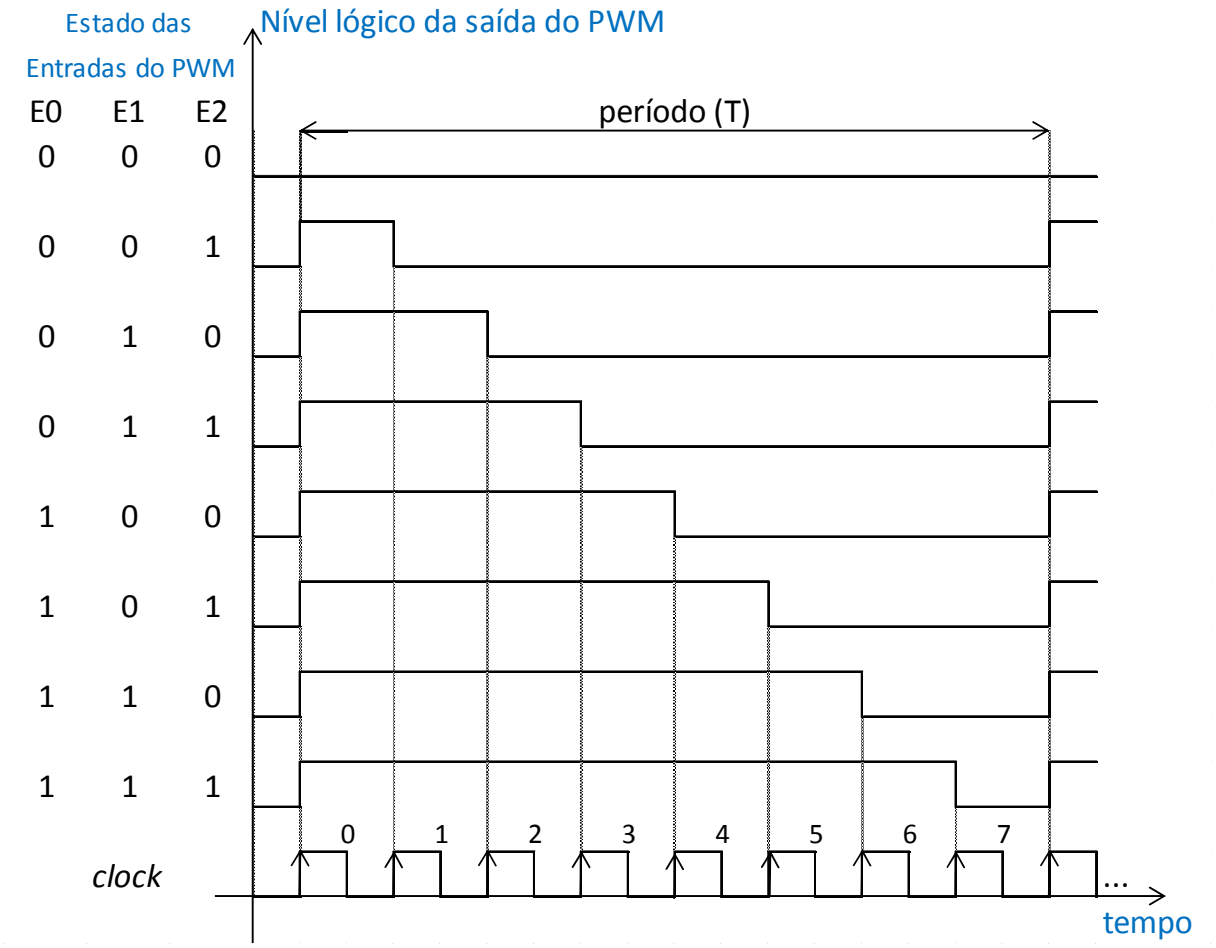


Figura 0-3: Forma de onda da saída do PWM em função do tempo, das entradas E0, E1 e E2 e do *clock*, respectivamente.

Na Figura 3-3, são ilustradas todas as possibilidades das entradas binárias E0, E1 e E2, que sincronizadas às bordas de subida do sinal de *clock,* resultam nas diferentes formas de onda da saída do PWM.

1.7.2 Descrição de *Hardware* em VHDL do PWM

Após a descrição funcional do circuito, a próxima etapa é a criação da descrição estrutural. Foi utilizada a linguagem VHDL, conforme está apresentado na Figura 3-4.

```
1   Entity PWM is
2       PORT
3               (
4               clock: IN bit;
5               ent: IN integer range 0 to 7;
6               sai: OUT bit
7               );
8   End PWM;
9
10  Architecture desenho of PWM is
11      Signal contador: integer range 0 to 7;
12      Begin
13          Process (clock)
14              Begin
15                  If clock ' event and clock = '1' then
16                          If contador < 7 then
17                          contador <= contador+1;
18                          else contador<=0;
19                          End if;
20                  End If;
21          End Process;
22          sai <= '1' when contador < ent else '0';
23      End desenho;
```

Figura 0-4: Descrição VHDL do circuito PWM.

No código VHDL da Figura 3-4, a Entidade PWM é declarada na linha 1. Esta entidade tem como entradas as variáveis "*clock"* (binária), e "ent" (inteiro que varia de 0 a 7), que são declaradas nas linhas 4 e 5, e a saída binária "sai" que é declarada na linha 6 como uma variável *bit*. A arquitetura da Entidade é a de um contador de 0 a 7, que é definida nas linhas 10 e 11. O funcionamento do circuito eletrônico é descrito pelas linhas 15 a 18, onde o contador é incrementado uma unidade, cada vez que a entrada "*clock*" passa de zero para um, até atingir sete. Na linha 22, a saída é habilitada em 1 lógico enquanto o valor do contador for menor que o valor colocado na entrada "ent".

Este código VHDL foi editado no *software Quartus* II da Altera, versão 8.1 *Web Edition* [10].

1.7.3 Elaboração do esquemático do PWM

A partir da descrição estrutural, o *software Quartus* II realiza uma sintetização lógica, gerando uma descrição de Transferência de Dados entre os Registradores (*Register Transfer Level* - RTL). O RTL é uma descrição em baixo nível do circuito eletrônico, especificando como os dados passam pelos registradores ou conjunto de registradores desse circuito. A partir da descrição RTL, é possível gerar com o *Quartus II* um esquemático correspondente a

esse circuito eletrônico, conforme está ilustrado na Figura 3-5. O procedimento para geração do esquemático encontra-se descrito no Apêndice C.

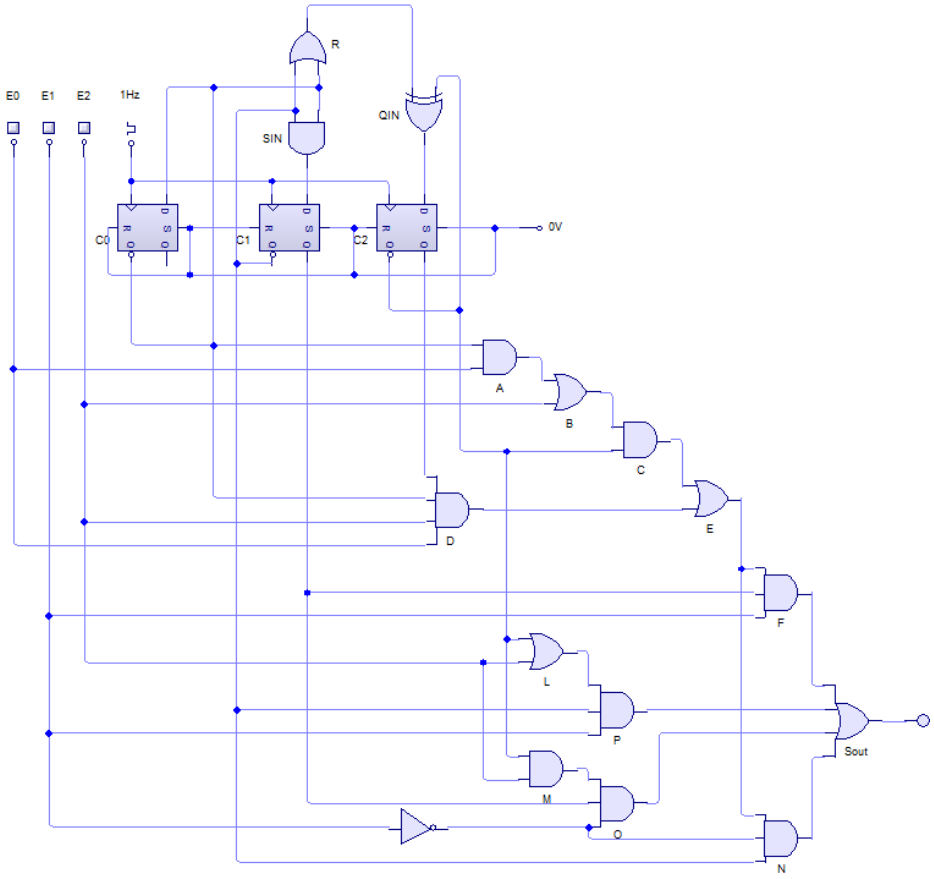


Figura 0-5: Esquemático do circuito PWM.

Na Figura 3-5 está ilustrado o esquema elétrico do circuito PWM que foi gerado pelo *Quartus II*, contendo diferentes portas lógicas (de A a F, de L à P, R, QIN, SIN e Sout) e *flip-flops* (C0, C1 e C2) do tipo D.

O circuito foi simulado no *software Circuit Wizard* [11] para teste de funcionalidade do circuito eletrônico do PWM. O *software Circuit Wizard* é utilizado na criação e simulação de circuitos digitais. Montado o esquemático, utilizando os componentes de sua biblioteca, realiza-se uma simulação de funcionamento, que permite a modificação dos estados das entradas (no caso deste circuito, E0, E1 e E2). A simulação demonstrou que a combinação binária do estados destas entradas modifica, o tempo em que a saída permanece em nível lógico 1, de acordo com a Figura 3-3.

1.7.4 Desenvolvimento do código Verilog e geração do leiaute do PWM

O método manual de criação de leiaute do circuito eletrônico do PWM já poderia ser iniciado com a utilização de um *software* gerador de máscaras, criando-se os MOSFETs (canal N e P) e a interligação entre eles, conforme o esquemático apresentado na Figura 3-5.

No caso deste projeto de dissertação de mestrado, foi proposta uma nova metodologia para a geração semi-automática de leiaute do circuito eletrônico PWM.

A ferramenta EDA de geração de leiaute utilizada foi o *Microwind II* [12]. Esta ferramenta possui um compilador para que através da inserção de um código em linguagem Verilog (uma outra linguagem de descrição de *hardware,* pois o *Microwind II* não aceita VHDL) seja gerado automaticamente o leiaute do circuito PWM.

A primeira etapa consistiu em criar um código em linguagem Verilog do circuito PWM. Conhecendo-se a sintaxe da linguagem Verilog, a partir do esquemático gerado através do *Quartus II* e validado pelo *Circuit Wizard* (Figura 3-5), este foi transcrito na linguagem Verilog, utilizando-se qualquer editor de textos. Este código Verilog é apresentado na Figura 3-6.

```
module PWM( E0,E1,E2,CLK,Sai);      //definição do módulo, suas entradas e saída

input CLK,E0,E1,E2;                 //declaração das entradas
output Sai;                         //declaração da saída
wire CLK_INV,CLK1_INV,CLK2_INV,CLK3_INV,CLK4_INV,CLK5_INV,CLK6_INV,CLK7_INV;
wire CLK8_INV,D_INV,D1_INV,D2_INV,S,R,S1,R1,S2,R2,R3,A,B,C,D,E,F,L,M,N,O,P,QIN,SIN
        ,E1_INV;                    // declaração das interconexões

//FLip Flop C2                      configuração interna do Flip flop C2

NOT n1 (CLK_INV, CLK);              // declaração padrão de uma porta inversora: CLK – entrada,
                                    //CLK_INV - saída
NOT n2 (CLK1_INV, CLK_INV);
NOT n3 (CLK2_INV, CLK1_INV);
NOT n4 (D_INV, QIN);
NAND nand1 (S, CLK,CLK2_INV,D_INV);  // declaração padrão de uma porta NAND: CLK,
                                      //CLK2_INV e D_INV – entradas, S - saída
NAND nand2 (R, CLK,CLK2_INV,QIN);
NAND nand3 (C2_INV, S,C2);          // entradas do flip flop: S e C2, saída do flip flop: C2_INV
NAND nand4 (C2, R,C2_INV);

//FLip Flop C1                      configuração interna do Flip flop C1

NOT n5 (CLK3_INV, CLK);
NOT n6 (CLK4_INV, CLK3_INV);
NOT n7 (CLK5_INV, CLK4_INV);
NOT n8 (D1_INV, SIN);
NAND nand5 (S1, CLK,CLK5_INV,D1_INV);
NAND nand6 (R1, CLK,CLK5_INV,SIN);
NAND nand7 (C1_INV, S1,C1);         //entradas do flip flop: S1 e C1, saída do flip flop: C1_INV
NAND nand8 (C1, R1,C1_INV);

//FLip Flop C0                      configuração interna do Flip flop C0

NOT n9 (CLK6_INV, CLK);
NOT n10 (CLK7_INV, CLK6_INV);
NOT n11 (CLK8_INV, CLK7_INV);
NOT n12 (D2_INV, C0_INV);
NAND nand9 (S2, CLK,CLK8_INV,D2_INV);
NAND nand10 (R2, CLK,CLK8_INV,C0_INV);
NAND nand11 (C0_INV, S1,C0);        //entradas do flip flop: S1 e C0, saída do flip flop: C0_INV
NAND nand12 (C0, R1,C0_INV);

//PORTAS LOGICAS            descrição do circuito formado pelas portas lógicas

AND and1(A,E0,C0_INV);
OR  or1(B,A,E2);
AND and2(C,B,C2_INV);
AND and3(D,C2,C0_INV,E0,E2);
OR  or2(E,C,D);
NOT not13(E1_INV,E1);
AND and4(O,C2_INV,E2,C1,E1_INV);
OR  or3(L,C2_INV,E2);
AND and5(P,L,C1_INV,E1);
AND and6(N,E,C1_INV,E1_INV);
AND and7(F,E,C1,E1);
OR  or4(R3,C1_INV,C0_INV);
XOR xor1(QIN,C2_INV,R3);
AND and8(SIN,C0_INV,C1_INV);
OR  or5(Sai,F,N,P,O);

ENDMODULE                   // fim do código
```

Figura 0-6: Código em linguagem Verilog do CI PWM.

O código Verilog do PWM é carregado no *Microwind II* através da opção "*Compile – Compile Verilog File*" e compilado. Na janela do *Microwind II* para a compilação do código Verilog, alguns parâmetros podem ser ajustados antes da geração do leiaute, como por exemplo, as dimensões dos MOSFETs (largura, W e comprimento, L), adição de *pads* para as entradas e saídas e as dimensões máximas da largura do leiaute. Clicando no botão "*Compile"* um leiaute é então gerado automaticamente.

A Figuras 3-7 apresenta todo o fluxo de projeto realizado para o circuito PWM, resultando no leiaute do circuito elétrico do PMW, utilizando-se a tecnologia Semicondutor-Metal-Óxido Complementar (*Complementary Metal-Oxide-Semiconductor,* CMOS) de 0,18μm convencional (*Bulk)*.

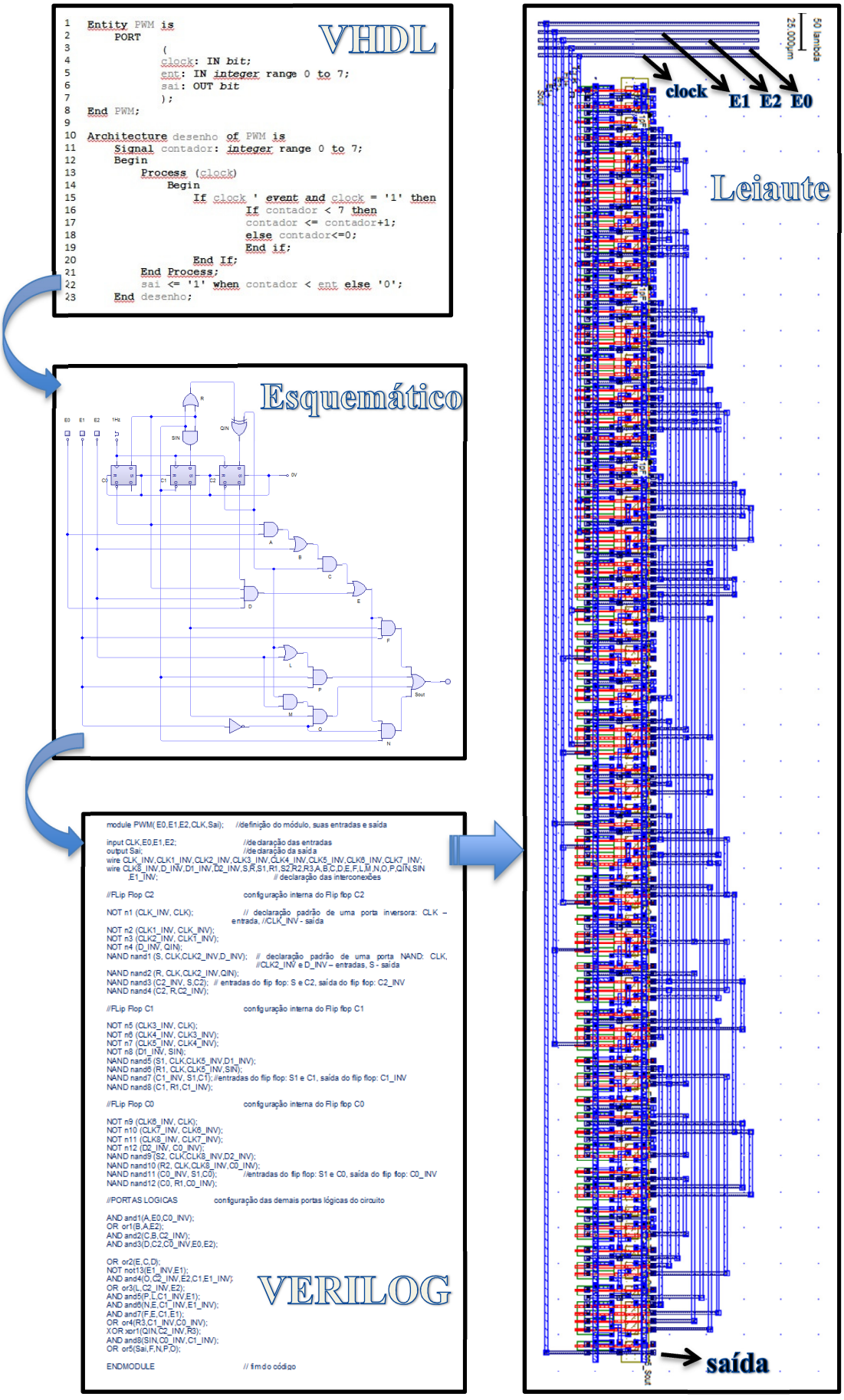


Figura 0-7: Fluxo do projeto de geração semiautomática do leiaute do circuito PWM.

1.7.5 Simulação do leiaute PWM no *Software Quartus II*

Após a geração semiautomática do leiaute do circuito integrado PWM, para garantir o seu funcionamento, foi realizada uma simulação das partes que contemplam o leiaute demonstrado (Figura 3-7). A seguir, será apresentada a verificação de apenas um *flip-flop*, as portas lógicas utilizadas neste projeto e, ao final, o circuito completo do PWM.

1.7.5.1 Verificação do leiaute de um *flip-flop* tipo D

A primeira parte do leiaute a ser simulada foi um dos *flip-flops* do tipo D que fazem parte do leiaute do circuito PWM, como mostra a Figura 3-8.

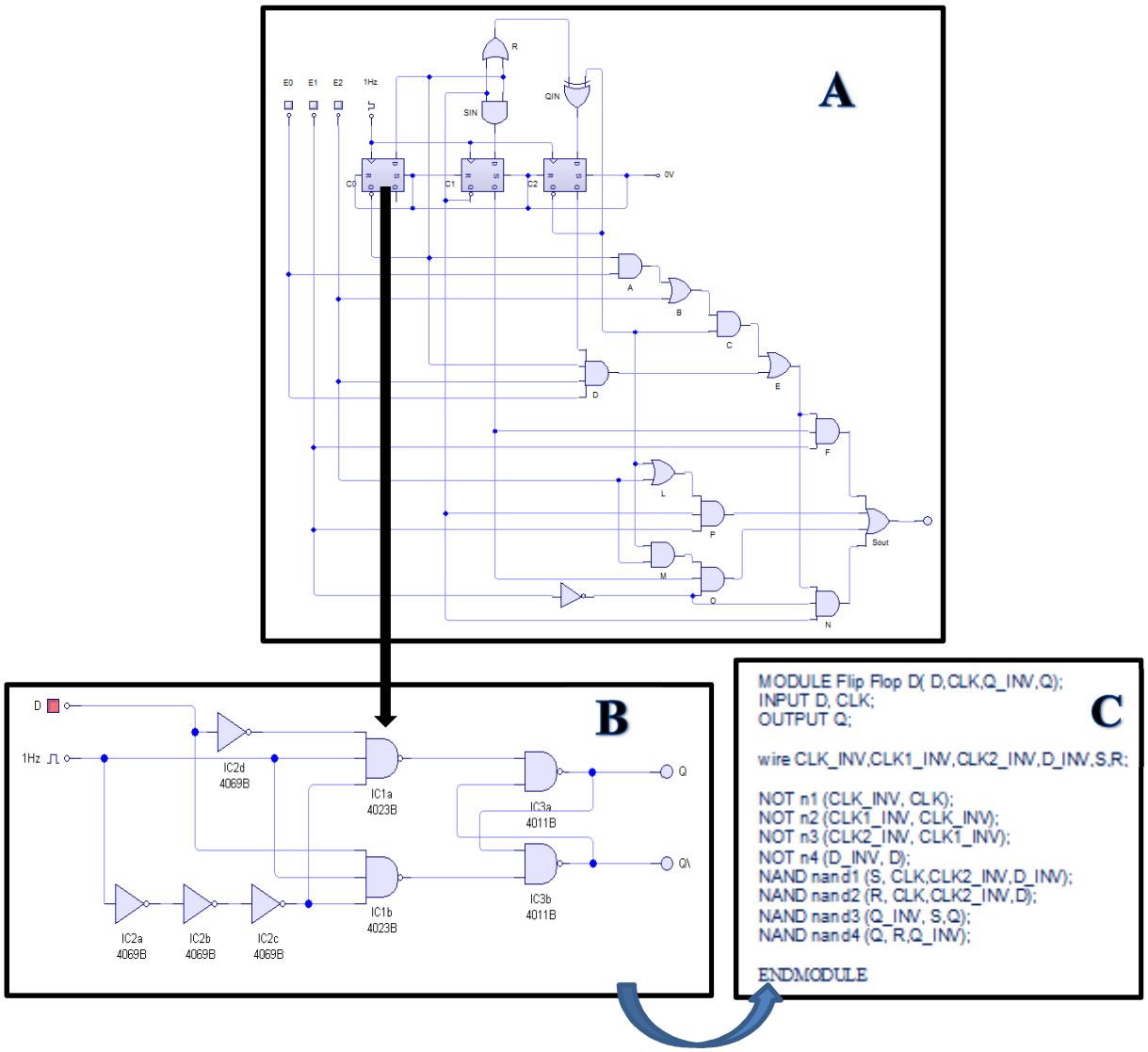


Figura 0-8: Esquemático do circuito PWM, circuito de um único *flip-flop* tipo D e o correspondente código Verilog.

Na Figura 3-8 apresentada acima, pode-se visualizar: A - o esquemático completo do circuito elétrico do PWM; B – o esquema elétrico em nível de portas de um dos *flip-flops* (C0) que compõem o circuito PWM; C – o código Verilog equivalente ao esquema elétrico do *flip-flop*.

A Figura 3-9 ilustra o leiaute gerado automaticamente, pelo *Microwind II* a partir do código Verilog do *flip-flop*.

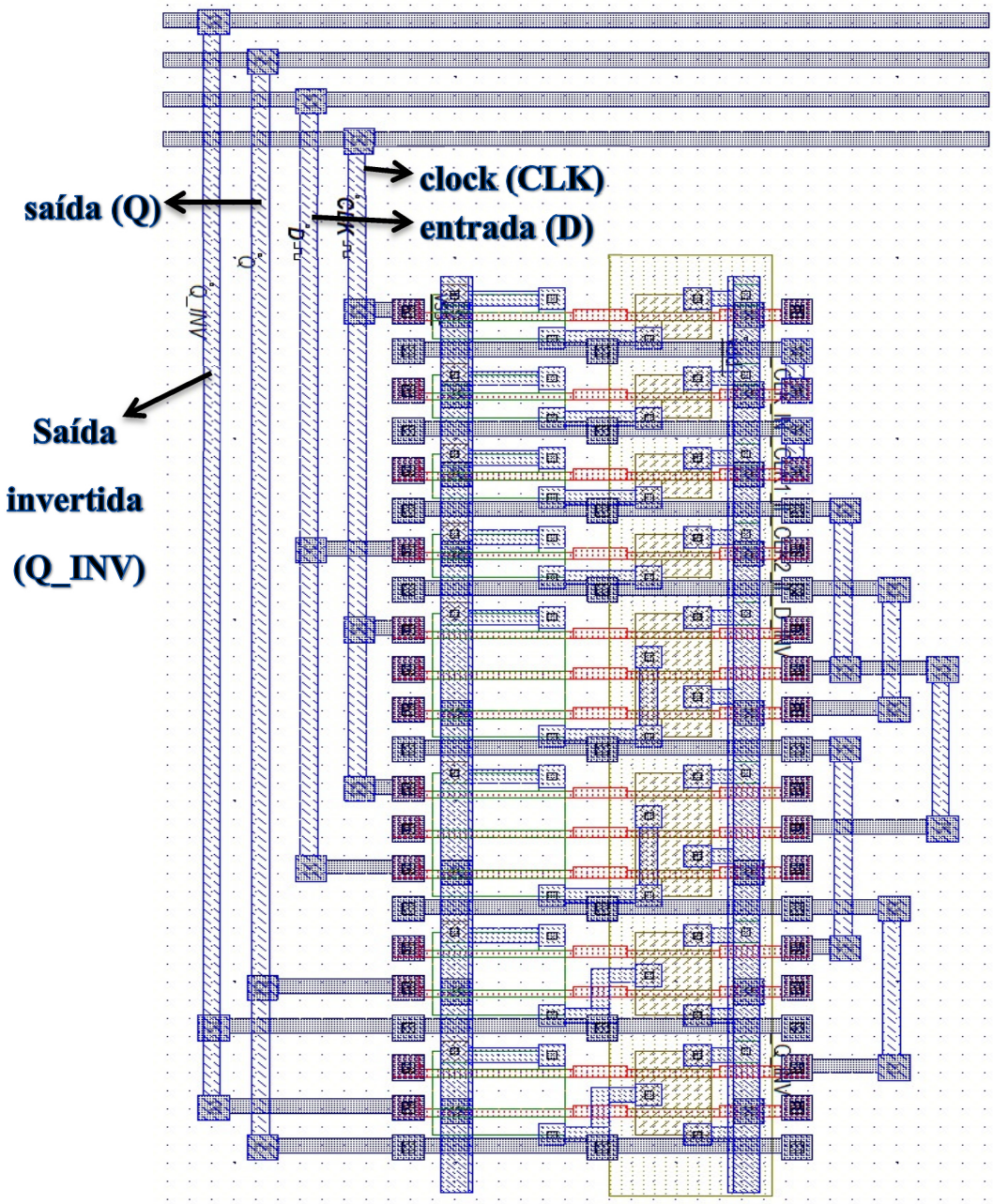


Figura 0-9: Leiaute do *flip-flop* gerado automaticamente pelo *Microwind II*.

A Figura 3-10 ilustra as formas de onda dos sinais CLK, D, Q_INV e Q, resultante da simulação temporizada que foi realizada no *Microwind II*. Após a geração do leiaute, é possível utilizar a ferramenta de simulação deste *software*, com a escolha dos parâmetros dos sinais que serão aplicados nas entradas (CLK e D) e obter as formas de onda das saídas do *flip-flop* (Q_INV e Q):

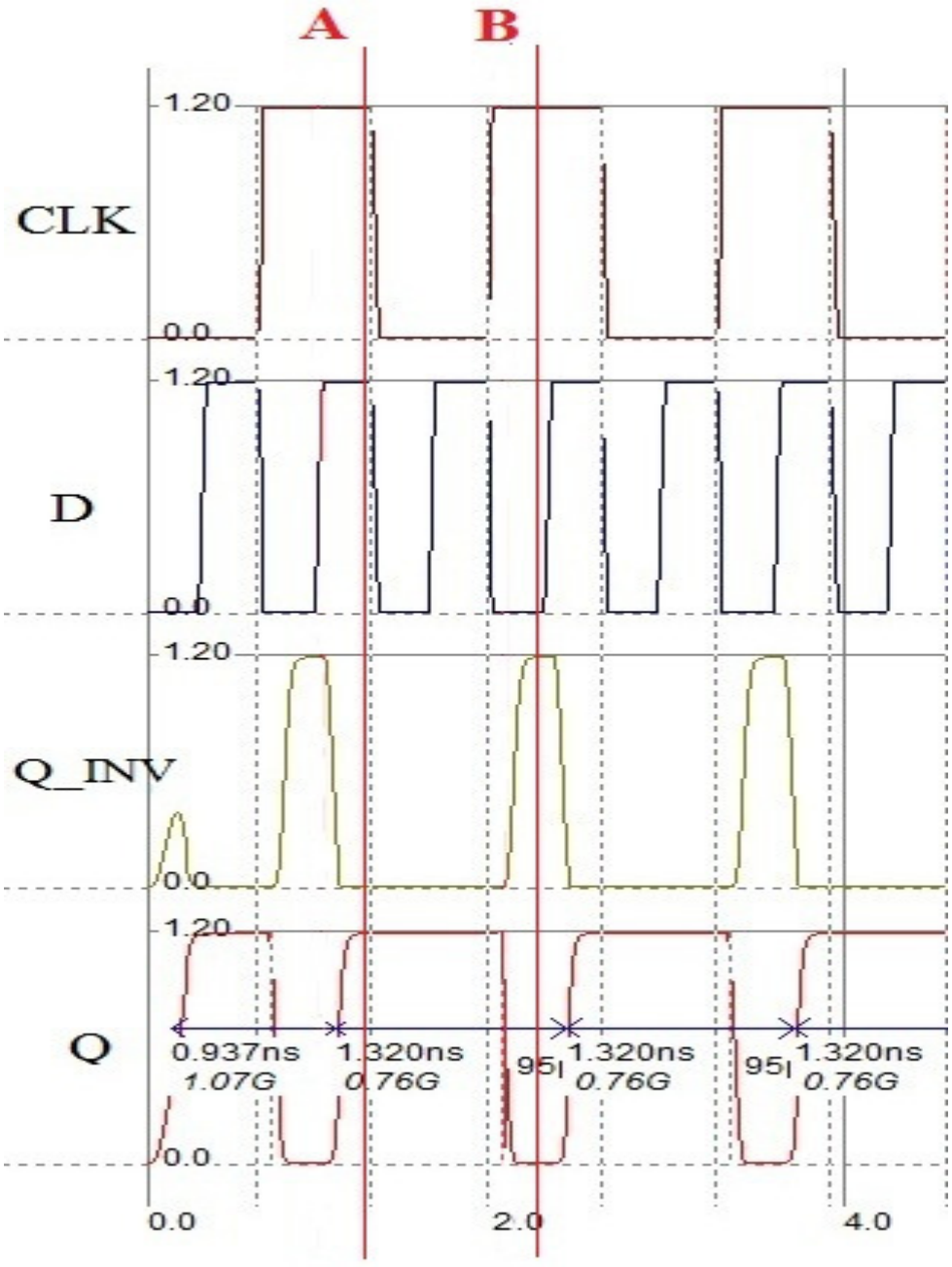


Figura 0-10: Simulação temporizada dos sinais de entrada e saída do bloco *flip-flop*.

Na Figura 3-10, pode-se observar os sinais de *clock* (CLK) e de entrada (D) gerados através do *Microwind II* para simular o funcionamento do circuito. Os sinais Q_INV e Q são as saídas do bloco *flip-flop* e de acordo com os estados de CLK e D, o sinal Q_INV é sempre

o inverso de Q. No primeiro instante (indicado como A no gráfico da Figura 3-10) em que D e CLK estão em nível lógico 1, a saída Q é acionada em nível lógico 1. No ponto indicado como B, no gráfico da Figura 3-10, quando a entrada D é modificada para nível lógico 0 e o sinal de *clock* (CLK) muda de 0 para 1, a saída Q é desligada, ou seja, vai para nível lógico 0.

O leiaute do *flip-flop* tido D gerado de maneira semiautomática pelo *Microwind II*, após esta simulação temporizada demonstrou-se correto e funcional para a implementação do circuito PWM.

1.7.5.2 Verificação do leiaute semiautomático das portas lógicas

Outras simulações foram realizadas com as demais portas lógicas que fazem parte do circuito elétrico do PWM. A Figura 3-11 apresenta: A - o esquemático completo do circuito elétrico do PWM; B – o esquema eletrônico criado com as portas lógicas do tipo E, OU, NE, NOU, Inversora e OU Exclusivo que compõem o circuito PWM e foram selecionadas para a verificação de leiaute gerado através da técnica de geração semiautomática; C – o código Verilog equivalente ao esquema elétrico destas portas lógicas.

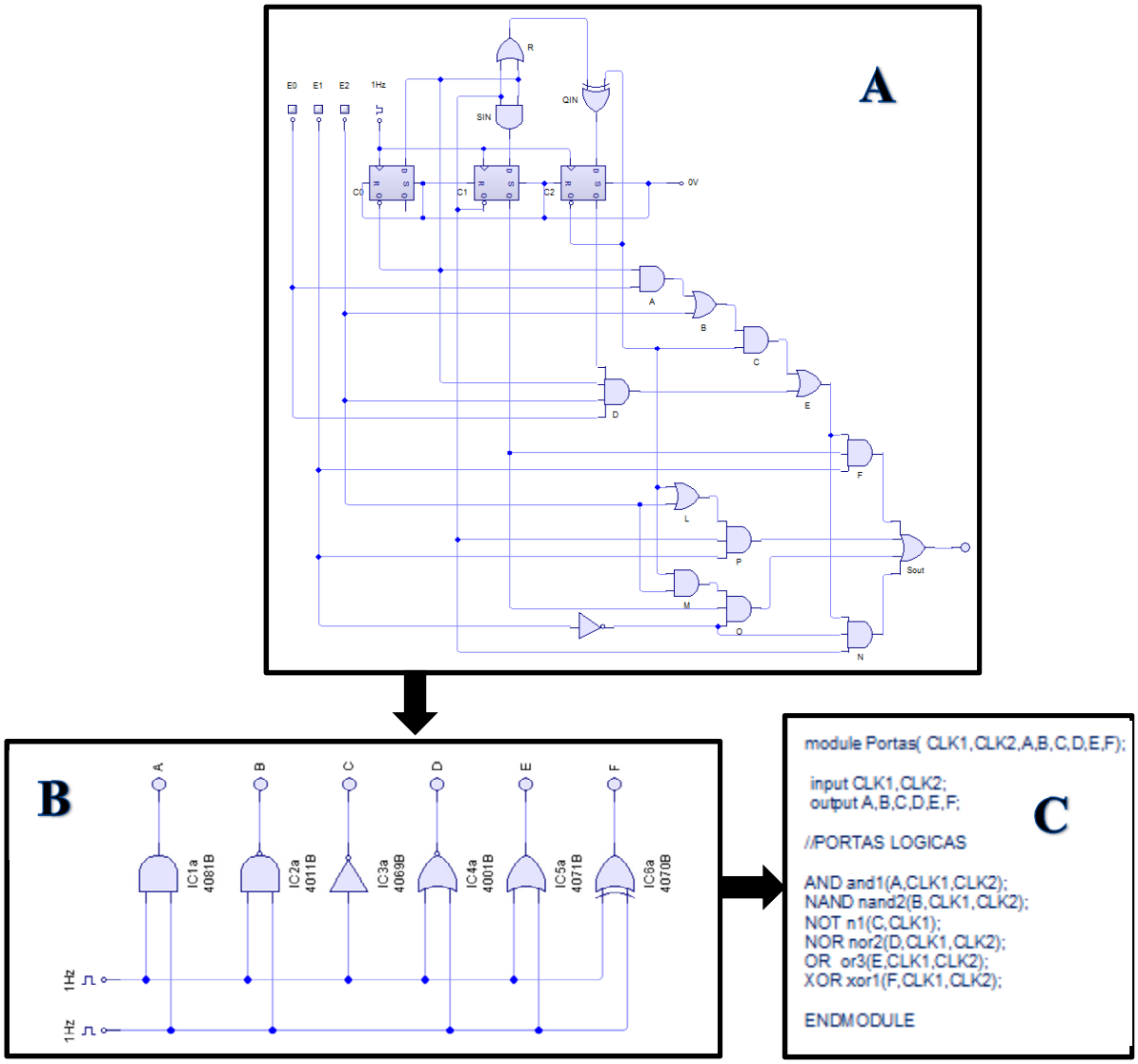


Figura 0-11: Esquemático do circuito PWM e circuito e código Verilog das portas lógicas do tipo E, NE, Inversora, NOU, OU e OU Exclusivo.

Nota-se que para a porta inversora, apenas um sinal de *clock* é necessário para a simulação.

A Figura 3-12 ilustra o leiaute gerado, automaticamente, pelo *Microwind II*, a partir do código Verilog das portas lógicas da Figura 3-11.

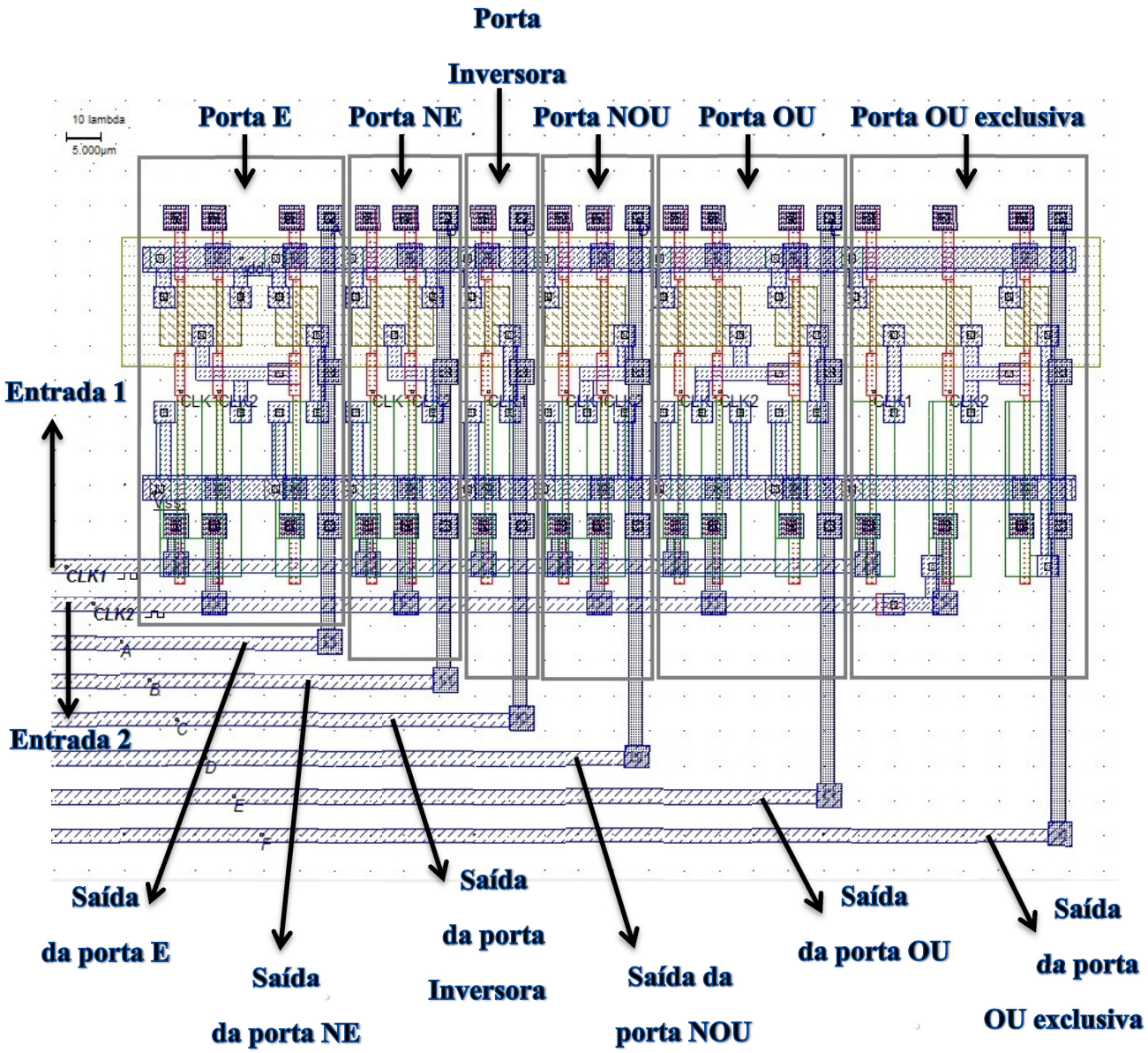


Figura 0-12: Leiaute gerado automaticamente pelo *Microwind II* das portas lógicas do tipo E, NE, Inversora, NOU, OU e OU Exclusivo.

A Figura 3-13 ilustra as formas de onda dos sinais de entrada CLK1 e CLK2, e os sinais de saída A, B, C, D, E e F, resultante da simulação temporizada que foi realizada no *Microwind II*.

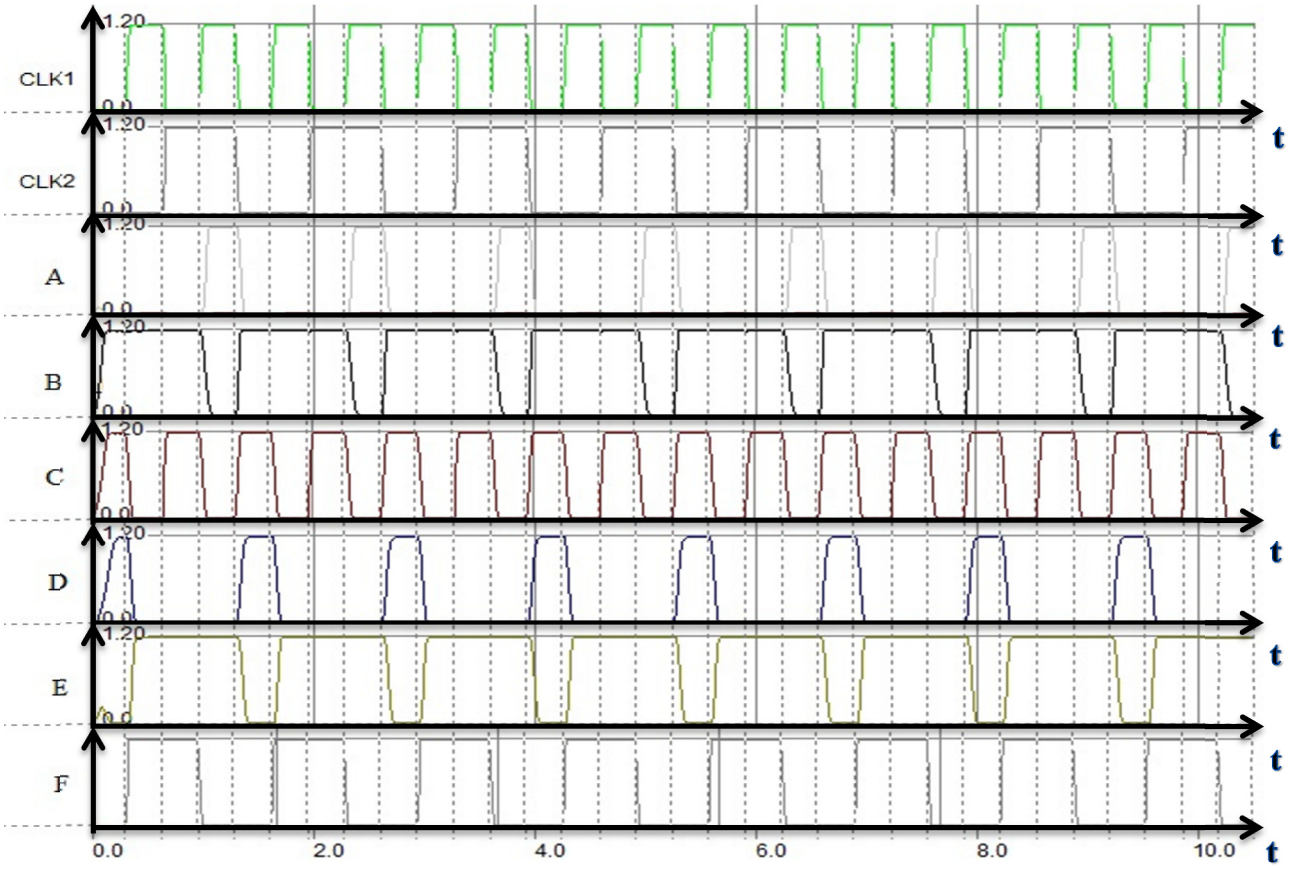


Figura 0-13: Simulação temporizada dos sinais de entrada e saída das portas lógicas do tipo E, NE, Inversora, NOU, OU e OU Exclusivo.

Na Figura 3-13, foram gerados dois sinais de *clock*, CLK1 e CLK2, para a simulação das entradas das portas lógicas. Conforme a combinação binária destas entradas, é gerado o sinal de saída, de acordo com a Tabela da Verdade de cada porta. A forma de onda A é o sinal resultante da porta lógica do tipo E. A forma de onda B é o sinal resultante da porta lógica do tipo NE. A forma de onda C é o sinal resultante da porta lógica do tipo Inversora. A forma de onda D é o sinal resultante da porta lógica do tipo NOU. A forma de onda E é o sinal resultante da porta lógica do tipo OU. Portanto a simulação elétrica provou o funcionamento do leiaute.

1.7.5.3 Verificação do leiaute completo do circuito PWM

O leiaute completo do circuito PWM também foi simulado. Acrescentando-se capacitores de 0,1pF no sinal de *clock,* foi gerado um pequeno atraso suficiente para a simulação acontecer com sucesso. A figura 3-14 apresenta o resultado da simulação.

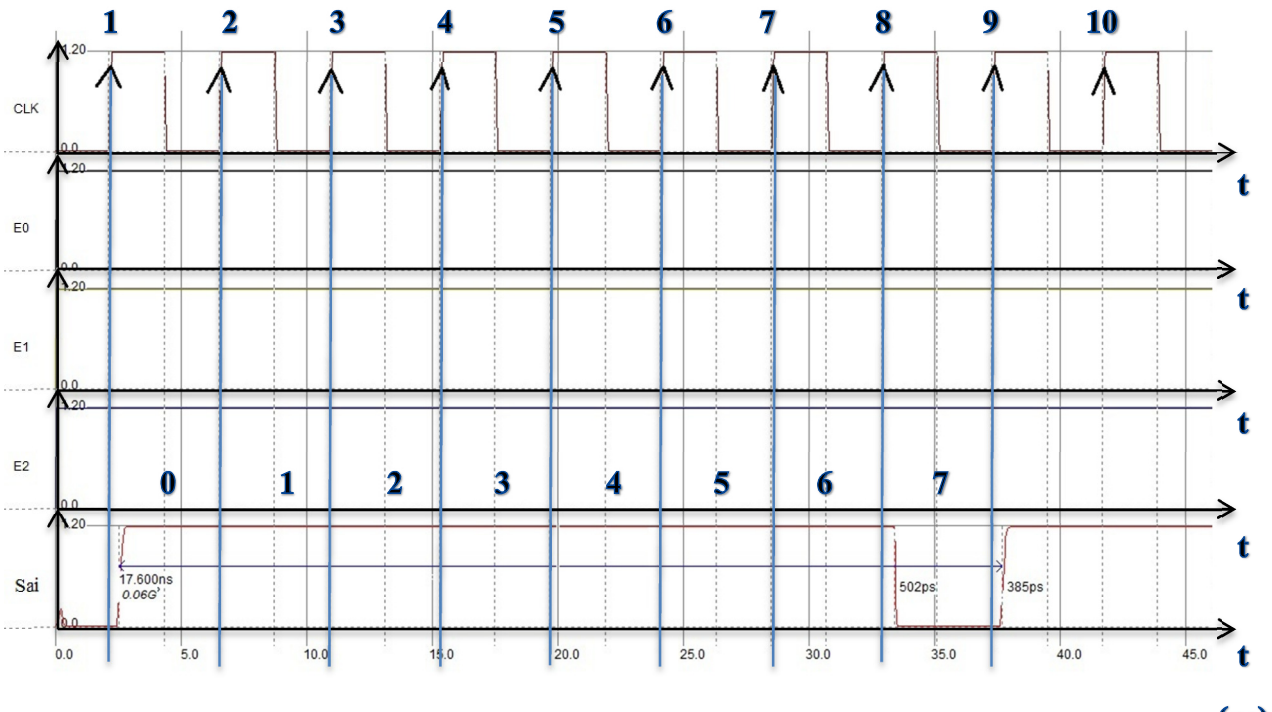


Figura 0-14: Simulação temporizada dos sinais de *clock* e saída do leiaute do circuito PWM para a condição de E0, E1 e E2 igual a 1.

Na Figura 3-14, o sinal de *clock* CLK é gerado pelo *Microwind II* para a simulação do circuito PWM. A forma de onda do sinal de saída (Sai) representa a saída do circuito PWM. Os sinais E0, E1 e E2 são os sinais de entrada do circuito, todos em nível lógico 1, representando 7 no contador, portanto a saída permanece em nível lógico 1 durante 7 ciclos de *clock.*

Entretanto, foi observado que a ferramenta *Microwind II* é bastante limitada para leiautes contendo muitos MOSFETs. Por tratar-se de uma ferramenta didática, não são considerados todos os efeitos das capacitâncias, resistências e indutâncias [12]. O *Microwind II* é próprio para aplicações educacionais [12]. Caso haja necessidade de validação de um leiaute industrial, deve-se utilizar uma ferramenta profissional [12].

Após a geração semi automática do leiaute pelo *Microwind II* deve-se utilizar a ferramenta para conferência das regras de projetos. Esta ferramente indicará possíveis erros de acordo com a tecnologia e *foundry* selecionadas. Deve-se então realizar uma edição manual do leiaute. Foi observado também que o método de geração semi automática do *Microwind II* não posiciona, não se sabe porque, as aberturas de contatos no local adequado da região ativa do transistor, e que são acrescentadas algumas máscaras e vias entre máscaras sem função ou desnecessárias que também devem ser editadas manualmente.

Deve-se salientar também que no momento da importação do arquivo Verilog pelo *Microwind II*, deve-se optar pelas dimensões de W dos transistores do tipo N maiores que as dimensões de W dos transistores do tipo P, devido a diferença de mobilidade de portadores.

Com o objetivo de uma simulação mais completa do funcionamento do circuito PWM, foi realizada também uma simulação funcional no *Quartus* II. Foram simuladas a patir do esquemático todas as possibilidades de combinações das três entradas e, nas Figuras 3-15, 3-16 e 3-17, são apresentados os resultados das simulações cujas entradas foram combinadas em 0, 5 e 7.

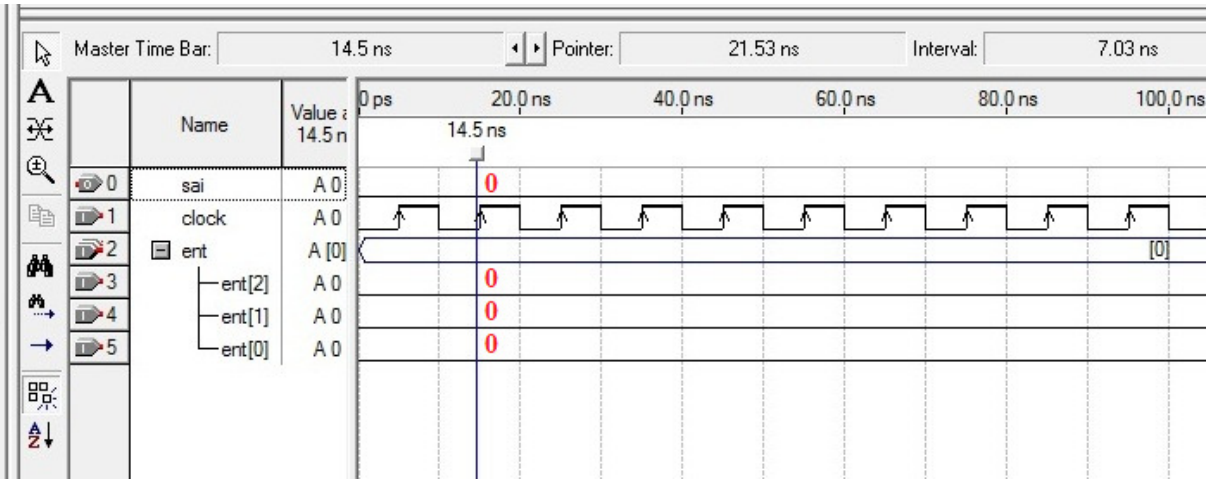


Figura 0-15: Resultado da simulação funcional do circuito PWM com as entradas combinadas em 0.

Na Figura 3-15, os sinais de entradas ent[0], ent[1] e ent[2] estão em nível lógico 0. Esta combinação binária resulta na contagem de 0 pulsos do sinal de *clock.* Portanto, o sinal de saída "sai" permanece sempre em nível lógico 0.

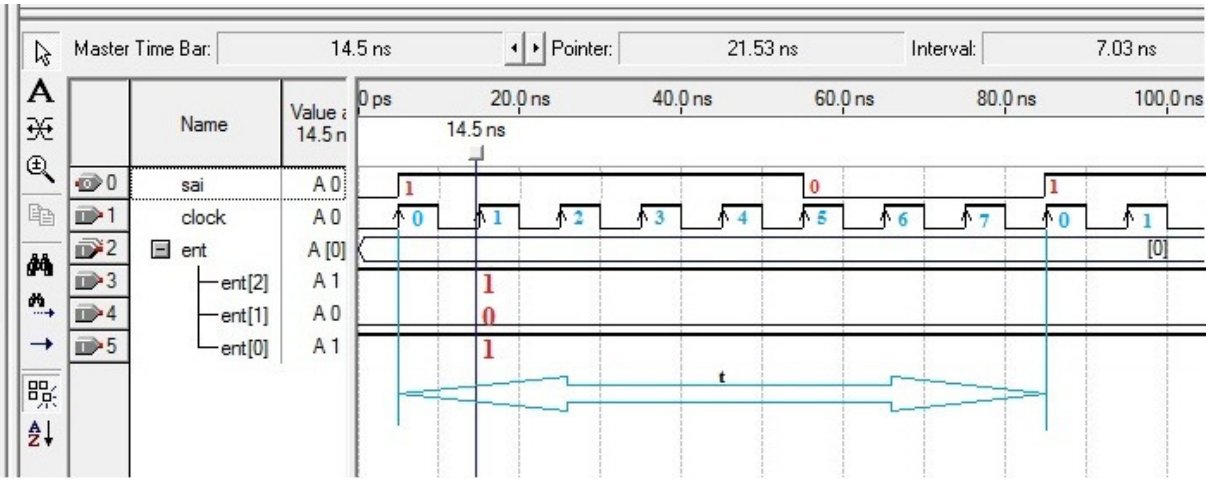


Figura 0-16: Resultado da simulação funcional do circuito PWM com as entradas combinadas em 5.

Na Figura 3-16, os sinais de entradas ent[0] e ent[2] estão em nível lógico 1, enquanto que o sinal de entrada ent[1] está em nível lógico 0. Esta combinação binária resulta na contagem de 5 pulsos do sinal de *clock.* Portanto, o sinal de saída "sai" permanece em nível lógico 1 durante 5 sinais de *clock*, retornado para o nível lógico 0 nos demais 3 ciclos de *clock*.

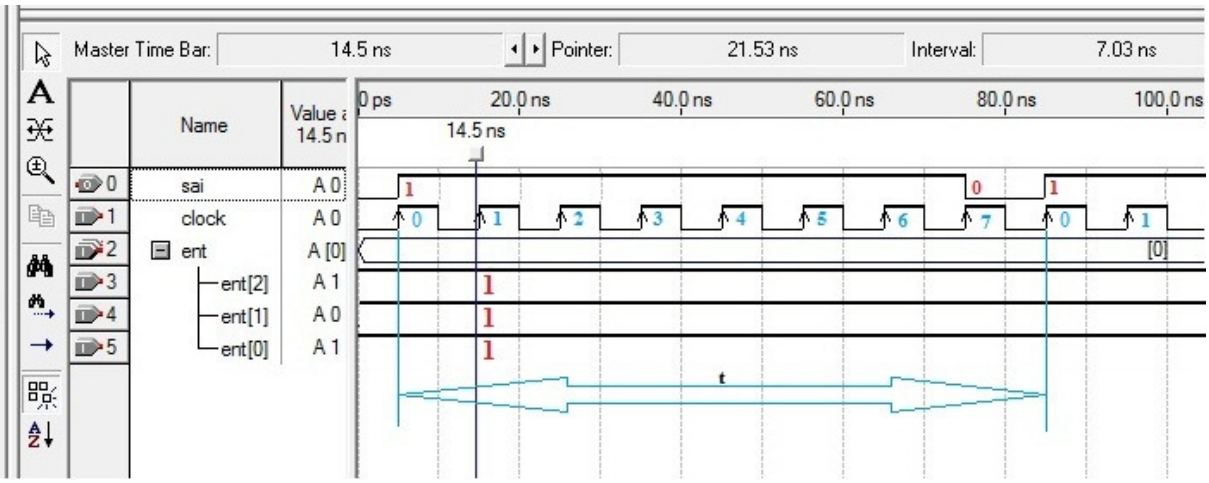


Figura 0-17: Resultado da simulação funcional do circuito PWM com as entradas combinadas em 7.

Na Figura 3-17, os sinais de entradas ent[0], ent[1] e ent[2] estão todos em nível lógico 1. Esta combinação binária resulta na contagem de 7 pulsos do sinal de *clock.* Portanto, o sinal de saída "sai" permanece em nível lógico 1 durante 7 ciclos de *clock*, retornado para o nível lógico 0 por 1 ciclo de *clock*.

GERAÇÃO AUTOMÁTICAS DOS LEIAUTES DOS BLOCOS ULA E RAM DO MICROCONTROLADOR 8051

Este capítulo tem por objetivo conceber os leiautes dos blocos ULA e RAM do microcontrolador 8051 de forma automática.

Será utilizado o fluxo de projeto para CI ASIC ilustrado na Figura 2-12, mas ao invés da utilização de ferramentas de demonstração, como feito no Capítulo 3, este capítulo percorre o fluxo utilizando ferramentas profissionais da *Mentor Graphics*.

Ao invés da simulação de pré-leiaute proposta pelo fluxo de projeto no projeto lógico, o microcontrolador 8051 foi implementado em uma placa didática de FPGA, para a validação do código VHDL e o teste de funcionamento do projeto de microcontrolador como um todo.

As ferramentas de leiaute da *Mentor Graphics* foram disponibilizadas no Centro Universitário da FEI e adaptadas ao fluxo de projeto para CI ASIC, conforme o Programa de Ensino Superior (*High Education Program*, HEP) do próprio fabricante das ferramentas de CAD.

1.8 Implementação em FPGA de um microcontrolador 8051 a partir de um código VHDL

O microcontrolador 8051 foi escolhido para ser implementado neste trabalho, devido a sua patente já ser de domínio público e seu código VHDL estar aberto e disponível para consulta nos mais variados sítios (*sites*) de concepção de circuitos integrados, tais como o Opencores.org [30].

A robustez de seu *hardware* e os recursos de CPU, memórias e entradas e saídas disponíveis trazem um atrativo para diversas aplicações em eletrônica [16].

Devido à complexidade de sua arquitetura, foram selecionadas algumas partes específicas para a elaboração do leiaute, que é descrito neste trabalho. Assim, este projeto se dedica à criação do leiaute da memória RAM e da Unidade Lógica Aritmética (ULA). A memória RAM terá uma capacidade de 128 *bytes* conforme a arquitetura do 8051 apresentada na Figura 4-1 [31].

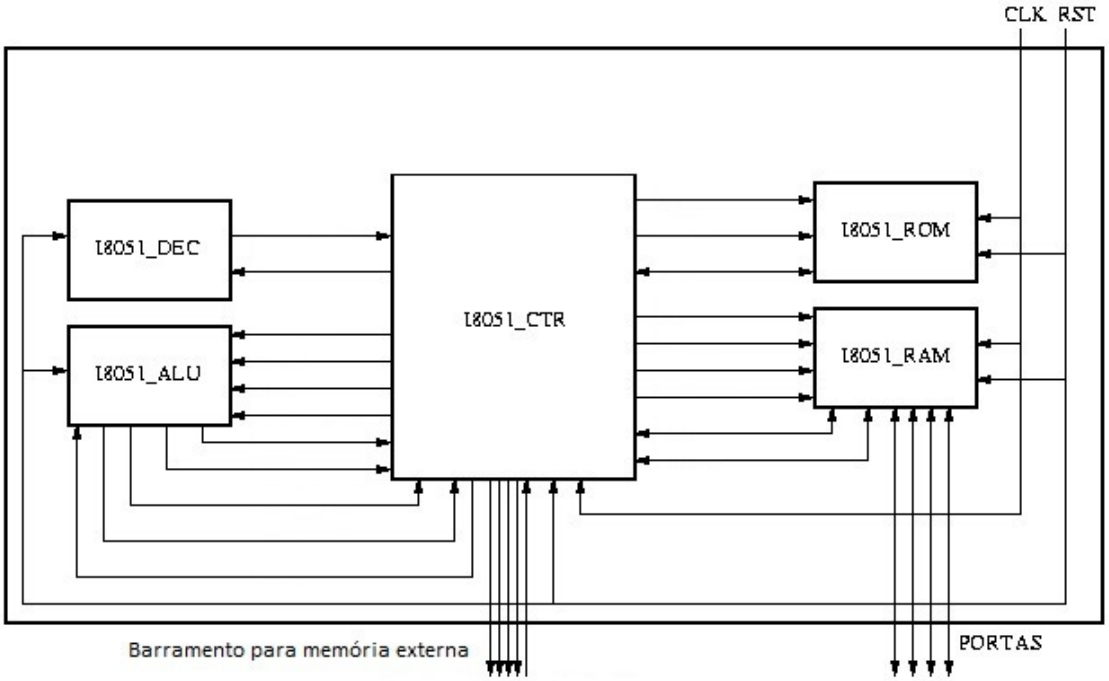


Figura 0-1: Arquitetura do microcontrolador 8051.
Fonte: [31].

Na Figura 4-1 é apresentada a organização dos blocos que compõem o 8051: a ULA (I8051_ALU), a RAM (I8051_RAM), a ROM (I8051_ROM), o *Core* (I8051_CTR) e o Decodificador de Instruções (I8051_DEC). São apresentados também os sinais de *clock* (CLK) e de *reset* (RST), as Portas de entrada ou saída e o barramento de endereçamento e dados para o mapeamento de memória externa, além das interconexões entre os blocos.

O código VHDL do microcontrolador 8051 utilizado pode ser encontrado no seguinte endereço de *web*: http://www.cs.ucr.edu/~dalton/i8051/i8051syn/, referente a um projeto desenvolvido pelo Departamento de Ciências da Computação da Universidade da Califórnia [31]. O projeto consiste da divisão do 8051 em blocos, descritos em VHDL, que podem ser utilizados na sintetização do fluxo de projeto ASIC do 8051. Os códigos VHDL utilizados para a criação dos blocos ULA e RAM podem ser encontrados no Apêndice A e no Apêndice B, respectivamente.

De posse dos códigos VHDL, antes da implementação do leiaute, uma etapa muito importante é a da sua validação. Para tal, utilizou-se uma placa FPGA didática. O *kit* didático disponibilizado pela FEI foi da Altera, modelo DE2-70, que possui um FPGA Cyclone II [32], simulação das entradas através de chaves mecânicas, e das saídas através de *LED*s. A comunicação com o computador se dá por meio de cabo USB e a sua alimentação é feita

através de uma fonte de alimentação externa. A Figura 4-2 ilustra o *kit* utilizado no processo de validação.

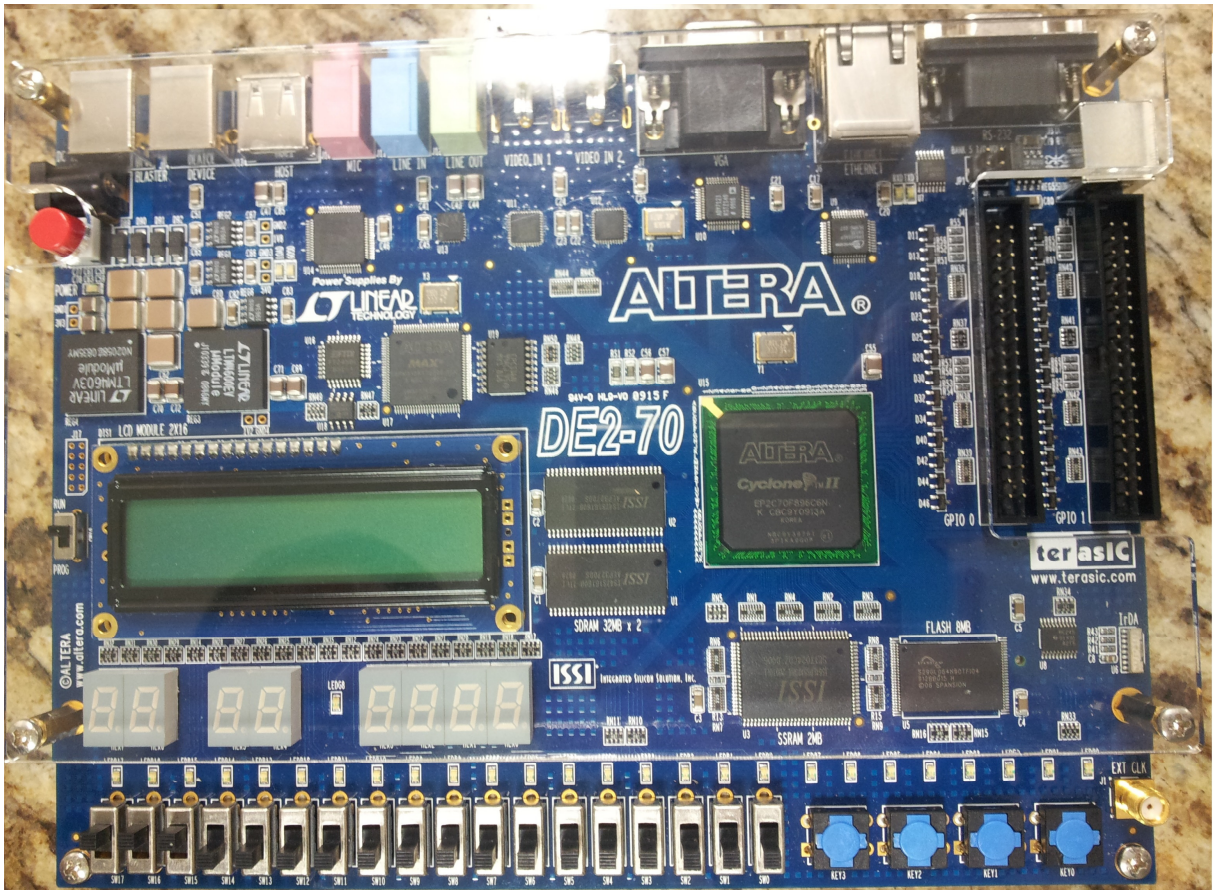


Figura 0-2: Placa didática com FPGA da Altera utilizada para a validação do VHDL utilizado no projeto do 8051.

Fonte: [32].

1.8.1 Criação de blocos VHDL no *software Quartus* II

Para a interação com o *kit* FPGA, é utilizado o *software Quartus* II da Altera. A versão utilizada foi 8.1 *Web Edition,* disponibilizada gratuitamente pelo fabricante em seu site [10].

O procedimento de utilização do *Quartus* II para a criação dos blocos VHDL estão descritos no Apêndice C.

Na Figura 4-3, é apresentado o esquemático da arquitetura do 8051 completa, gerada pelo projeto, nos passos descritos pelo tutorial do Apêndice C.

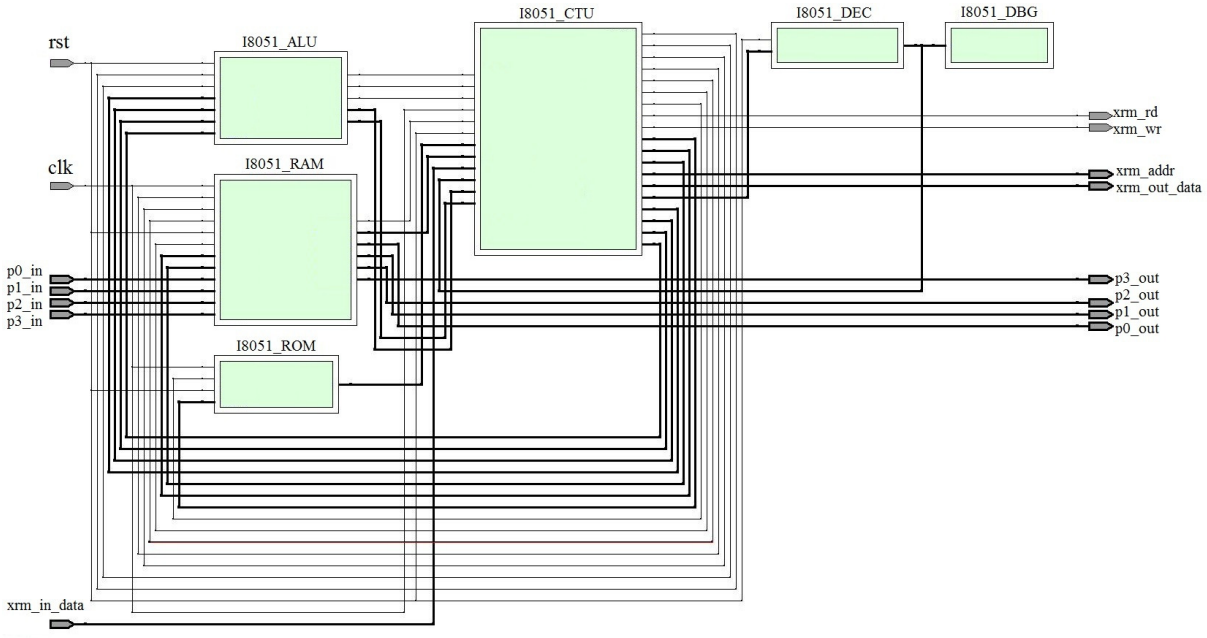


Figura 0-3: Esquemático completo do 8051 gerado no *Quartus* pela compilação do código VHDL.

1.8.2 Programas utilizados para teste de funcionamento do 8051 implementado com VHDL no FPGA

No caso deste projeto, está sendo utilizada uma memória ROM programada por máscara. Como o programa do usuário é armazenado na memória ROM, a cada programa é gerado um leiaute diferente para a memória ROM. Portanto, o leiaute a ser gerado, deve ser de acordo com o programa que desejamos executar no microcontrolador. Isto significa que o código VHDL da memória ROM, que inserimos no projeto do *Quartus*, deve estar de acordo com esta programação.

Para esta etapa, o Departamento de Ciências da Computação da Universidade da Califórnia disponibilizou junto ao código VHDL, um aplicativo para converter um código de programação em *Assembly* em um código de descrição VHDL, para ser inserido no projeto do leiaute do circuito integrado.

O procedimento para realizar o programa do 8051 consiste em, inicialmente, criar uma programação em *Assembly* com a rotina a ser executada pelo microcontrolador (programa do usuário). Este arquivo, que contém um programa em *Assembly*, deverá ser salvo com a extensão .asm. É necessário utilizar um aplicativo que converta este arquivo .asm em um arquivo .obj, e depois de .obj em arquivo .hex. Foi utilizado para este projeto, um aplicativo gratuito chamado Pequi [33]. Através do terminal de comando do *Linux*, ou do *prompt* de comando do *Windows*, o aplicativo conversor do Departamento de Ciências da Computação

da Universidade da Califórnia deve ser invocado. Através de linhas de comando próprias do aplicativo, a partir do arquivo .hex gerado anteriormente pelo compilador *Assembly*, é gerado um arquivo com descrição VHDL da ROM para utilização no projeto do microcontrolador 8051 [31].

Este código VHDL da ROM gerado pelo processo descrito acima, deve ser carregado no projeto do *Quartus* II, salvo e compilado, como descrito anteriormente para os outros blocos de código VHDL.

Para a validação parcial do funcionamento do microcontrolador 8051 a partir dos códigos VHDL, foram desenvolvidos cinco programas testes para o 8051 e implementados na estrutura FPGA. Abaixo, são apresentados os programas desenvolvidos em linguagem *Assembly,* chamados de Acender um *led,* Partida direta, Manipulação de um *byte,* Escrita e leitura da memória RAM e Subtração.

1.8.2.1 Acender um *led*

Inicialmente, foi desenvolvido um programa em *Assembly* bem simples, cujo objetivo principal foi simplesmente testar as funcionalidades básicas do projeto, acendendo um *led* da placa FPGA que foi ligado ao bit 0 da porta 0 (P0.0), que simula uma saída do microcontrolador (no microcontrolador 8051, para entrada = 0 a saída é habilitada e para entrada = 1, a saída é desabilitada), a partir de uma chave mecânica que existe na placa FPGA que foi ligada ao bit 0 da porta 1 (P1.0), simulando uma entrada:

```
$mod51                  ; Diretiva exclusiva do PEQUI
    ORG     0000        ; Inicia o programa no endereço 0
INICIO:
    SETB    P0.0        ; Atribui 1 no pino P0.0, desligando o LED
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
```

1.8.2.2 Partida direta

Neste outro programa em *Assembly,* é gerada uma simples condição lógica equivalente a uma partida direta de um motor. Três entradas foram simuladas pelas chaves mecânicas da placa FPGA: P1.0 porta de entrada, que simula um botão Desliga; P1.1 porta de entrada, que simula um Relê térmico; P1.2 porta de entrada, que simula um botão Liga. P0.0 é a porta de saída, que na placa, é simulada por um *led,* como se fosse o Motor. O estado da saída depende do estado das entradas. Por exemplo, se a entrada P1.2 for acionada, o *led* é acionado e se forem acionadas as entradas P1.0 ou P1.1, o *led* desliga.

```
$mod51                  ; Diretiva exclusiva do PEQUI
    ORG     0000        ; Inicia o programa no endereço 0
INICIO:
    SETB    P0.0        ; Atribui 1 no pino P0.0, desligando o LED
VOLTA:
    JB      P1.0, INICIO ; Testa se a porta P1.0 (botão Desliga) é igual a 1.
                         ;Se for, vai para INICIO
    JB      P1.1, INICIO ; Testa se a porta P1.1 (Relê térmico) é igual a 1. Se
                         ; for, vai para INICIO
    JNB     P1.2, LIGA   ; Testa se a porta P1.2 (botão Liga) é igual a 0. Se
                         ; for, vai para LIGA
    JMP     VOLTA        ; Pula para o endereço VOLTA
LIGA:
    CLR     P0.0         ; Atribui 0 no pino P0.0, ligando o LED
    JMP     VOLTA        ; Pula para o endereço VOLTA
END                      ; Finaliza o programa
```

1.8.2.3 Manipulação de um *byte*

Este programa em *Assembly*, diferentemente dos anteriores, que apenas manipulavam um *bit*, manipula um *byte*. Ao pressionar uma chave mecânica da placa FPGA, que simula a entrada da porta P1.1, todos os *bits* da porta de saída P0 são habilitados, e conectados a oito

*Led*s da placa FPGA. Pressionando outra chave mecânica da placa FPGA, que simula a entrada da porta P1.0, todos os *bits* da porta de saída P0 são desabilitados.

```
$mod51                    ; Diretiva exclusiva do PEQUI
    ORG     0000H         ; Inicia o programa no endereço 0
INICIO:
    MOV     P0,#0FFH      ; Move o valor 0xff em hexa para a PORTA P0
                          ; (Desliga o LED)
VOLTA:
    JB      P1.0, INICIO  ; Testa se a porta P1.0 (botão Desliga) é igual a 1.
                          ; Se for, vai para INICIO
    JNB     P1.1, LIGA    ; Testa se a porta P1.1 (botão Liga) é igual a 0. Se
                          ; for, vai para LIGA
    JMP     VOLTA         ; Pula para o endereço VOLTA
LIGA:
    MOV     P0,#0FEH      ; Move o valor 0xfe em hexa para a PORTA P0
                          ; (Ligando apenas um LED)
    JMP     VOLTA         ; Pula para o endereço VOLTA
END                       ; Finaliza o programa
```

1.8.2.4 Escrita e leitura da memória RAM

O objetivo deste outro programa em *Assembly* foi testar especificamente a funcionalidade de uma faixa de memória da RAM interna do 8051. São realizadas rotinas de escrita e de leitura nos registradores R0, R1, R2 e R3. Através destas rotinas nos registradores R1, R2 e R3, uma sub-rotina denominada TEMPO foi implementada para criar um atraso de 1 segundo. O valor escrito no registrador R0 é transferido para a porta P1, acionando um *led* da estrutura FPGA. Após a decorrência da sub-rotina de atraso, é movido outro valor para a porta P1 que apaga o *led.*

```
$mod51                      ; Diretiva exclusiva do PEQUI
    ORG     0000h           ; Inicia o programa no endereço 0

INICIO:
    MOV R0,#0FEH        ;Move o valor feh para o registrador R0
    MOV P1,R0           ;Move o valor armazenado no registrador R0 para a porta
                        ;P1 (11111110b)
    CALL TEMPO          ;Chama a sub-rotina TEMPO
    MOV P1,#0FFH        ;Move o valor ffh para a porta P1 (11111111b)
    CALL TEMPO          ; Chama a sub-rotina TEMPO
    MOV P1,R0           ;Move o valor armazenado no registrado R0 para a porta
                        ;P1 (11111110b)
    CALL TEMPO          ; Chama a sub-rotina TEMPO
    JMP INICIO          ;Volta para o INICIO

 TEMPO:                 ; Sub-rotina TEMPO
    MOV     R1,#0D5H    ;Move o valor d5h para o registrador R1
 VOLTA1:
    MOV     R2,#0FFH    ;Move o valor ffh para o registrador R2
 VOLTA2:
    MOV     R3,#0FFH    ;Move o valor ffh para o registrador R3
    DJNZ    R3,$        ;Decrementa o registrador R3 até ele chegar em 0
    DJNZ    R2,VOLTA1   ;Decrementa o registrador R2, se for 0 vai para
                        ;a linha de baixo, senão vai para VOLTA1
    DJNZ    R1,VOLTA2   ;Decrementa o registrador R1, se for 0 vai para a
                        ; linha de baixo, senão vai para VOLTA2
    RET                 ;Retorna
 END
```

1.8.2.5 Subtração

O objetivo deste outro programa em *Assembly* foi testar especificamente a funcionalidade do bloco ULA. Uma operação aritmética (subtração) é executada em A (acumulador) e em uma posição de memória, e o resultado desta operação é transferido para a

porta P1, que demonstra a funcionalidade do programa através do estado dos *leds* conectados a esta porta, que acendem ou apagam conforme o valor transferido de A para P1.

```
$mod51                  ; Diretiva exclusiva do PEQUI
    ORG     0000h       ; Inicia o programa no endereço 0
INICIO:
    MOV R0,#0FEH        ;Move o valor feh para o registrador R0
    MOV P1,R0           ;Move o valor armazenado no registrador R0 para a
                        ;porta P1 (11111110b)
    CALL TEMPO          ;Chama a sub-rotina TEMPO
    MOV A,#0FD          ;Move o valor fdh para o acumulador
    MOV P1,A            ;Move o valor armazenado no acumulador para a
                        ;porta P1 (11111101b)
    CALL TEMPO          ;Chama a sub-rotina TEMPO
    SUBB A,R0           ;Subtrai do valor do acumulador, o valor armazenado no
                        ;registrador R0. O resultado é armazenado no
                        ;acumulador
    MOV P1,A            ;Move o resultado da subtração, armazenado
                        ;no acumulador, para a porta P1 (00000001b)
    CALL TEMPO          ;Chama a sub-rotina TEMPO
    JMP INICIO          ;Volta para o início
TEMPO:                  ;Chama a sub-rotina TEMPO
    MOV     R1,#0D5H    ;Move o valor d5h para o registrador R1
VOLTA1:
    MOV     R2,#0FFH    ;Move o valor ffh para o registrador R2
VOLTA2:
    MOV     R3,#0FFH    ;Move o valor ffh para o registrador R3
    DJNZ    R3,$        ;Decrementa o registrador R3 até ele chegar em 0
    DJNZ    R2,VOLTA1   ;Decrementa o registrador R2, se for 0 vai para
                        ;a linha de baixo, senão vai para VOLTA1
    DJNZ    R1,VOLTA2   ;Decrementa o registrador R1, se for 0 vai para a
                        ; linha de baixo, senão vai para VOLTA2
    RET                 ;Retorna
END
```

1.8.3 Teste de funcionamento do 8051 com a placa FPGA

Cada um dos programas em *Assembly* acima foi testado individualmente. A primeira etapa do teste, ainda consiste na configuração do *software Quartus*. Como os programas simulam entradas e saídas nas portas do microcontrolador, é necessário atribuir os pinos da placa FPGA para as portas do microcontrolador. Para esta etapa, se faz necessário o uso do manual da placa Altera, para saber qual a pinagem utilizada na montagem física da placa didática de FPGA, destinada às chaves mecânicas e aos *Leds* [32].

Este projeto de circuito integrado do microcontrolador 8051 apresenta quatro portas de 8 *bits*, onde cada *bit* de cada uma das portas pode ser programado individualmente como entrada ou como saída [31]. Ao realizar o teste de cada programa, deve-se observar quais portas devem ser utilizadas e então atribuir pinos da estrutura FPGA. A atribuição de pinos deve obedecer à disponibilidade das chaves para as entradas e dos *Led*s para as saídas. Além disso, é necessário definir os pinos de *Clock* e *Reset* cada vez que se configura uma FPGA [34]. O *clock* é gerado pela própria placa didática, e o *reset* é simulado através de uma chave mecânica.

Todos os programas testados foram validados com sucesso. Os resultados demonstrados na placa didática FPGA foram exatamente como projetados. Houve a exata interação entre as portas de entradas e saídas, como foram programados em *Assembly*. Além de verificarem a funcionalidade do microcontrolador, verificaram de maneira específica os blocos RAM e ULA do projeto.

A validação dos códigos VHDL foi concluída com êxito. A próxima etapa do projeto, portanto, foi a geração automática do leiaute utilizando-se as ferramentas profissionais da *Mentor Graphics* de implementação de leiaute.

1.9 Geração automática de leiaute utilizando ferramentas profissionais de CAD

Após a validação do código VHDL do 8051 realizada em FPGA, seguindo o fluxo de projeto de CI ASIC descrito na Figura 2-12, o próximo passo é a geração automática de leiaute utilizando ferramentas profissionais.

Devido à complexidade do circuito do microcontrolador 8051, foram selecionados os blocos ULA e RAM para a geração do leiaute conforme o fluxo estudado.

A Figura 4-4 apresenta em quais momentos as ferramentas profissionais da *Mentor Graphics* foram utilizadas no fluxo de projeto de CI ASIC.

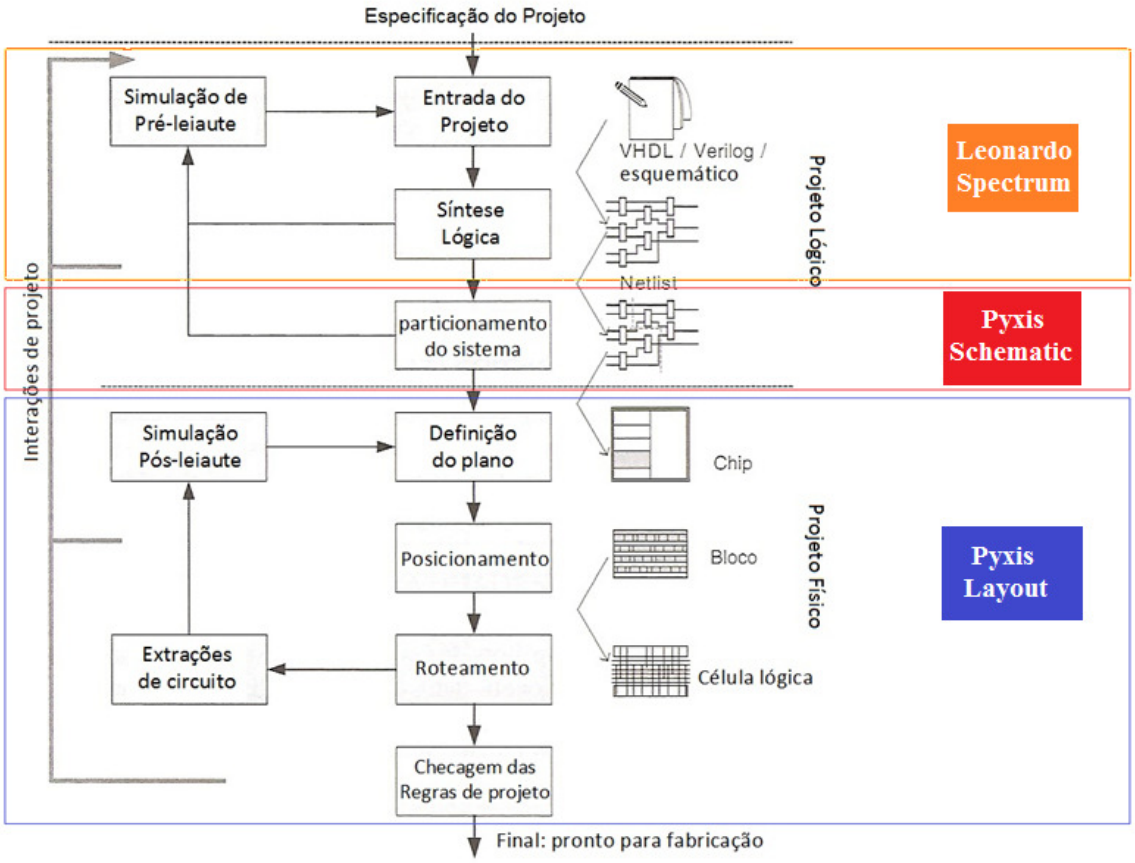


Figura 0-4: Fluxo de projeto para CIs ASIC utilizando ferramentas profissionais de leiautes de CIs da *Mentor Graphics*.

Adaptado de: [25]

1.9.1 Ferramentas profissionais de leiaute da *Mentor Graphics*

A *Mentor Graphics Corporation* criou, em 1985, o HEP com o objetivo de proporcionar às universidades as ferramentas necessárias para a capacitação de engenheiros leiautistas para a indústria eletrônica de CIs [9].

Na prática, as universidades parceiras do programa têm acesso exclusivo a um portal que garante um suporte *on-line* para dúvidas e problemas na utilização das ferramentas, treinamentos presenciais e *on-line*, informações e novidades sobre o uso das ferramentas e *downloads* de programas e pacotes para os produtos da *Mentor Graphics* [9].

A FEI faz parte de um grupo de mais de 1200 universidades parceiras do programa *Mentor Graphics* HEP, garantindo acesso aos recursos citados acima para realização de projetos como o deste trabalho.

O HEP traz uma série de *softwares*, que devem ser escolhidos de acordo com o segmento de utilização, como por exemplo: Projetos de CIs nanométricos, Projeto, verificação e teste, Placas de circuito impresso, Análise Mecânica, Cabeamento e Desenvolvimento de *softwares* embarcados [9].

As ferramentas utilizam uma plataforma Linux para operação. As licenças operam via servidor da *Mentor Graphics*, que se comunica com o servidor da universidade através da *internet* e *software* de gerenciamento de licenças. Portanto, para a utilização dos *softwares* é necessário que a estação de trabalho esteja em rede e o servidor de licenças configurado de acordo com os procedimentos do manual da ferramenta. É altamente recomendável o *download* e a instalação dos *patches*, atualizações e/ou correções, disponibilizados no portal [35].

Após a instalação e configuração dos *softwares* necessários a este trabalho, foi baixado no portal HEP, o *Kit* para Projeto ASIC (*ASIC Design Kit*, ADK). Este conjunto traz todos os arquivos necessários à criação de um projeto totalmente compatível com as tecnologias e os fabricantes de CIs mais utilizados, como AMI de 0.5µm e 1.2/1.5 µm, e TSMC 0.35µm, 0.25µm e 0.18µm disponíveis no MOSIS [36]. São bibliotecas, símbolos, regras de projetos, arquivos de mapeamento e atalhos de personalização das ferramentas, que são disponibilizados pela *Mentor Graphics* para facilitar o desenvolvimento de projetos para quem não está familiarizado com a metodologia tradicional da indústria na concepção de CIs [36].

O *kit* traz também um tutorial que demonstra como realizar o fluxo básico de procedimentos para as ferramentas de projeto trabalharem com os arquivos do ADK. A versão baixada do ADK foi a 3.1 de 2005, que contém todas as informações para utilização dos *softwares Leonardo Spectrum, Modelsim* (substituído no fluxo de projeto deste trabalho pelo *Quartus* da Altera), *ICStation e Calibre* [36].

Após a utilização deste conjunto ADK, foi detectado uma incompatibilidade, pois o *software ICStation* foi substituído pela *Mentor Graphics* pelo *Pyxis Layout* v10. Devido a esta

substituição, o ADK 3.1 tornou-se obsoleto, sendo substituído pelo Conjunto para Projeto Genérico (*Generic Design Kit*, GDK), também disponibilizado no portal HEP em abril de 2013 [37].

O GDK, assim como o ADK, traz um pacote com nove tutoriais para a as diversas aplicações das ferramentas *Mentor Graphics* como Captura de Esquemático e Automação de Leiaute [37]. Um nova ferramenta foi acrescentada para facilitar a navegação pelos diversos arquivos utilizados nos projetos, o *Project Navigator* [37].

Uma configuração importante deve ser realizada após a instalação dos *softwares:* a definição das variáveis de ambiente. No arquivo de sistema do Linux, são definidos os caminhos de diretórios onde estão instalados os *softwares.* Desta forma, as ferramentas são invocadas a partir do terminal do sistema operacional. Também são definidos: o caminho de diretório raiz da aplicação (MGC_HOME) e o diretório das bibliotecas para os aplicativos (MGC_GENLIB).

O processo de geração automática de leiaute do CI digital utiliza várias ferramentas para sua realização. A seguir é descrita a utilização de cada ferramenta e a sua correspondente etapa dentro do fluxo de geração de leiaute automático [37].

1.9.2 Sintetização dos arquivos VHDL dos blocos ULA e RAM do 8051 utilizando o *software Leonardo Spectrum*

O arquivo VHDL validado na estrutura FPGA será a entrada das informações do projeto de leiaute do circuito digital do 8051. Dentro do fluxo de projeto ASIC da *Mentor Graphics*, a ferramenta *Leonardo Spectrum* é a responsável pela síntese lógica, conforme foi descrito no capítulo 2, item 2.3.2.1 [38].

Existem duas plataformas para este *software*: o Modo de Interface Gráfica do Usuário (*Graphical User Interface,* GUI Mode) e o Modo de Linhas de Comando (*Command-Line Mode*) [13]. O pacote da *Mentor Graphics* HEP adquirido pela FEI possui apenas a licença para o segundo modo, sendo este utilizado na realização deste trabalho.

Um código VHDL pode ser totalmente simulado, mas nem sempre pode ser sintetizado, principalmente quando é utilizado o padrão IEEE VHDL 1076 [38], pois alguns arquivos VHDL não podem ser representados por um circuito digital, ou em alguns casos, não são representados de maneira precisa [38]. Um exemplo deste tipo de problema é o tempo de atraso do sinal de entrada para a saída [38].

Sintetizar um código VHDL nada mais é do que produzir registradores e combinações lógicas em nível RTL de modo automático, sem a intervenção do usuário [38]. Os algoritmos do estado-da-arte da síntese são capazes de otimizar os circuitos RTL levando-se em conta, especificamente, as características das tecnologias escolhidas [38].

Mas os algoritmos para otimização de desempenho elétrico dos circuitos digitais ainda não são tão robustos para aplicações gerais. Então, o resultado da síntese lógica depende da descrição VHDL utilizada [38]. O *Leonardo Spectrum* é capaz de sintetizar todos os níveis de abstração resultando em uma descrição *Netlist* do circuito digital, minimizada conforme a lógica necessária e a tecnologia escolhida [38].

A Figura 4-5 apresenta o diagrama de blocos da ferramenta *Leonardo Spectrum* dentro de um fluxo *Top-down* simplificado de um projeto de CI digital ASIC [38]:

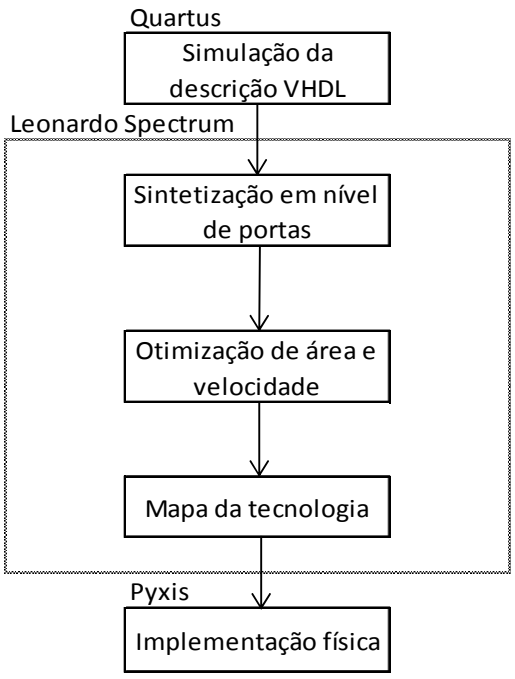


Figura 0-5: Diagrama de blocos da sintetização lógica da descrição VHDL realizados pelo *Leonardo Spectrum.* Fonte: [38].

Para a sintetização do VHDL, é necessário inicialmente escolher uma das bibliotecas referente à tecnologia de fabricação de CIs desejada, como por exemplo TSMC 0.35µm. Depois disso, é realizada a leitura do arquivo VHDL selecionado, no caso o mesmo arquivo utilizado como entrada no *software Quartus,* que foi utilizado para a validação do microcontrolador 8051.

Digitando o comando *optimize,* o *Leonardo Spectrum* realiza a otimização do resultado da sintetização. Esta otimização pode ser controlada pelo usuário através de parâmetros e argumentos do comando [13]. Pode-se selecionar o grau de otimização, acrescentar *buffers* às entradas e saídas, minimizar área (padrão), otimização de atrasos, atraso e área, entre outros [13]. Os resultados da otimização dependem das regras de projetos da tecnologia selecionada [13].

Após a otimização, pode-se solicitar um *report,* que apresenta as características do resultado obtido.

A Figura 4-6 mostra um exemplo do relatório gerado pelo comando "*report_area*" do circuito ULA do 8051.

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

Cell: I8051_ALU    View: BHV    Library: work

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

 Number of ports :                      50
 Number of nets :                      890
 Number of instances :                 860
 Number of references to this view :     0

Total accumulated area :
 Number of gates :                    1185
 Number of accumulated instances :     860
Info, Command 'report_area' finished successfully
```

Figura 0-6: Exemplo de um relatório gerado no *software Leonardo Spectrum* do circuito ULA do 8051.

Na Figura 4-6, o relatório da sintetização apresenta as características do projeto: *Number of ports,* quantos tipos de portas foram utilizadas (Inversora, E, OU, etc.), *Number of nets,* o número de ligações entre os dispositivos, *Number os instances,* o número de dispositivos criados a partir da descrição VHDL. Apresenta ainda *Number of gates,* o número de portas utilizadas.

Um *script* (conjunto de comandos) é apresentado na Figura 4-7 com o objetivo exemplificar a sequência de comandos do *Leonardo Spectrum* para a síntese de um projeto de leiaute. Finalizando a síntese lógica, os últimos comandos criam uma instância de um arquivo Verilog, ou seja, é criado um arquivo contendo o resultado do processo de otimização. Um comando define o nome do arquivo e qual diretório de destino.

```
1   load_library /usr/Mentor/Leonardo_spectrum_V2011/lib/tsmc035_typ
                ; carrega a biblioteca da tecnologia selecionada
2   read {/usr/Mentor/8051/ULA.vhd}
                ; carrega o arquivo VHDL de entrada do projeto
3   optimize
                ; comando para iniciar o processo de otimização do código Verilog
4   report_area
                ; comando para gerar o relatório da sintetização
5   set vhdl_write_use_packages {library ieee,adk; use ieee.std_logic_1164.all; use
    adk.adk_components.all;}
                ; carrega os pacotes VHDL
6   apply_rename_rules -ruleset VERILOG
                ; utiliza na sintetização as regras para geração de um arquivo Verilog
7   auto_write /usr/Mentor/8051/8051_ULA.v
                ; cria arquivo 8051_ULA.v no diretório especificado
```

Figura 0-7: *Script* de comandos utilizados no *Leonardo Spectrum* para a sintetização lógica.

A Figura 4-7 apresenta todos os comandos necessários e utilizados na sequência de sintetização de um código VHDL para o projeto do 8051 no *software Leonardo Spectrum*. Na linha 1, a tecnologia é selecionada, que neste caso foi a CMOS da TSMC de 0,35µm. Na linha 2, o arquivo VHDL que foi simulado no *Quartus* e que é a entrada de dados do fluxo, contendo toda a descrição da lógica do projeto do bloco ULA do 8051. Na linha 3, o processo de otimização padrão, conforme descrito anteriormente, é executado. Na linha 4, um comando é definido com o intuito de apresentar um relatório das características da sintetização, exatamente como apresentado na Figura 4-6. As linhas 5 e 6 definem o processo de criação de um arquivo Verilog com o resultado do processo de síntese lógica. A linha 7, nomeia o arquivo Verilog "8051_ULA.v" armazenando no diretório indicado.

O Apêndice D apresenta o código do arquivo Verilog do bloco ULA do microcontrolador 8051, gerado conforme o procedimento acima descrito.

1.9.3 Criação das células padrão e do esquemático dos blocos ULA e RAM do 8051 no *software Pyxis Schematic*

A criação das células padrão é o processo de geração da representação esquemática das partes que compõem o projeto digital [14]. Tal criação pode ser manual, posicionando as portas lógicas com as respectivas interligações entre elas, ou automática, através da importação de um arquivo Verilog, contendo o *Netlist* gerado no processo de sintetização [14].

O esquemático gerado é utilizado no processo de geração automática do leiaute do circuito integrado digital, sendo esta etapa portanto fundamental para a realização do circuito digital [14].

A ferramenta utilizada nesta etapa é o *Pyxis Schematic*, na versão 10.2 de 2012. Por muitos anos essa ferramenta chamou-se *Design Architect,* nome que ficou muito conhecido no segmento. A invocação da ferramenta permaneceu alusiva ao antigo nome, pois no terminal do Linux deve-se digitar "DA_IC" [14].

Como este projeto tem por objetivo a simplificação e automação do processo de geração de leiaute, continuando a seguir o fluxo de projeto proposto na Figura 4.4, foi descartada a opção da geração manual do esquemático, por ser um projeto extremamente complicado e com muitos transistores. Selecionou-se, portanto, a geração automática do esquemático, através do processo de importação do arquivo verilog, utilizando o *Pyxis Schematic*.

Para esta importação, além do arquivo Verilog gerado no processo de sintetização, é necessário o arquivo de mapeamento ".v_tmf". Esse arquivo associa módulos e as portas primitivas da biblioteca da tecnologia selecionada ao esquemático, que é gerado a partir da descrição do *Netlist* da sintetização [14]. Este arquivo obrigatoriamente deve ser gerado antes do processo de importação do arquivo Verilog [14].

A criação do arquivo de mapeamento não é simples, pois há uma sintaxe própria da ferramenta que exige a declaração de todos os símbolos utilizados do projeto de leiaute (portas lógicas), associando a uma biblioteca de símbolos correspondente (no caso do GDK esta biblioteca é a GDK *Gates*). Assim, a própria ferramenta *Pyxis Schematic* gera um modelo que será utilizado como ponto de partida. No momento em que se realiza a importação do arquivo Verilog, pode-se optar pela geração deste modelo, sendo editado de acordo com as características do projeto, para ser utilizado na importação.

Um exemplo do modelo de arquivo .v_tmf de mapeamento da ULA do 8051 é apresentado na Figura 4-8.

```
1   // Template file for mapping primitive gates and modules to EDDM components.
2   // v10.3_1.0     Wed Oct 30 08:59:59 PDT 2013
3  // Each line in the mapping file specifies mapping information for a primitive, a module,
4   // a net connection to a pin, a symbol connection to a pin, or a module library.
5   // Primitive symbol definitions needed by the schematic generator.
6   //
7   p portin symbol_path/portin X
8   p portout symbol_path/portout X
9   p portbi symbol_path/portbi X
10  p ripper symbol_path/rip/1X1 bundle wire
11  p pagein symbol_path/offpag.in IN
12  p pageout symbol_path/offpag.out OUT
13  p netcon symbol_path/netcon IN OUT
14  p ground symbol_path/vss X
15  p power symbol_path/vdd X
16  //
17 m mux21_ni $GDKGATES/mux21 Y A0 A1 S0
18 m nor04 $GDKGATES/nor04 Y A0 A1 A2 A3
19 m or03 $GDKGATES/or03 Y A0 A1 A2
20 m xnor2 $GDKGATES/xnor2 Y A0 A1
21 m nor02_2x $GDKGATES/nor02_2x Y A0 A1
22 m aoi22 $GDKGATES/aoi22 Y A0 A1 B0 B1
23 m oai21 $GDKGATES/oai21 Y A0 A1 B0
24 m nand02 $GDKGATES/nand02 Y A0 A1
25 m nand03 $GDKGATES/nand03 Y A0 A1 A2
26 m inv02 $GDKGATES/inv02 Y A
27 m oai32 $GDKGATES/oai32 Y A0 A1 A2 B0 B1
28 m nand04 $GDKGATES/nand04 Y A0 A1 A2 A3
29 m xor2 $GDKGATES/xor2 Y A0 A1
30 m aoi21 $GDKGATES/aoi21 Y A0 A1 B0
31 m inv01 $GDKGATES/inv01 Y A
32 m oai22 $GDKGATES/oai22 Y A0 A1 B0 B1
33 m oai222 $GDKGATES/oai222 Y A0 A1 B0 B1 C0 C1
33 m aoi332 $GDKGATES/aoi332 Y A0 A1 A2 B0 B1 B2 C0 C1
34 m nor02ii $GDKGATES/nor02ii Y A0 A1
35 m oai221 $GDKGATES/oai221 Y A0 A1 B0 B1 C0
36 m aoi32 $GDKGATES/aoi32 Y A0 A1 A2 B0 B1
37 m nand02_2x $GDKGATES/nand02_2x Y A0 A1
38 m aoi322 $GDKGATES/aoi322 Y A0 A1 A2 B0 B1 C0 C1
39 m nor03_2x $GDKGATES/nor03_2x Y A0 A1 A2
40 m ao221 $GDKGATES/ao221 Y A0 A1 B0 B1 C0
41 m oai322 $GDKGATES/oai322 Y A0 A1 A2 B0 B1 C0 C1
42 m aoi222 $GDKGATES/aoi222 Y A0 A1 B0 B1 C0 C1
43 m or02 $GDKGATES/or02 Y A0 A1
44 m and02 $GDKGATES/and02 Y A0 A1
45 m buf02 $GDKGATES/buf02 Y A
46 m buf04 $GDKGATES/buf04 Y A
47 m mux21 $GDKGATES/mux21 Y A0 A1 S0
48 m and04 $GDKGATES/and04 Y A0 A1 A2 A3
49 m ao22 $GDKGATES/ao22 Y A0 A1 B0 B1
50 m or04 $GDKGATES/or04 Y A0 A1 A2 A3
51 m and03 $GDKGATES/and03 Y A0 A1 A2
```

Figura 0-8: Modelo de arquivo de mapeamento gerado pelo *Pyxis Schematic* do projeto de um Somador.

A Figura 4-8 mostra nas linhas de 1 a 5, um cabeçalho gerado pelo *software Pyxis Schematic,* contendo a sua versão e a data de geração do arquivo. As linhas de 7 a 15 apresentam linhas de comando que se iniciam por "p", especificando uma parte genérica ou primitiva do Verilog, partes comuns a todos os projetos como tensão de alimentação e portas de entrada e saída. Nas linhas de 17 a 51, são apresentadas linhas de comando iniciadas por "m", que significam um módulo Verilog específico ao projeto, como por exemplo a linha 19 com uma porta OU com três entradas.

De forma mais explicativa, a edição do arquivo de mapeamento deve-se realizar linha a linha. Para partes genéricas ou primitivas do Verilog, nas linhas que se iniciam por "p", deve-se utilizar a sintaxe [14]: `<type> <primitive> <symbol_path> <pin_list>`

Para módulos Verilog, nas linhas que se iniciam por "m", deve-se utilizar a sintaxe [14]: `<type> <module_name> <symbol_path> <pin_list>`

O primeiro passo da edição consiste em substituir as linhas que apresentem a ocorrência "symbol_path", atribuídas automaticamente pelo *software Pyxis Schematic,* pelo caminho correto da localização das bibliotecas da tecnologia selecionada [14].

O segundo passo é confirmar se o nome dos símbolos do modelo de arquivo de mapeamento está de acordo com os nomes das células da biblioteca [14]. Por exemplo: um módulo pode ter sido declarado como NOR2X1, mas na biblioteca é chamado NOR2 [14]. Isso deve ser ajustado para que eles apresentem o mesmo nome.

Nos Apêndices E e F são apresentados os arquivos de mapeamento dos blocos ULA e RAM do microcontrolador 8051, gerados e editados conforme o procedimento descrito anteriormente.

O processo de importação do Verilog é concluído quando, além de selecionar o arquivo Verilog, é selecionado o arquivo de mapeamento editado como demonstrado acima e escolhe-se um caminho de diretório (geralmente "*My Library"* dentro do seu projeto de trabalho) para o destino do esquemático.

Com o esquemático implementado, será possível visualizá-lo no *Project Navigator* um objeto do tipo *Schematic*. Com um duplo clique neste objeto, será aberto o esquemático do circuito desejado no *Pyxis Schematic.*

Pode-se editar o esquemático caso o leiautista veja necessidade. Deve-se observar se não existe nenhuma sobreposição dos objetos presentes no esquemático. Deve-se ainda realizar uma edição referente à tensão de alimentação de cada objeto (partes do circuito como portas, etc.), VDD e VSS. Implicitamente, cada objeto possui tensão de alimentação VDD e VSS. No esquemático, não existem conexões entre estas alimentações. É, portanto, indicado

pela *Mentor Graphics*, criar duas portas bidirecionais, conectando uma a VDD e outra a VSS, para interligar todos os pontos de alimentação dos objetos.

Ao término das edições, deve-se checar o esquemático através do clique no ícone *Check and Save*. É mostrada uma caixa de mensagens que exibirá erros, ou então a condição necessária para avançar até a geração do leiaute que significa o final do circuito esquemático checado, nenhum erro encontrado e salvo com sucesso.

Nas Figuras 4-9 e 4-10 são mostrados os esquemáticos dos blocos ULA e RAM do microcontrolador 8051, gerados conforme o procedimento descrito acima.

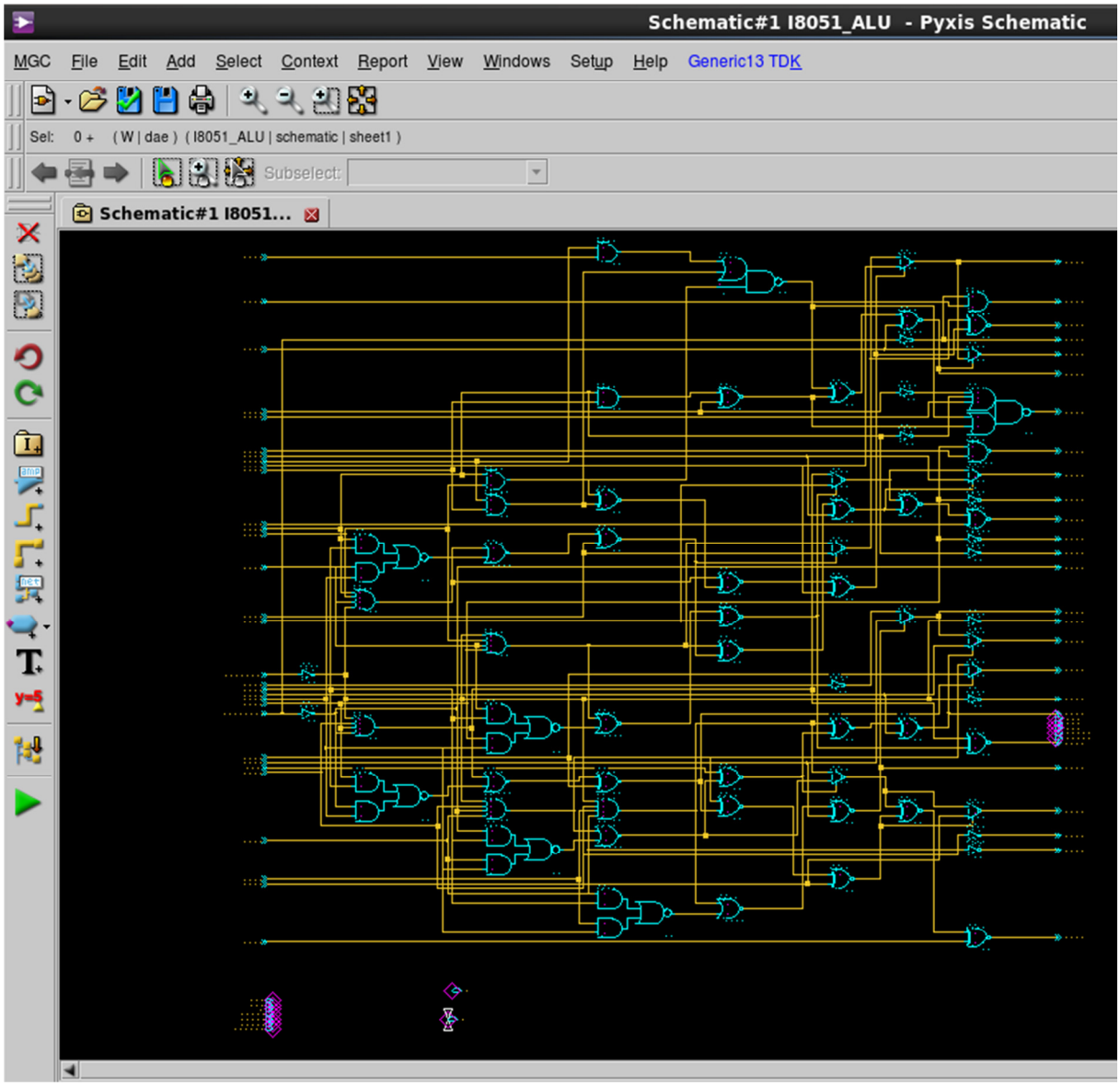


Figura 0-9: Esquemático do bloco ULA do microcontrolador 8051.

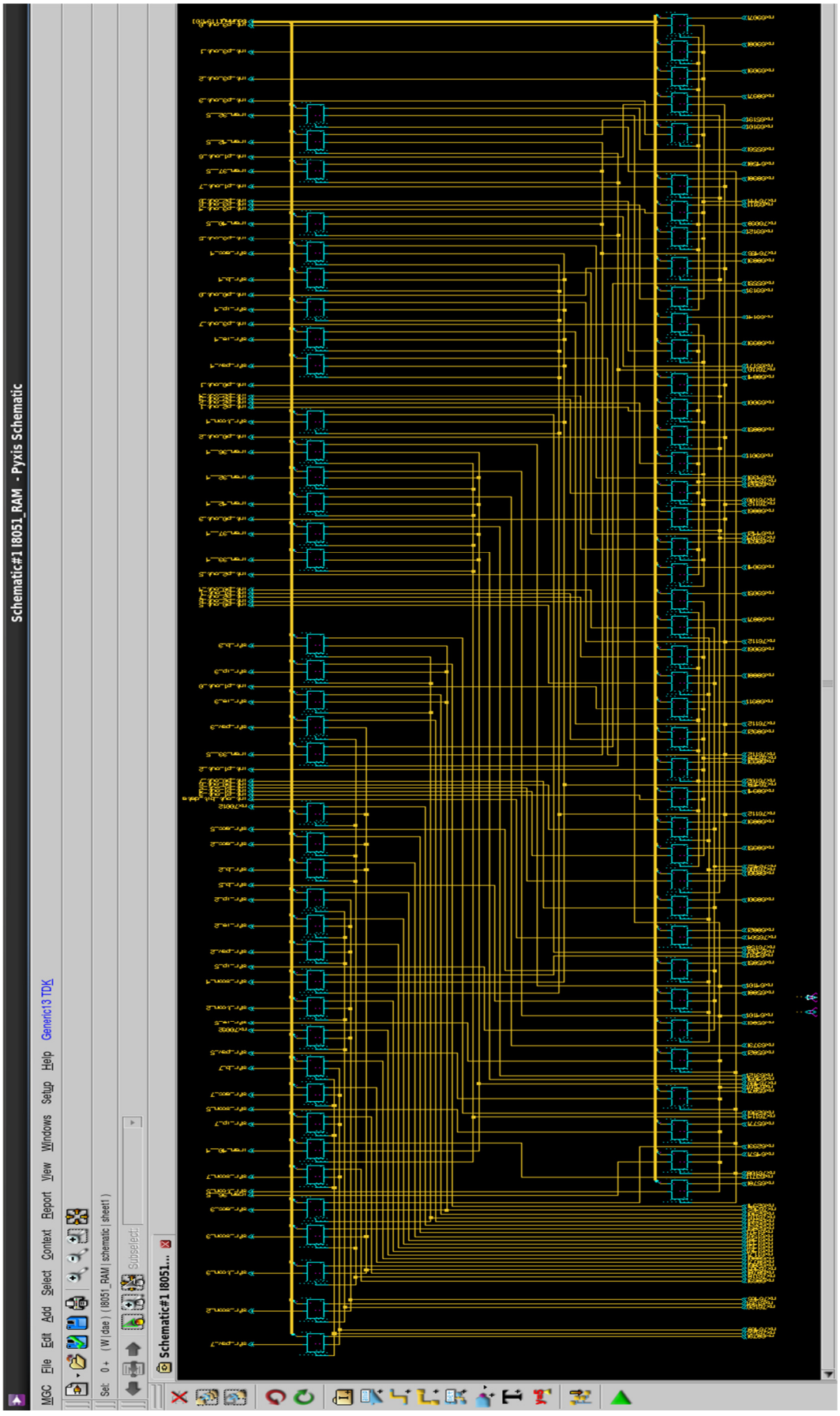


Figura 0-10: Esquemático do bloco RAM do microcontrolador 8051.

1.9.4 Geração automática do leiaute dos blocos ULA e RAM do 8051 utilizando o *software Pyxis Layout*

De acordo com a hierarquia do projeto de leiaute do circuito integrado, pode-se gerar o leiaute de cada bloco separadamente [15].

A geração automática utiliza de vários passos até a obtenção do leiaute final. Na Figura 4-11 é apresentado um diagrama de blocos dos passos percorridos no *software Pyxis Layout* para a realização de um leiaute automático:

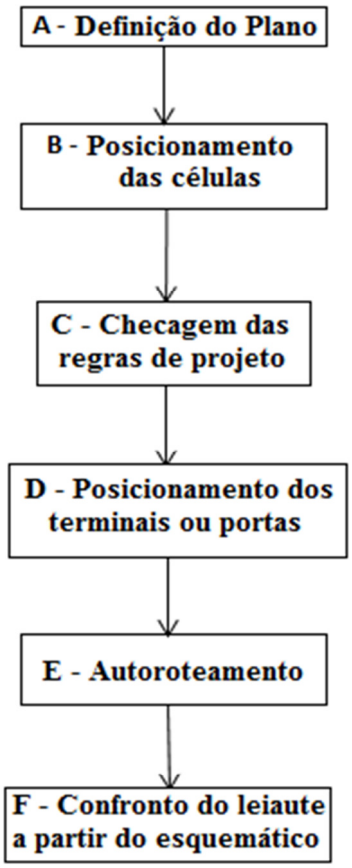


Figura 0-11: Diagrama de blocos da geração de leiaute automático de um CI digital no *software Pyxis Layout.* Fonte: [15].

A seguir será detalhado cada processo para a geração automática do leiaute dos blocos ULA e ROM do 8051, conforme os passos apresentados no diagrama de blocos.

Para geração do leiaute, deve-se criar um novo objeto do tipo *Layout.* Assim que este objeto é criado, com um duplo clique sobre ele, será invocada automaticamente a ferramenta *Pyxis Layout.* Assim como o *Pyxis Schematic*, a invocação da ferramenta diretamente pelo terminal do Linux é feita através de um comando alusivo ao antigo nome, “IC”.

O *Pyxis Layout* permite a realização de projetos de leiautes completos de CI, inclusive projetos com hierarquia de células e blocos padrão, podendo utilizar esquemáticos ou *Netlist* na criação dos leiautes para acelerar a criação física [15].

A ferramenta traz ainda algumas interfaces importantes para a geração de um leiaute confiável. *Calibre nmDRC* e *nmLVS* são ferramentas computacionais para verificação física e integridade elétrica dos projetos [15].

A geração do leiaute se dará utilizando alguns processos automáticos da ferramenta. A *Mentor Graphics* chama de Leiaute Guiado pelo Esquemático (*Schematic Driven Layout,* SDL), o processo de geração automática do leiaute a partir do esquemático como entrada de dados [15].

A primeira etapa da efetiva criação do leiaute é o processo de Definição do Plano (**A** - *Floorplanner*) que consiste no processo de arranjo dos diversos blocos do projeto (os subcircuitos) e o detalhamento do leiaute [15]. Neste processo, as informações das interconexões advindas do esquemático são lidas, gerando uma base de dados [15]. Depois disso, deve-se posicionar os blocos maiores do projeto no leiaute [15]. Além disso, pode-se particionar o plano e estimar a área necessária do leiaute final do circuito [15].

A etapa seguinte é a de posicionamento (**B** - *Placement*) das células (pequenos circuitos formados por transistores, como por exemplo um *flip-flop*). De acordo com a tecnologia selecionada, o *Pyxis Layout* utilizará o banco de dados gerado pelo esquemático, e posicionará as células da biblioteca da tecnologia nos blocos previamente posicionados. Posicionadas as células, antes de prosseguir, deve-se realizar a Checagem das Regras de Projeto (**C** - DRC) pelo comando *ICRules*, de acordo com as restrições impostas pela tecnologia selecionada. No próximo passo serão posicionadas as portas (**D** – Terminais ou Portas).

A próxima etapa consiste no Autorroteamento (**E** – *Autorouter*) das ligações entre as portas das células. De acordo com o *Netlist* armazenado no banco de dados, a ferramenta demonstra quais as ligações entre as células obedecem à lógica do esquemático. O roteamento pode ser realizado manualmente, mas para otimização de tempo na geração do leiaute, a ferramenta oferece um roteamento automático das ligações bastante confiável.

Finalizando, o Confronto do Leiaute a partir do Esquemático (**F** – LVS) verifica se o leiaute gerado está de acordo a lógica do esquemático que foi utilizado como fonte para a geração do leiaute.

Após a geração automática do leiaute, o mesmo pode ser editado de acordo com as necessidades do leiautista. Pode-se, por exemplo, adicionar textos que facilitem a

compreensão e entendimento do leiaute. Ou ainda a utilização de técnicas avançadas de leiaute, como a utilização de dispositivos *dummy*.

Os leiautes dos blocos ULA e RAM do microcontrolador 8051 foram gerados conforme procedimentos descritos acima. Os resultados dos leiautes gerados encontram-se nos Apêndices G e H.

O bloco de leiaute da RAM do microcontrolador 8051 é composto pela interligação de diversos blocos de *flip-flops*. A Figura 4-12 apresenta com maiores detalhes um destes *flip-flops*.

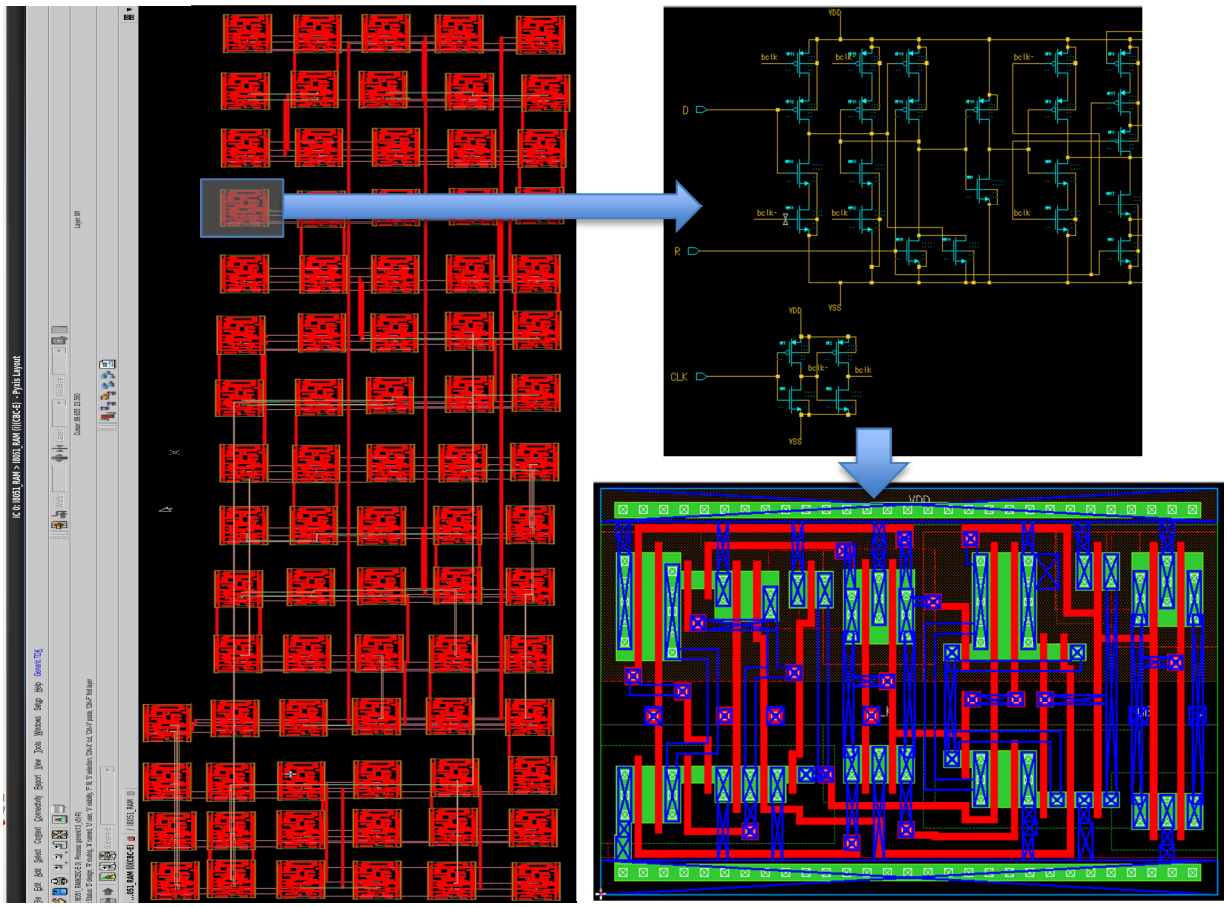


Figura 0-12: Detalhe de um *flip-flop* do bloco RAM do leiaute do microcontrolador 8051.

Na Figura 4-12 pode-se verificar o esquemático equivalente do bloco *flip-flop* do leiaute do microcontrolador 8051 e o seu respectivo leiaute gerado automaticamente pelo *software Pyxis Layout*.

A Figura 4-13 ilustra o detalhe de duas portas inversoras existentes no leiaute do *flip-flop*.

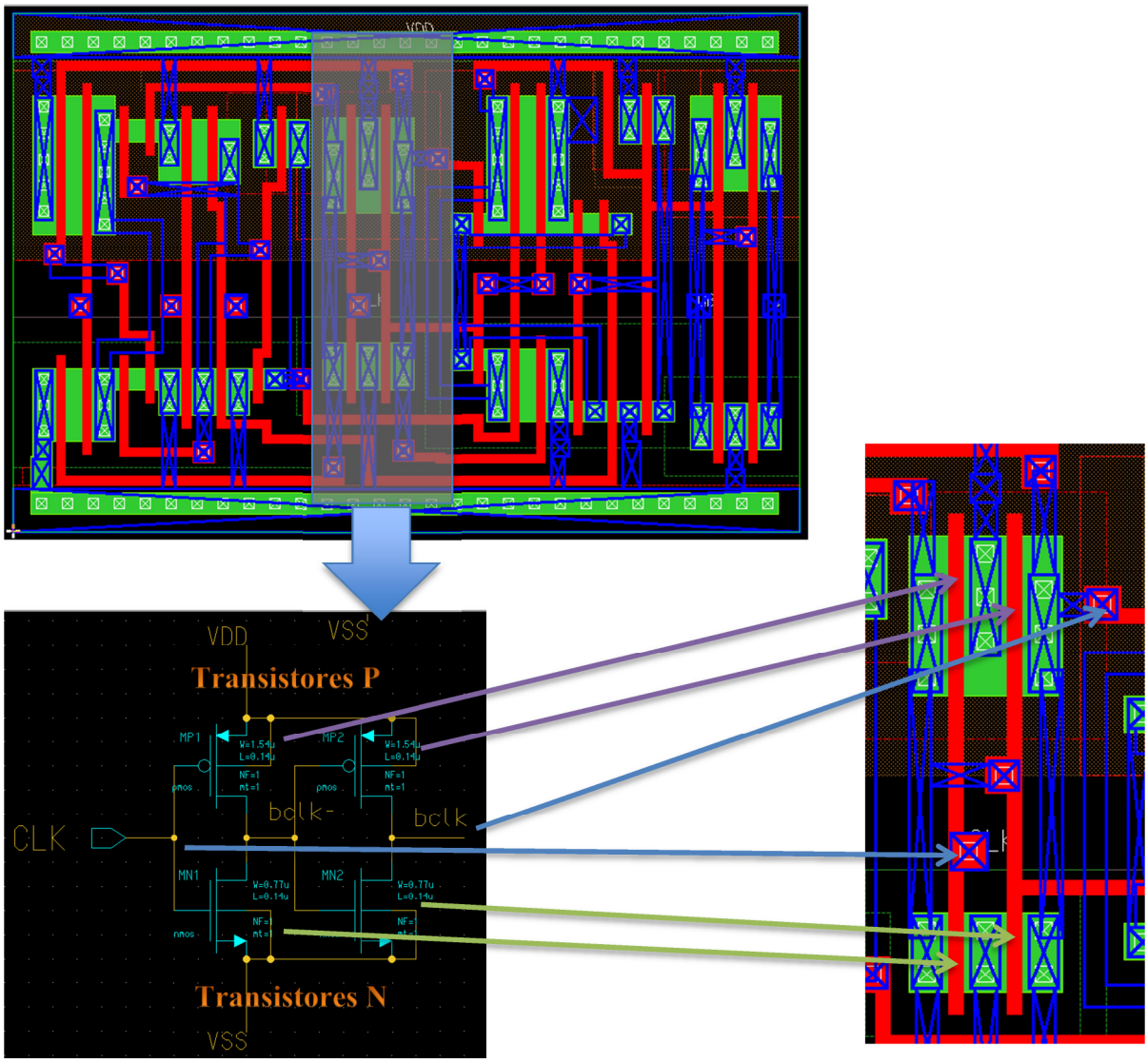


Figura 0-13: Detalhe de duas portas inversores do leiaute de um *flip-flop* do bloco RAM.

Na Figura 4-13 pode-se verificar que no leiaute do *flip-flop* existem duas portas inversores gerando um atraso no sinal de *clock*. O esquemático deste atraso através das duas portas inversoras é detalhado e ao lado o seu respectivo leiaute.

O Apêndice I traz um tutorial de utilização do *software Pyxis Layout* para a geração automática de leiaute.

Depois de gerado o leiaute, deve-se realizar a simulação lógica do mesmo no *software Pyxis Layout,* para garantir o seu funcionamento final, conforme foi realizado no capítulo 3.

Embora o leiaute já esteja pronto, é necessário exportar o projeto para o formato de arquivos GDSII, conforme explicado no item 2.3.2.3 do capítulo 2. O comando *Write GDSII* será responsável pela criação dos arquivos de máscaras, que poderão ser abertos por outros leiautistas ou enviado para os fabricantes de CIs para a sua fabricação.

CONCLUSÃO

Este trabalho estudou o fluxo de projetos de geração automática de leiautes de circuitos integrados digitais.

Foi implementado um fluxo de desenvolvimento de projetos de CIs digitais não complexos com ferramentas em versão demonstração: concepção do projeto, criação e edição, compilação e simulação de um código VHDL, criação e simulação de um esquemático, criação de um arquivo Verilog, geração, edição e simulação de um leiaute e a geração dos arquivos para manufatura, que possibilita qualquer instituição de ensino ou um "hobbista" desenvolver o seu projeto de CI de forma semiautomática, que é muito melhor que a implementação de um leiaute manual.

A novidade apresentada foi a utilização da linguagem Verilog para automação do processo de concepção do leiaute do CI. Apesar do fluxo ter se demonstrado funcional, pois atingiu o objetivo final de criação de um leiaute de forma semiautomática, com posterior simulação de funcionamento, o *Microwind II* apresentou limitações de simulação para implementação de circuitos integrados digitais com maior número de transistores pela sua versão demonstração. Além disso, os seus recursos para a edição do leiaute e integração com outras ferramentas de *softwares* para fabricação de CIs são bastante limitados (não possui exportação de dados, não importa VHDL, etc.). A simulação do funcionamento do leiaute também apresenta restrições em relação a parâmetros de configuração (não considera todos os efeitos de capacitâncias, resistências, etc.).

O microcontrolador 8051 foi implementado na placa didática de FPGA, validando o funcionamento do código VHDL.

Iniciou-se, portanto, pelo processo de validação dos códigos VHDL dos blocos ULA e RAM do 8051 em aplicações práticas disponíveis na literatura através da placa didática FPGA. O *software Quartus II* demonstrou-se bastante versátil na fase inicial do projeto, possibilitando a edição do código VHDL e a pré-implementação do projeto em FPGA, no qual foi possível verificar o funcionamento experimental do 8051. A utilização desta ferramenta consagrada para o uso de FPGA, no caso o *Quartus II* da Altera, para a edição e simulação do código VHDL, facilita a utilização do fluxo de projetos de leiautes de CIs, pelos projetistas iniciantes.

O fluxo de projetos de CI com ferramentas profissionais foi seguido utilizando-se as ferramentas de CAD da *Mentor Graphics*. Foi desenvolvido o projeto de leiaute dos blocos RAM e ULA do microcontrolador 8051 através da utilização de ferramentas profissionais na

concepção do leiaute de circuitos integrados digitais de maior porte. O código RTL gerado pelo *software Quartus II* foi sintetizado pelo *software Leonardo Spectrum* gerando um arquivo Verilog. Este arquivo Verilog foi importado pelo *software Pyxis Schematic* que gerou o arquivo *Netlist* dos blocos ULA e RAM do microtrolador 8051. O *software Pyxis Layout* gerou automáticamente os leiaute dos blocos de RAM e ULA do 8051. Foi percorrido o fluxo de projeto da geração do leiaute automático com a utilização dos *softwares* e tutoriais disponibilizados pelo programa de ensino superior da *Mentor Graphics.*

Para dar continuidade a este trabalho, sugiro como trabalhos futuros:

a) A fabricação do CI PWM no Mosis;
b) A possibilidade de utilizar o VHDL na geração automática de leiaute no *software Microwind*, para evitar todo o trabalho manual do desenvolvimento do fluxo de projeto proposto;
c) A simulação do leiaute do 8051 utilizando-se as ferramentas da *Mentor Graphics*;
d) O projeto de um CI 8051 com diversos periféricos (por exemplo: RF) e a utilização de transistores de geometria não convencional (diamante, octo e *wave*);
e) A publicação de um artigo com o conteúdo do Capítulo III;
f) A publicação de um livro ensinando a projetar CIs digitais não complexos.

REFERÊNCIAS

1. BASS, M. J.; CHRISTENSEN, C. M. The Future of the Microprocessor Business. **IEEE Spectrum**, 2002. p. 34-39.

2. ITOH, K. A Historical Review of Low-Power, Low-Voltage Digital MOS Circuits Development. **IEEE Solid-State Circuits Magazine**, 2013. p. 27-39.

3. MARTIN, B. Automation Comes to Analog. **IEEE Spectrum**, 2001. p. 70-75.

4. MARTINS, R.; LOURENÇO, N.; HORTA, N. LAYGEN II – Automatic Analog ICs Layout Generator based on a Template Approach. **GECCO'12**, 2012.

5. RUPP, K.; SELBERHERR, S. The Economic Limit to Moore's Law. **Proceedings of the IEEE**, Wien, Austria, mar 2010. p. 351 - 353.

6. HELLEMANS, A. Ring Around the Nanowire. **IEEE Spectrum**, 2013. p. 14-15.

7. DAMIANO, R.; REEVES, D. S. Logic synthesis for ASICs. **IEEE Spectrum**, 1991. p. 26, 32-33 e 62.

8. INC, C. D. S. Disponível em: <http://www.cadence.com/us/pages/default.aspx>. Acesso em: 5 junho 2014.

9. CORPORATION, M. G. Site da Mentor Graphics. **High Education Program**. Disponível em: <http://mentor.com/company/higher_ed/>. Acesso em: 5 Junho 2014.

10. ALTERA. **Download Center.** Disponível em: <https://www.altera.com/download/sw/dnl-sw-index.jsp>. Acesso em: Junho 2014.

11. CONCEPTS, N. W. **Circuit Wizard.** 2014. Disponível em: <http://www.new-wave-concepts.com/>. Acesso em: 5 Junho 2014.

12. SICARD, E. **Microwind.** Disponível em: <www.microwind.org>. Acesso em: 5 Junho 2014.

13. GRAPHICS, M. **LeonardoSpectrum Reference Manual**. Oregon - Estados Unidos: [s.n.], 2012.

14. GRAPHICS, M. **Pyxis Schematic User's Manual**. Oregon - Estados Unidos: [s.n.], 2012.

15. GRAPHICS, M. **PyxisTM Layout User's Manual**. Oregon - Estados Unidos: [s.n.], 2013.

16. GIMENEZ, S. P. **Microcontroladores 8051:** teoria e prática. 1. ed. São Paulo: Érica, 2010.

17. MENEZES, M. P.; SATO, L. M.; MIDORIKAWA, E. T. **Tutorial para Criar e Simular Circuitos Digitais no Altera Quartus II - versão 9.1**. [S.l.]: Escola Politécnica da USP, 2011. (Apostila)

18. ASHENDEN, P. J. **The VHDL Cookbook**. 1. ed. South Australia: Dept. Computer Science University of Adelaide, 1990.

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

20. VAHID, F. **Digital Design with RTL Design, VHDL, and Verilog**. 2. ed. Califórnia: Wiley, 2011.

21. COSTA, C. D. **Projetos de Circuitos Digitais com FPGA**. 1. ed. São Paulo: Érica, 2009.

22. OLIVEIRA, A. S. D.; ANDRADE, F. S. D. **Sistemas Embarcados:** hardware e firmware na prática. 1. ed. São Paulo: Érica, 2006.

23. ZELENOVSKY, R.; MENDONÇA, A. **Microcontroladores:** Programação e Projeto com a Família 8051. Rio de Janeiro: MZ Editora, 2005.

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

25. DESCHAMPS, J.-P. **Synthesis of arithmetic circuits**. New Jersey: Wiley, 2006.

26. 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, Campinas.

27. STAN, M. et al. Teaching Top-Down ASICISoC Design vs Bottom-Up Custom VLSI. **IEEE International Conference on Microelectronic Systems Education**, 2007.

28. IC Station User's Manual. [S.l.]: Mentor Graphics Corporation, 2003. Software Version v8.9_9.

29. HIRSCH, M.; SIEWIOREK, D. P. The Effect of Placement of Automatically Extracted Structure. **IEEE Transactions on Computer-Aided Design**, Setembro 1992.

30. OPENCORES, 1999-2014. Disponível em: <http://opencores.org/>. Acesso em: 5 junho 2014.

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

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

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

34. LUZ, H. B.; FERREIRA, P. H. W. **QUARTUS II para Projeto de CPLD/FPGA**. São Bernardo do Campo: Centro Universitário da FEI, 2009. (Apostila)

35. GRAPHICS, M. **Pyxis Installation Instructions**. v10.2. ed. Oregon - Estados Unidos: [s.n.], 2012.

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

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

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

39. MUSSOLINI, T. P. **Desenvolvimento de um microcontrolador de 8bits em VHDL baseado no conjunto de instruções do 8051 com comunicação serial I²C e criptografia AES128**. Itajubá: Universidade Federal de Itajubá, 2011. Dissertação de Mestrado.

40. RODRIGUES, J. N. **A manual on ASIC Front to Back End Design Flow**. IEEE International Conference on Microelectronic System Education. [S.l.]: [s.n.]. 2005.

APÊNDICE A – CÓDIGO VHDL DO BLOCO “ULA” DO 8051

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use WORK.I8051_LIB.all;

entity I8051_ALU is
    port(rst     : in  STD_LOGIC;
         op_code : in  UNSIGNED (3 downto 0);
         src_1   : in  UNSIGNED (7 downto 0);
         src_2   : in  UNSIGNED (7 downto 0);
         src_3   : in  UNSIGNED (7 downto 0);
         src_cy  : in  STD_LOGIC;
         src_ac  : in  STD_LOGIC;
         des_1   : out UNSIGNED (7 downto 0);
         des_2   : out UNSIGNED (7 downto 0);
         des_cy  : out STD_LOGIC;
         des_ac  : out STD_LOGIC;
         des_ov  : out STD_LOGIC);
end I8051_ALU;

 architecture BHV of I8051_ALU is


    procedure PCSADD (a : UNSIGNED (15 downto 0);
                      b : UNSIGNED (7 downto 0);
                      r : out UNSIGNED (15 downto 0)) is

        variable v1, v2 : SIGNED (15 downto 0);
    begin
        v1 := SIGNED(a);
        if( b(7) = '1' ) then
            v2 := SIGNED(CM_8 & b);
        else
            v2 := SIGNED(C0_8 & b);
        end if;
        v1 := v1 + v2;
        r := UNSIGNED(v1);
    end PCSADD;


    procedure PCUADD (a : UNSIGNED (15 downto 0);
                      b : UNSIGNED (7 downto 0);
                      r : out UNSIGNED (15 downto 0)) is
    begin
        r := a + b;
    end PCUADD;

    procedure DO_ADD (a, b : in  UNSIGNED (7 downto 0);
                      c : in  STD_LOGIC;
                      r : out UNSIGNED (7 downto 0);
                      cy, ac, ov : out STD_LOGIC) is
        variable v1, v2, v3, v4 : UNSIGNED (4 downto 0);
        variable v5, v6, v7, v8 : UNSIGNED (3 downto 0);
        variable v9, vA, vB, vC : UNSIGNED (1 downto 0);
    begin
        v1 := "0" & a(3 downto 0);
        v2 := "0" & b(3 downto 0);
        v3 := "0" & "000" & c;
        v4 := v1 + v2 + v3;
        v5 := "0" & a(6 downto 4);
```

```
    v6 := "0" & b(6 downto 4);
    v7 := "0" & "00" & v4(4);
    v8 := v5 + v6 + v7;
    v9 := "0" & a(7);
    vA := "0" & b(7);
    vB := "0" & v8(3);
    vC := v9 + vA + vB;
    r(7) := vC(0);
    r(6) := v8(2);
    r(5) := v8(1);
    r(4) := v8(0);
    r(3) := v4(3);
    r(2) := v4(2);
    r(1) := v4(1);
    r(0) := v4(0);
    cy := vC(1);
    ac := v4(4);
    ov := vC(1) xor v8(3);
end DO_ADD;


procedure DO_SUB (a, b : in  UNSIGNED (7 downto 0);
                  c : in  STD_LOGIC;
                  r : out UNSIGNED (7 downto 0);
                  cy, ac, ov : out STD_LOGIC) is
    variable v1, v2, v3, v4 : UNSIGNED (4 downto 0);
    variable v5, v6, v7, v8 : UNSIGNED (3 downto 0);
    variable v9, vA, vB, vC : UNSIGNED (1 downto 0);
begin
    v1 := "1" & a(3 downto 0);
    v2 := "0" & b(3 downto 0);
    v3 := "0" & "000" & c;
    v4 := v1 - v2 - v3;
    v5 := "1" & a(6 downto 4);
    v6 := "0" & b(6 downto 4);
    v7 := "0" & "00" & (not v4(4));
    v8 := v5 - v6 - v7;
    v9 := "1" & a(7);
    vA := "0" & b(7);
    vB := "0" & (not v8(3));
    vC := v9 - vA - vB;
    r(7) := vC(0);
    r(6) := v8(2);
    r(5) := v8(1);
    r(4) := v8(0);
    r(3) := v4(3);
    r(2) := v4(2);
    r(1) := v4(1);
    r(0) := v4(0);
    cy := not vC(1);
    ac := not v4(4);
    ov := (not vC(1)) xor (not v8(3));
end DO_SUB;


procedure DO_MUL(a, b : in UNSIGNED (7 downto 0);
                 r : out UNSIGNED (15 downto 0);
                 ov : out STD_LOGIC) is
    variable v1 : UNSIGNED (15 downto 0);
begin
    v1 := a * b;
    r := v1;
```

```
    if( v1(15 downto 8) /= C0_8 ) then
        ov := '1';
    else
        ov := '0';
    end if;
end DO_MUL;


procedure DO_DIV(a, b : in UNSIGNED (7 downto 0);
                 r : out UNSIGNED (15 downto 0);
                 ov : out STD_LOGIC) is
    variable v1 : UNSIGNED (15 downto 0);
    variable v2, v3 : UNSIGNED (8 downto 0);
begin
    if( b = C0_8 ) then
        r(7 downto 0) := CD_8;
        r(15 downto 8) := CD_8;
        ov := '1';
    elsif( a = b ) then
        r(7 downto 0) := C1_8;
        r(15 downto 8) := C0_8;
        ov := '0';
    elsif( a < b ) then
        r(7 downto 0) := C0_8;
        r(15 downto 8) := src_1;
        ov := '0';
    else
        v1(7 downto 0) := a;
        v1(15 downto 8) := C0_8;
        v3 := "0" & b;
        for i in 0 to 7 loop
            v1(15 downto 1) := v1(14 downto 0);
            v1(0) := '0';
            v2 := "1" & v1(15 downto 8);
            v2 := v2 - v3;
            if( v2(8) = '1' ) then
                v1(0) := '1';
                v1(15 downto 8) := v2(7 downto 0);
            end if;
        end loop;
        r := v1;
        ov := '0';
    end if;
end DO_DIV;


procedure DO_DA(a : in UNSIGNED (7 downto 0);
                c1, c2 : in STD_LOGIC;
                r : out UNSIGNED (7 downto 0);
                cy : out STD_LOGIC) is
    variable v : UNSIGNED (8 downto 0);
begin
    v := "0" & a;
    if( (c2 = '1') or (v(3 downto 0) > C9_4) ) then
        v := v + "000000110";
    end if;
    v(8) := v(8) or c1;
    if( v(8) = '1' or (v(7 downto 4) > C9_4) )then
        v := v + "001100000";
    end if;
    r := v(7 downto 0);
    cy := v(8);
```

```
        end DO_DA;


    begin
        process(rst, op_code, src_1, src_2, src_3, src_cy, src_ac)
            variable v16 : UNSIGNED (15 downto 0);
            variable v8 : UNSIGNED (7 downto 0);
            variable v_cy, v_ac, v_ov : STD_LOGIC;
        begin
            if( rst = '1' ) then
                des_1 <= CD_8;
                des_2 <= CD_8;
                des_cy <= '-';
                des_ac <= '-';
                des_ov <= '-';
            else
                case op_code is
                    when ALU_OPC_ADD    =>
                        DO_ADD(src_1,  src_2,  src_cy,  v8,  v_cy,  v_ac,
v_ov);
                        des_1 <= v8;
                        des_2 <= CD_8;
                        des_cy <= v_cy;
                        des_ac <= v_ac;
                        des_ov <= v_ov;
                    when ALU_OPC_SUB    =>
                        DO_SUB(src_1,  src_2,  src_cy,  v8,  v_cy,  v_ac,
v_ov);
                        des_1 <= v8;
                        des_2 <= CD_8;
                        des_cy <= v_cy;
                        des_ac <= v_ac;
                        des_ov <= v_ov;
                    when ALU_OPC_MUL    =>
                        DO_MUL(src_1, src_2, v16, v_ov);
                        des_1 <= v16(7 downto 0);
                        des_2 <= v16(15 downto 8);
                        des_cy <= '-';
                        des_ac <= '-';
                        des_ov <= v_ov;
                    when ALU_OPC_DIV    =>
                        DO_DIV(src_1, src_2, v16, v_ov);
                        des_1 <= v16(7 downto 0);
                        des_2 <= v16(15 downto 8);
                        des_cy <= '-';
                        des_ac <= '-';
                        des_ov <= v_ov;
                    when ALU_OPC_DA     =>
                        DO_DA(src_1, src_cy, src_ac, v8, v_cy);
                        des_1 <= v8;
                        des_2 <= CD_8;
                        des_cy <= v_cy;
                        des_ac <= '-';
                        des_ov <= '-';
                    when ALU_OPC_NOT    =>
                        des_1(7) <= not src_1(7);
                        des_1(6) <= not src_1(6);
                        des_1(5) <= not src_1(5);
                        des_1(4) <= not src_1(4);
                        des_1(3) <= not src_1(3);
                        des_1(2) <= not src_1(2);
```

```
            des_1(1) <= not src_1(1);
            des_1(0) <= not src_1(0);
            des_2 <= CD_8;
            des_cy <= '-';
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_AND    =>
            des_1(7) <= src_1(7) and src_2(7);
            des_1(6) <= src_1(6) and src_2(6);
            des_1(5) <= src_1(5) and src_2(5);
            des_1(4) <= src_1(4) and src_2(4);
            des_1(3) <= src_1(3) and src_2(3);
            des_1(2) <= src_1(2) and src_2(2);
            des_1(1) <= src_1(1) and src_2(1);
            des_1(0) <= src_1(0) and src_2(0);
            des_2 <= CD_8;
            des_cy <= '-';
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_XOR    =>
            des_1(7) <= src_1(7) xor src_2(7);
            des_1(6) <= src_1(6) xor src_2(6);
            des_1(5) <= src_1(5) xor src_2(5);
            des_1(4) <= src_1(4) xor src_2(4);
            des_1(3) <= src_1(3) xor src_2(3);
            des_1(2) <= src_1(2) xor src_2(2);
            des_1(1) <= src_1(1) xor src_2(1);
            des_1(0) <= src_1(0) xor src_2(0);
            des_2 <= CD_8;
            des_cy <= '-';
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_OR     =>
            des_1(7) <= src_1(7) or src_2(7);
            des_1(6) <= src_1(6) or src_2(6);
            des_1(5) <= src_1(5) or src_2(5);
            des_1(4) <= src_1(4) or src_2(4);
            des_1(3) <= src_1(3) or src_2(3);
            des_1(2) <= src_1(2) or src_2(2);
            des_1(1) <= src_1(1) or src_2(1);
            des_1(0) <= src_1(0) or src_2(0);
            des_2 <= CD_8;
            des_cy <= '-';
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_RL     =>
            des_1(0) <= src_1(7);
            des_1(7 downto 1) <= src_1(6 downto 0);
            des_2 <= CD_8;
            des_cy <= '-';
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_RLC    =>
            des_1(0) <= src_cy;
            des_1(7 downto 1) <= src_1(6 downto 0);
            des_2 <= CD_8;
            des_cy <= src_1(7);
            des_ac <= '-';
            des_ov <= '-';
        when ALU_OPC_RR     =>
            des_1(7) <= src_1(0);
```

```
                    des_1(6 downto 0) <= src_1(7 downto 1);
                    des_2 <= CD_8;
                    des_cy <= '-';
                    des_ac <= '-';
                    des_ov<= '-';
                when ALU_OPC_RRC    =>
                    des_1(7) <= src_cy;
                    des_1(6 downto 0) <= src_1(7 downto 1);
                    des_2 <= CD_8;
                    des_cy <= src_1(0);
                    des_ac <= '-';
                    des_ov <= '-';
                when ALU_OPC_PCSADD =>
                    PCSADD(src_2 & src_1, src_3, v16);
                    des_1 <= v16(7 downto 0);
                    des_2 <= v16(15 downto 8);
                    des_cy <= '-';
                    des_ac <= '-';
                    des_ov <= '-';
                when ALU_OPC_PCUADD =>
                    PCUADD(src_2 & src_1, src_3, v16);
                    des_1 <= v16(7 downto 0);
                    des_2 <= v16(15 downto 8);
                    des_cy <= '-';
                    des_ac <= '-';
                    des_ov <= '-';
                when others         =>
                    des_1 <= CD_8;
                    des_2 <= CD_8;
                    des_cy <= '-';
                    des_ac <= '-';
                    des_ov <= '-';
            end case;
        end if;
    end process;
end BHV;
```

APÊNDICE B – CÓDIGO VHDL DO BLOCO “RAM” DO 8051

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use WORK.I8051_LIB.all;

entity I8051_RAM is
    port(rst          : in  STD_LOGIC;
         clk          : in  STD_LOGIC;
         addr         : in  UNSIGNED (7 downto 0);
         in_data      : in  UNSIGNED (7 downto 0);
         out_data     : out UNSIGNED (7 downto 0);
         in_bit_data  : in  STD_LOGIC;
         out_bit_data : out STD_LOGIC;
         rd           : in  STD_LOGIC;
         wr           : in  STD_LOGIC;
         is_bit_addr  : in  STD_LOGIC;
         p0_in        : in  UNSIGNED (7 downto 0);
         p0_out       : out UNSIGNED (7 downto 0);
         p1_in        : in  UNSIGNED (7 downto 0);
         p1_out       : out UNSIGNED (7 downto 0);
         p2_in        : in  UNSIGNED (7 downto 0);
         p2_out       : out UNSIGNED (7 downto 0);
         p3_in        : in  UNSIGNED (7 downto 0);
         p3_out       : out UNSIGNED (7 downto 0));
end I8051_RAM;


architecture BHV of I8051_RAM is
    type IRAM_TYPE is array (0 to 127) of UNSIGNED (7 downto 0);
    signal iram : IRAM_TYPE;
    signal sfr_b    : UNSIGNED (7 downto 0);
    signal sfr_acc  : UNSIGNED (7 downto 0);
    signal sfr_psw  : UNSIGNED (7 downto 0);
    signal sfr_ie   : UNSIGNED (7 downto 0);
    signal sfr_ip   : UNSIGNED (7 downto 0);
    signal sfr_sp   : UNSIGNED (7 downto 0);
    signal sfr_dpl  : UNSIGNED (7 downto 0);
    signal sfr_dph  : UNSIGNED (7 downto 0);
    signal sfr_pcon : UNSIGNED (7 downto 0);
    signal sfr_scon : UNSIGNED (7 downto 0);
    signal sfr_sbuf : UNSIGNED (7 downto 0);
    signal sfr_tcon : UNSIGNED (7 downto 0);
    signal sfr_tmod : UNSIGNED (7 downto 0);
    signal sfr_tl0  : UNSIGNED (7 downto 0);
    signal sfr_th0  : UNSIGNED (7 downto 0);
    signal sfr_tl1  : UNSIGNED (7 downto 0);
    signal sfr_th1  : UNSIGNED (7 downto 0);
begin
    process(rst, clk)


        procedure GET_BYTE (a : UNSIGNED (7 downto 0);
                            v : out UNSIGNED (7 downto 0)) is
        begin
            case a is
                when R_B    => v := sfr_b;
                when R_ACC  => v := sfr_acc;
                when R_PSW  => v := sfr_psw;
                when R_IP   => v := sfr_ip;
                when R_IE   => v := sfr_ie;
```

```
                    when R_SP   => v := sfr_sp;
                    when R_P0   => v := p0_in;
                    when R_P1   => v := p1_in;
                    when R_P2   => v := p2_in;
                    when R_P3   => v := p3_in;
                    when R_DPL  => v := sfr_dpl;
                    when R_DPH  => v := sfr_dph;
                    when R_PCON => v := sfr_pcon;
                    when R_SCON => v := sfr_scon;
                    when R_SBUF => v := sfr_sbuf;
                    when R_TCON => v := sfr_tcon;
                    when R_TMOD => v := sfr_tmod;
                    when R_TL0  => v := sfr_tl0;
                    when R_TL1  => v := sfr_tl1;
                    when R_TH0  => v := sfr_th0;
                    when R_TH1  => v := sfr_th1;
                    when  others  =>  v  :=  iram(conv_integer(a(6  downto
0)));
                end case;
            end GET_BYTE;
            procedure SET_BYTE (a : UNSIGNED (7 downto 0);
                                v : UNSIGNED (7 downto 0)) is
            begin
                case a is
                    when R_B    => sfr_b <= v;
                    when R_ACC  => sfr_acc <= v;
                    when R_PSW  => sfr_psw <= v;
                    when R_IP   => sfr_ip <= v;
                    when R_IE   => sfr_ie <= v;
                    when R_SP   => sfr_sp <= v;
                    when R_P0   => p0_out <= v;
                    when R_P1   => p1_out <= v;
                    when R_P2   => p2_out <= v;
                    when R_P3   => p3_out <= v;
                    when R_DPL  => sfr_dpl <= v;
                    when R_DPH  => sfr_dph <= v;
                    when R_PCON => sfr_pcon <= v;
                    when R_SCON => sfr_scon <= v;
                    when R_SBUF => sfr_sbuf <= v;
                    when R_TCON => sfr_tcon <= v;
                    when R_TMOD => sfr_tmod <= v;
                    when R_TL0  => sfr_tl0 <= v;
                    when R_TL1  => sfr_tl1 <= v;
                    when R_TH0  => sfr_th0 <= v;
                    when R_TH1  => sfr_th1 <= v;
                    when  others  =>  iram(conv_integer(a(6  downto  0)))  <=
v;
                end case;
            end SET_BYTE;
            procedure  GET_BIT  (a  :  UNSIGNED  (7  downto  0);  v  :  out
STD_LOGIC) is
                variable vu : UNSIGNED (7 downto 0);
                variable vi : INTEGER;
            begin
                vu := a(7 downto 3) & "000";
                vi := conv_integer(a(2 downto 0));
                case vu is
                    when R_B    => v := sfr_b(vi);
                    when R_ACC  => v := sfr_acc(vi);
                    when R_PSW  => v := sfr_psw(vi);
                    when R_IP   => v := sfr_ip(vi);
```

```
                    when R_IE   => v := sfr_ie(vi);
                    when R_SP   => v := sfr_sp(vi);
                    when R_P0   => v := p0_in(vi);
                    when R_P1   => v := p1_in(vi);
                    when R_P2   => v := p2_in(vi);
                    when R_P3   => v := p3_in(vi);
                    when R_SCON => v := sfr_scon(vi);
                    when R_TCON => v := sfr_tcon(vi);
                    when others =>
                        vu := "0010" & a(6 downto 3);
                        v := iram(conv_integer(vu))(vi);
                end case;
            end GET_BIT;
            procedure SET_BIT (a : UNSIGNED (7 downto 0); v : STD_LOGIC)
is
                variable vu : UNSIGNED (7 downto 0);
                variable vi : INTEGER;
            begin
                vu := a(7 downto 3) & "000";
                vi := conv_integer(a(2 downto 0));
                case vu is
                    when R_B    => sfr_b(vi) <= v;
                    when R_ACC  => sfr_acc(vi) <= v;
                    when R_PSW  => sfr_psw(vi) <= v;
                    when R_IP   => sfr_ip(vi) <= v;
                    when R_IE   => sfr_ie(vi) <= v;
                    when R_SP   => sfr_sp(vi) <= v;
                    when R_P0   => p0_out(vi) <= v;
                    when R_P1   => p1_out(vi) <= v;
                    when R_P2   => p2_out(vi) <= v;
                    when R_P3   => p3_out(vi) <= v;
                    when R_SCON => sfr_scon(vi) <= v;
                    when R_TCON => sfr_tcon(vi) <= v;
                    when others =>
                        vu := "0010" & a(6 downto 3);
                        iram(conv_integer(vu))(vi) <= v;
                end case;
            end SET_BIT;
            variable v8 : UNSIGNED (7 downto 0);
            variable v1 : STD_LOGIC;
        begin
            if( rst = '1' ) then
                for i in 0 to 127 loop
                    iram(i) <= CD_8;
                end loop;
                sfr_b    <= C0_8;
                sfr_acc  <= C0_8;
                sfr_psw  <= C0_8;
                sfr_ie   <= C0_8;
                sfr_ip   <= C0_8;
                sfr_sp   <= CD_8;
                sfr_dpl  <= C0_8;
                sfr_dph  <= C0_8;
                sfr_pcon <= C0_8;
                sfr_scon <= C0_8;
                sfr_sbuf <= C0_8;
                sfr_tcon <= C0_8;
                sfr_tmod <= C0_8;
                sfr_tl0  <= C0_8;
                sfr_th0  <= C0_8;
                sfr_tl1  <= C0_8;
```

```
            sfr_th1  <= C0_8;
            out_data <= CD_8;
            out_bit_data <= '-';
            p0_out <= CD_8;
            p1_out <= CD_8;
            p2_out <= CD_8;
            p3_out <= CD_8;
        elsif( clk'event and clk = '1' ) then
            if( rd = '1' ) then
                if( is_bit_addr = '1' ) then
                    GET_BIT(addr, v1);
                    out_bit_data <= v1;
                else
                    GET_BYTE(addr, v8);
                    out_data <= v8;
                end if;
            elsif( wr = '1' ) then
                if( is_bit_addr = '1' ) then
                    SET_BIT(addr, in_bit_data);
                else
                    SET_BYTE(addr, in_data);
                end if;
            end if;
        end if;
    end process;
end BHV;
```

APÊNDICE C – TUTORIAL DE UTILIZAÇÃO DO SOFTWARE *QUARTUS* II PARA CRIAÇÃO DE BLOCOS VHDL

Inicialmente, deve-se criar um novo projeto de um CI. No menu de arquivos (*File*) é selecionada a opção *New Project Wizard.* A primeira janela, conforme apresentado na Figura C.1, apresentará alguns campos importantes a serem preenchidos: o caminho do diretório de trabalho, o nome do projeto e o nome da entidade de mais alto nível [34].

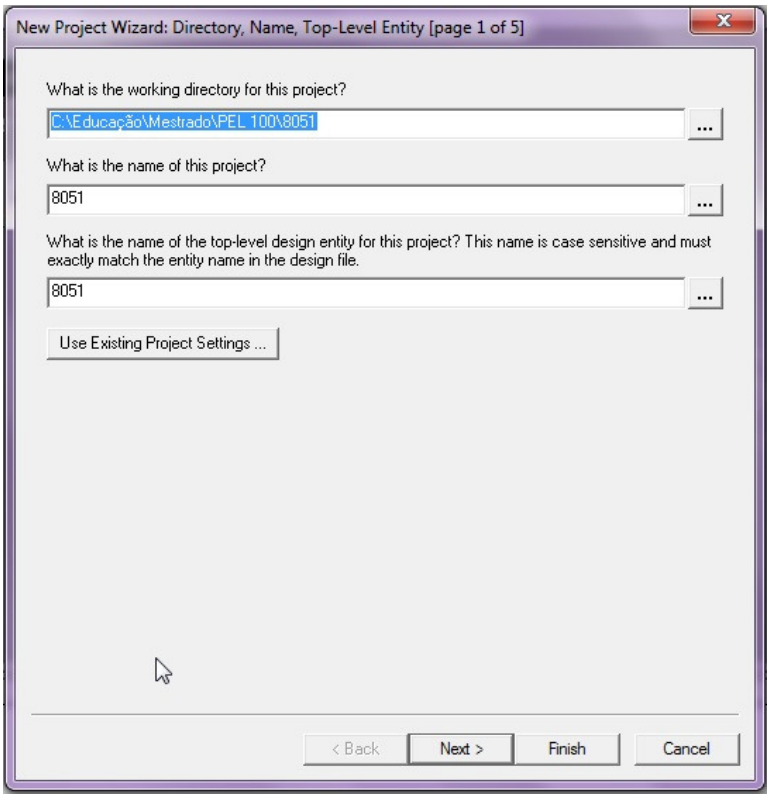


Figura C. 1 - Primeira janela da criação de um novo projeto no *Quartus* II.

Acionando-se o botão *Next* é apresentada a segunda janela, que não necessita ser preenchida. Ela é utilizada para inserção de arquivos VHDL, etapa que pode ser realizada depois, se necessário. Acionando-se novamente o botão *Next* será mostrada a terceira janela, que deverá ser selecionada a família de dispositivos FPGA. Neste caso foi selecionada a família *Cyclone II*. Nessa mesma janela deve-se selecionar o modelo do dispositivo dentro desta família. A Figura C.2 ilustra esta janela de configuração [34].

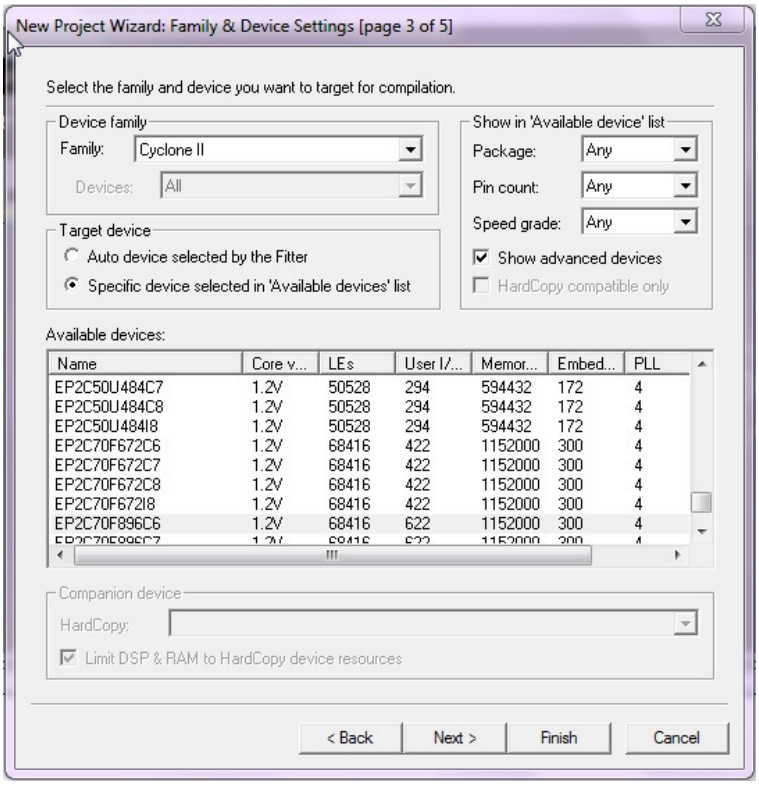


Figura C. 2 - Terceira janela da criação de um novo projeto no *Quartus* II.

A informação do modelo de FPGA dentro da família *Cyclone II* encontra-se impressa na máscara do *chip* central da placa, abaixo dos nomes impressos *Altera* e *Cyclone II*. Acionando-se o botão *Next* é apresentada a quarta janela, utilizada apenas se outro *software* EDA for utilizado para síntese, simulação ou análise. Acionando-se o botão *Next,* a quinta e última janela, conforme a Figura C.3, apresenta um resumo das informações do projeto. Clicando no botão *Finish* o projeto estará criado [34].

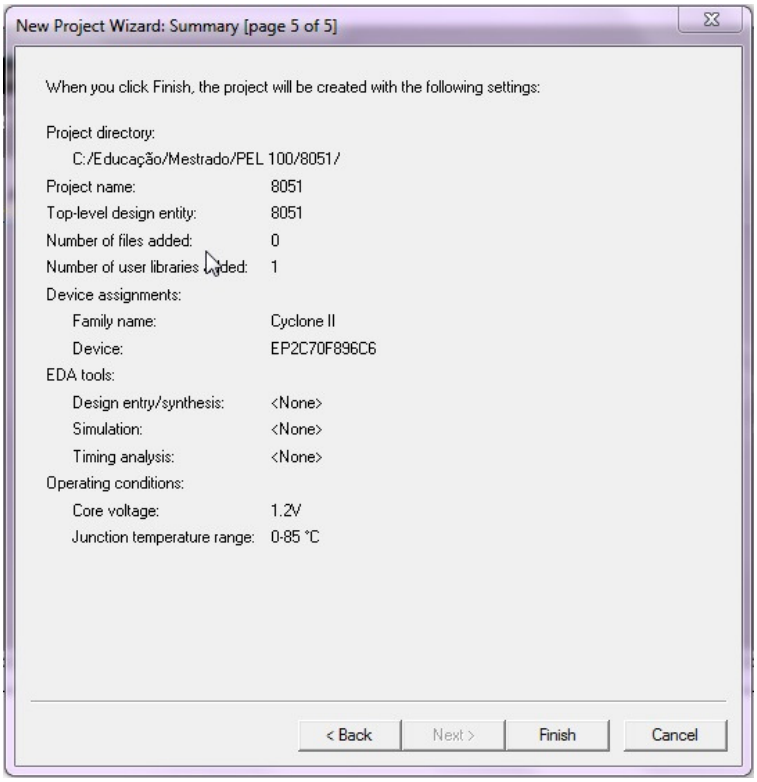


Figura C. 3 - Quinta janela da criação de um novo projeto no *Quartus* II.

A próxima etapa é de inserção dos códigos VHDL no projeto. Para tal, deve-se abrir o editor VHDL no caminho: *File / New / VHDL File.* Na janela que será aberta, digita-se o código VHDL. Ao término da edição, deve-se salvar e compilar este bloco VHDL, clicando no botão *Start Compilation,* posicionado na barra de ferramentas na parte superior da tela [21].

Após a compilação é possível, inclusive, verificar o esquemático resultante do código VHDL, seguindo o caminho: *File* / *Create* / *Create Symbol Files for Current File*. Será gerado um arquivo RTL que pode ser visualizado através do caminho: *Tools* / *Netlist Viewers* / RTL *Viewer*. Serão apresentados blocos que, ao serem clicados duplamente desagrupam em partes menores para a visualização do esquemático.

A seguir, serão apresentados os passos para a gravação do projeto em FPGA. No menu do *Quartus* II, acessar *Assignments* / *Pin planner.* É aberta uma janela dentro do *Quartus* com o aplicativo para a configuração dos pinos da estrutura FPGA, conforme demonstra a Figura C.4. Neste caso foram definidos as portas de entrada P0 e P1, e as portas de saída P0 e P1. Todas as portas são de 8 *bits.*

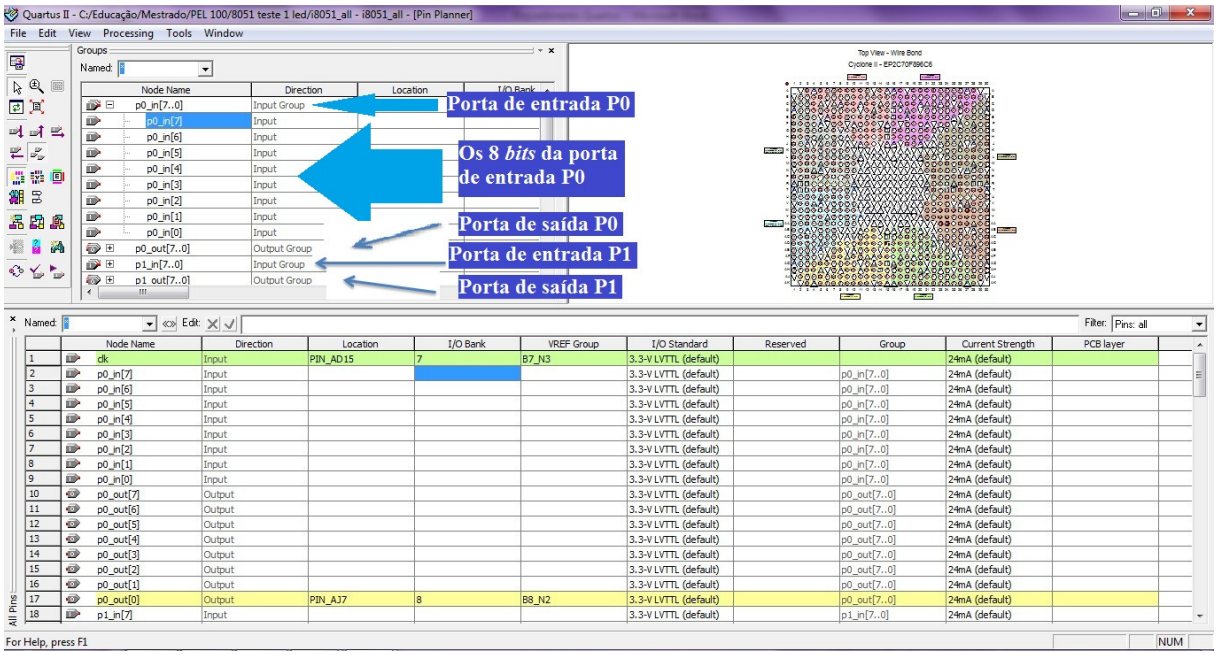


Figura C. 4: Tela do *Pin Planner* do *software Quartus II* para a configuração dos pinos da placa FPGA.

Posteriormente foi realizada a transferência do código VHDL do projeto para a estrutura FPGA. É necessário conectar um cabo USB, do conector USB da placa FPGA para o conector USB do computador. Conectar a fonte de alimentação ao respectivo conector da placa, e pressionar o botão vermelho da placa FPGA para energizá-la. Será executada uma rotina inicial de teste. Então o aplicativo *Pin planner* pode ser fechado.

Retornando à tela inicial do *Quartus* II, na barra de ferramentas da parte superior, existe um ícone denominado *Programmer.* Ao clicar neste ícone, será aberta uma nova janela dentro do *software Quartus*. Se a conexão USB for validada e realmente estiver operando corretamente a comunicação entre o *kit* da placa didática FPGA e o computador, a placa será detectada automaticamente, então pressionando o botão *Start,* o código VHDL do projeto será gravado na placa didática do FPGA.

Este procedimento deve ser repetido para cada projeto que se deseje realizar o teste de funcionamento do VHDL. Depois de programado o FPGA, o teste consiste simplesmente em acionar as entradas e observar o estado das saídas.

APÊNDICE D – CÓDIGO VERILOG DO BLOCO “ULA” DO 8051

```
module I8051_ALU ( rst, op_code, src_1, src_2, src_3, src_cy,
src_ac, des_1, des_2, des_cy, des_ac, des_ov ) ;
    input rst ;
    input [3:0]op_code ;
    input [7:0]src_1 ;
    input [7:0]src_2 ;
    input [7:0]src_3 ;
    input src_cy ;
    input src_ac ;
    output [7:0]des_1 ;
    output [7:0]des_2 ;
    output des_cy ;
    output des_ac ;
    output des_ov ;
    wire int_src_3_7, int_src_3_6, int_src_3_5, int_src_3_4,
int_src_3_3,int_src_3_2, int_src_3_1, int_src_3_0, int_src_ac,
int_des_1_7,int_des_1_6, int_des_1_5, int_des_1_4, int_des_1_3,
int_des_1_2,int_des_1_1, int_des_1_0, int_des_2_7, int_des_2_6,
int_des_2_5, int_des_2_4, int_des_2_3, int_des_2_2, int_des_2_1,
int_des_2_0, int_des_cy, int_des_ac, int_des_ov, nx2, nx4, nx10,
nx20, nx24, nx28, nx34, nx46, nx56, nx68, nx74, nx78, nx82, nx88,
nx90, nx100, nx102, nx106, nx112, nx124, nx132, nx140, nx150, nx164,
nx174, nx184, nx192, nx204, nx208, nx214, nx220, nx236, nx240,
nx256, nx268, nx272, nx276, nx278, nx288, nx298, nx304, nx308,
nx322, nx326, nx340, nx344, nx354, nx358, nx362, nx366, nx372,
nx394, nx398, nx412, nx416, nx436, nx450, nx462, nx466, nx470,
nx474, nx478, nx480, nx490, nx500, nx508, nx512, nx526, nx530,
nx550, nx562, nx566, nx580, nx584, nx594, nx598, nx602, nx606,
nx610, nx614, nx638, nx650, nx654, nx668, nx672, nx692, nx704,
nx708, nx726, nx728, nx748, nx758, nx762, nx766, nx770, nx774,
nx778, nx782, nx786, nx788, nx792, nx802, nx804, nx812, nx814,
nx816, nx824, nx830, nx832, nx836, nx842, nx844, nx848, nx850,
nx852, nx856, nx866, nx868, nx872, nx874, nx876, nx888, nx890,
nx894, nx902, nx906, nx920, nx926, nx938, nx950, nx952, nx956,
nx964, nx968, nx970, nx996, nx1006, nx1014, nx1016, nx1040, nx1042,
nx1050, nx1052, nx1054, nx1070, nx1072, nx1080, nx1082, nx1096,
nx1114, nx1118, nx1128, nx1156, nx1162, nx1172, nx1174, nx1180,
nx1182, nx1184, nx1200, nx1202, nx1212, nx1226, nx1242, nx1250,
nx1252, nx1280, nx1288, nx1292, nx1302, nx1306, nx1308, nx1312,
nx1314, nx1318, nx1334, nx1336, nx1344, nx1346, nx1374, nx1382,
nx1384, nx1402, nx1410, nx1418, nx1422, nx1432, nx1434, nx1438,
nx1440, nx1444, nx1446, nx1450, nx1452, nx1456, nx1468, nx1474,
nx1480, nx1490, nx1492, nx1494, nx1496, nx1504, nx1516, nx1518,
nx1544, nx1554, nx1564, nx1592, nx1600, nx1608, nx1616, nx1620,
nx1630, nx1632, nx1636, nx1640, nx1642, nx1646, nx1648, nx1652,
nx1654, nx1658, nx1674, nx1676, nx1684, nx1686, nx1712, nx1720,
nx1736, nx1738, nx1766, nx1774, nx1782, nx1790, nx1798, nx1800,
nx1802, nx1804, nx1814, nx1816, nx1820, nx1822, nx1826, nx1828,
nx1832, nx1834, nx1838, nx1840, nx1844, nx1846, nx1850, nx1866,
nx1868, nx1878, nx1900, nx1902, nx1904, nx1914, nx1922, nx1934,
nx1942, nx1950, nx1958, nx1966, nx1974, nx1976, nx1980, nx1982,
nx1986, nx1990, nx1992, nx1996, nx1998, nx2002, nx2004, nx2008,
nx2024, nx2028, nx2052, nx2054, nx2064, nx2072, nx2080, nx2088,
nx2096, nx2106, nx2108, nx2114, nx2116, nx2120, nx2122, nx2126,
```

```
nx2128, nx2132, nx2134, nx2138, nx2140, nx2148, nx2152, nx2154,
nx2176, nx2188, nx2196, nx2204, nx2212, nx2220, nx2228, nx2230,
nx2232, nx2236, nx2240, nx2242, nx2246, nx2248, nx2252, nx2264,
nx2268, nx2292, nx2294, nx2304, nx2312, nx2320, nx2328, nx2336,
nx2338, nx2340, nx2344, nx2346, nx2350, nx2352, nx2356, nx2360,
nx2374, nx2386, nx2396, nx2398, nx2408, nx2416, nx2424, nx2432,
nx2434, nx2438, nx2440, nx2444, nx2448, nx2456, nx2460, nx2484,
nx2486, nx2496, nx2504, nx2512, nx2514, nx2516, nx2520, nx2524,
nx2532, nx2536, nx2538, nx2560, nx2562, nx2572, nx2580, nx2584,
nx2614, nx2616, nx2626, nx2640, nx2658, nx2666, nx2694, nx2704,
nx2708, nx2722, NOT_op_code_3, NOT_op_code_2,NOT_op_code_1,
NOT_op_code_0, NOT_src_1_7, NOT_src_1_6, NOT_src_1_5, NOT_src_1_4,
NOT_src_1_3, NOT_src_1_2, NOT_src_1_1, NOT_src_1_0, NOT_src_2_7,
NOT_src_2_6, NOT_src_2_5, NOT_src_2_4, NOT_src_2_3, NOT_src_2_2,
NOT_src_2_1, NOT_src_2_0, NOT_src_cy, nx2409, nx2423, nx2441,
nx2451, nx2457, nx2461, nx2467, nx2475, nx2481, nx2487, nx2493,
nx2501, nx2507, nx2513, nx2517, nx2523, nx2535, nx2541, nx2547,
nx2555, nx2559, nx2571, nx2577, nx2583, nx2589, nx2591, nx2595,
nx2609, nx2631, nx2637, nx2639, nx2643, nx2645, nx2649, nx2655,
nx2663, nx2671, nx2677, nx2684, nx2689, nx2697, nx2703, nx2713,
nx2725, nx2731, nx2735, nx2741, nx2745, nx2751, nx2756, nx2760,
nx2764, nx2770, nx2775, nx2779, nx2784, nx2789, nx2793, nx2801,
nx2803, nx2805, nx2807, nx2809, nx2811, nx2814, nx2817, nx2820,
nx2823, nx2825, nx2844, nx2847, nx2854, nx2861, nx2868, nx2881,
nx2893, nx2896, nx2902, nx2905, nx2908, nx2910, nx2914, nx2916,
nx2920, nx2922, nx2925, nx2928, nx2931, nx2936, nx2939, nx2941,
nx2943, nx2950, nx2952, nx2954, nx2956, nx2958, nx2960, nx2962,
nx2964, nx2974, nx2976, nx2979, nx2996, nx2999, nx3001, nx3003,
nx3012, nx3017, nx3024, nx3029, nx3033, nx3036, nx3038, nx3040,
nx3043, nx3046, nx3051, nx3054, nx3056, nx3061, nx3064, nx3066,
nx3070, nx3074, nx3078, nx3081, nx3083, nx3086, nx3088, nx3095,
nx3098, nx3101, nx3103, nx3105, nx3109, nx3113, nx3118, nx3120,
nx3122, nx3126, nx3130, nx3132, nx3135, nx3138, nx3140, nx3142,
nx3147, nx3149, nx3151, nx3155, nx3160, nx3163, nx3165, nx3169,
nx3172, nx3178, nx3181, nx3186, nx3188, nx3191, nx3194, nx3199,
nx3202, nx3206, nx3209, nx3215, nx3219, nx3222, nx3224, nx3226,
nx3228, nx3231, nx3234, nx3237, nx3240, nx3244, nx3247, nx3251,
nx3253, nx3256, nx3264,nx3266, nx3268, nx3270, nx3276, nx3281,
nx3284, nx3290, nx3295, nx3297, nx3301, nx3305, nx3308, nx3315,
nx3319, nx3324, nx3326, nx3329, nx3335, nx3339, nx3343, nx3354,
nx3359, nx3362, nx3365, nx3368, nx3370, nx3374, nx3379, nx3382,
nx3386, nx3389, nx3392, nx3394, nx3397, nx3400, nx3405, nx3408,
nx3411, nx3415, nx3418, nx3422, nx3426, nx3430, nx3435, nx3440,
nx3443, nx3446, nx3449, nx3455, nx3460, nx3463, nx3467, nx3472,
nx3474, nx3482, nx3484, nx3486, nx3488, nx3490, nx3492, nx3494,
nx3496, nx3498, nx3500, nx3502, nx3504, nx3506, nx3508, nx3510,
nx3512, nx3514, nx3516, nx3518, nx3520, nx3522, nx3524, nx3526,
nx3528, nx3530, nx3532, nx3534, nx3536, nx3538, nx3540, nx3542,
nx3544, nx3546, nx3548, nx3550, nx3552, nx3554, nx3558, nx3564,
nx3566, nx3570, nx3572, nx3576, nx3578, nx3582, nx3584, nx3588,
nx3590, nx3594, nx3596, nx3600, nx3602, nx3606, nx3608, nx3610,
nx3612, nx3614, nx3618, nx3620, nx3622, nx3626, nx3628, nx3632,
nx3634, nx3636, nx3640, nx3642, nx3644, nx3648, nx3650, nx3652,
nx3656, nx3658, nx3660, nx3666, nx3668, nx3674, nx3676, nx3678,
nx3682, nx3684, nx3686, nx3692, nx3694, nx3696, nx3698, nx3700,
nx3702, nx3704, nx3706, nx3710, nx3712, nx3714, nx3716, nx3718,
```

```
nx3720, nx3722, nx3724, nx3726, nx3728, nx3730, nx3732, nx3738,
nx3740, nx3742, nx3744, nx3746, nx3748, nx3750, nx3752, nx3754,
nx3756, nx3758, nx3760, nx3762, nx3764, nx3766, nx3768, nx3770,
nx3772, nx3774, nx3776, nx3778, nx3780, nx3782, nx3784, nx3786,
nx3788, nx3792, nx3794, nx3796, nx3798, nx3800, nx3802, nx3804,
nx3806, nx3808, nx3810, nx3812, nx3814, nx3816, nx3818, nx3820,
nx3822, nx3824, nx3826, nx3828, nx3830, nx3832, nx3834, nx3836,
nx3838, nx3840, nx3842, nx3844, nx3846, nx3848, nx3850, nx3852,
nx3854, nx3856, nx3858, nx3860, nx3862, nx3864, nx3866, nx3868,
nx3870, nx3872, nx3874, nx3876, nx3878, nx3880, nx3882, nx3884,
nx3886, nx3888, nx3890, nx3892, nx3894, nx3896;
    wire [29:0] \$dummy ;
    mux21_ni ix2727 (.Y (int_des_ov), .A0 (nx2722), .A1 (nx2694),
.S0 (NOT_op_code_0)) ;
    nor04 ix2410 (.Y (nx2409), .A0 (nx2708), .A1 (nx2524), .A2
(nx2448), .A3 (nx2704)) ;
    or03 ix2709 (.Y (nx2708), .A0 (nx2572), .A1 (nx890), .A2
(nx2580)) ;
    mux21_ni ix2573 (.Y (nx2572), .A0 (nx2496), .A1 (nx2520), .S0
(nx2793)) ;
    mux21_ni ix2497 (.Y (nx2496), .A0 (nx2408), .A1 (nx2444), .S0
(nx2784)) ;
    mux21_ni ix2409 (.Y (nx2408), .A0 (nx2304), .A1 (nx2356), .S0
(nx2770)) ;
    mux21_ni ix2305 (.Y (nx2304), .A0 (nx2188), .A1 (nx2252), .S0
(nx2751)) ;
    xnor2 ix2424 (.Y (nx2423), .A0 (nx2072), .A1 (nx2134)) ;
    mux21_ni ix2073 (.Y (nx2072), .A0 (nx1950), .A1 (nx1998), .S0
(nx2535)) ;
    mux21_ni ix1951 (.Y (nx1950), .A0 (nx1782), .A1 (nx1834), .S0
(nx2501)) ;
    mux21_ni ix1783 (.Y (nx1782), .A0 (nx1608), .A1 (nx1642), .S0
(nx2475)) ;
    mux21_ni ix1609 (.Y (nx1608), .A0 (nx1418), .A1 (nx1440), .S0
(nx2457)) ;
    mux21_ni ix1419 (.Y (nx1418), .A0 (nx1162), .A1 (nx1302), .S0
(nx2451)) ;
    nor02_2x ix1303 (.Y (nx1302), .A0 (nx1292), .A1 (nx2441)) ;
    aoi22 ix2442 (.Y (nx2441), .A0 (nx3864), .A1 (nx3674), .B0
(nx3870), .B1 (nx3682)) ;
    xnor2 ix2452 (.Y (nx2451), .A0 (nx832), .A1 (nx1302)) ;
    xnor2 ix2458 (.Y (nx2457), .A0 (nx1438), .A1 (nx1440)) ;
    xnor2 ix1439 (.Y (nx1438), .A0 (nx1292), .A1 (nx2461)) ;
    xnor2 ix2462 (.Y (nx2461), .A0 (nx1432), .A1 (nx1434)) ;
    nor02_2x ix1433 (.Y (nx1432), .A0 (nx1422), .A1 (nx2467)) ;
    aoi22 ix2468 (.Y (nx2467), .A0 (nx3802), .A1 (nx3674), .B0
(nx3864), .B1 (nx3682)) ;
    xnor2 ix2476 (.Y (nx2475), .A0 (nx1640), .A1 (nx1642)) ;
    xnor2 ix1641 (.Y (nx1640), .A0 (nx1616), .A1 (nx2481)) ;
    mux21_ni ix1617 (.Y (nx1616), .A0 (nx1292), .A1 (nx1434), .S0
(nx2461)) ;
    xnor2 ix2482 (.Y (nx2481), .A0 (nx844), .A1 (nx1636)) ;
    xnor2 ix1637 (.Y (nx1636), .A0 (nx1422), .A1 (nx2487)) ;
    xnor2 ix2488 (.Y (nx2487), .A0 (nx1630), .A1 (nx1632)) ;
    nor02_2x ix1631 (.Y (nx1630), .A0 (nx1620), .A1 (nx2493)) ;
```

```
    aoi22 ix2494 (.Y (nx2493), .A0 (nx3808), .A1 (nx3674), .B0
(nx3802), .B1 (nx3682)) ;
    xnor2 ix2502 (.Y (nx2501), .A0 (nx1832), .A1 (nx1834)) ;
    xnor2 ix1833 (.Y (nx1832), .A0 (nx1790), .A1 (nx2507)) ;
    mux21_ni ix1791 (.Y (nx1790), .A0 (nx1616), .A1 (nx1636), .S0
(nx2481)) ;
    xnor2 ix2508 (.Y (nx2507), .A0 (nx1826), .A1 (nx1828)) ;
    xnor2 ix1827 (.Y (nx1826), .A0 (nx1798), .A1 (nx2513)) ;
    mux21_ni ix1799 (.Y (nx1798), .A0 (nx1422), .A1 (nx1632), .S0
(nx2487)) ;
    xnor2 ix2514 (.Y (nx2513), .A0 (nx1820), .A1 (nx1822)) ;
    xnor2 ix1821 (.Y (nx1820), .A0 (nx1620), .A1 (nx2517)) ;
    xnor2 ix2518 (.Y (nx2517), .A0 (nx1814), .A1 (nx1816)) ;
    nor02_2x ix1815 (.Y (nx1814), .A0 (nx1804), .A1 (nx2523)) ;
    aoi22 ix2524 (.Y (nx2523), .A0 (nx3816), .A1 (nx3674), .B0
(nx3808), .B1 (nx3682)) ;
    xnor2 ix2536 (.Y (nx2535), .A0 (nx1996), .A1 (nx1998)) ;
    xnor2 ix1997 (.Y (nx1996), .A0 (nx1958), .A1 (nx2541)) ;
    mux21_ni ix1959 (.Y (nx1958), .A0 (nx1790), .A1 (nx1828), .S0
(nx2507)) ;
    xnor2 ix2542 (.Y (nx2541), .A0 (nx1990), .A1 (nx1992)) ;
    xnor2 ix1991 (.Y (nx1990), .A0 (nx1966), .A1 (nx2547)) ;
    mux21_ni ix1967 (.Y (nx1966), .A0 (nx1798), .A1 (nx1822), .S0
(nx2513)) ;
    xnor2 ix2548 (.Y (nx2547), .A0 (nx852), .A1 (nx1986)) ;
    xnor2 ix1987 (.Y (nx1986), .A0 (nx1974), .A1 (nx2555)) ;
    mux21_ni ix1975 (.Y (nx1974), .A0 (nx1620), .A1 (nx1816), .S0
(nx2517)) ;
    xnor2 ix2556 (.Y (nx2555), .A0 (nx1980), .A1 (nx1982)) ;
    xnor2 ix1981 (.Y (nx1980), .A0 (nx1804), .A1 (nx2559)) ;
    xnor2 ix2560 (.Y (nx2559), .A0 (nx1800), .A1 (nx1976)) ;
    xnor2 ix2135 (.Y (nx2134), .A0 (nx2571), .A1 (nx2132)) ;
    xnor2 ix2572 (.Y (nx2571), .A0 (nx2080), .A1 (nx2128)) ;
    mux21_ni ix2081 (.Y (nx2080), .A0 (nx1958), .A1 (nx1992), .S0
(nx2541)) ;
    xnor2 ix2129 (.Y (nx2128), .A0 (nx2577), .A1 (nx2126)) ;
    xnor2 ix2578 (.Y (nx2577), .A0 (nx2088), .A1 (nx2122)) ;
    mux21_ni ix2089 (.Y (nx2088), .A0 (nx1966), .A1 (nx1986), .S0
(nx2547)) ;
    xnor2 ix2123 (.Y (nx2122), .A0 (nx2583), .A1 (nx2120)) ;
    xnor2 ix2584 (.Y (nx2583), .A0 (nx2096), .A1 (nx2116)) ;
    mux21_ni ix2097 (.Y (nx2096), .A0 (nx1974), .A1 (nx1982), .S0
(nx2555)) ;
    xnor2 ix2117 (.Y (nx2116), .A0 (nx2589), .A1 (nx2114)) ;
    xnor2 ix2590 (.Y (nx2589), .A0 (nx2591), .A1 (nx2595)) ;
    oai21 ix2592 (.Y (nx2591), .A0 (nx1802), .A1 (nx1976),
.B0(nx1800));
    xnor2 ix2596 (.Y (nx2595), .A0 (nx2106), .A1 (nx2108)) ;
    nand02 ix2610 (.Y (nx2609), .A0 (nx1934), .A1 (nx2008)) ;
    mux21_ni ix1935 (.Y (nx1934), .A0 (nx1766), .A1 (nx1846), .S0
(nx2697)) ;
    mux21_ni ix1767 (.Y (nx1766), .A0 (nx1592), .A1 (nx1654), .S0
(nx2684)) ;
    mux21_ni ix1593 (.Y (nx1592), .A0 (nx1402), .A1 (nx1452), .S0
(nx2671)) ;
```

```
    mux21_ni ix1403 (.Y (nx1402), .A0 (nx1280), .A1 (nx1314), .S0
(nx2655)) ;
    mux21_ni ix1281 (.Y (nx1280), .A0 (nx1180), .A1 (nx1156), .S0
(nx1182)) ;
    mux21_ni ix1157 (.Y (nx1156), .A0 (nx1050), .A1 (nx814), .S0
(nx1052)) ;
    xnor2 ix1053 (.Y (nx1052), .A0 (nx2631), .A1 (nx1050)) ;
    oai21 ix2632 (.Y (nx2631), .A0 (nx1042), .A1 (nx802), .B0
(nx2637)) ;
    nand03 ix2638 (.Y (nx2637), .A0 (nx802), .A1 (nx3822),
.A2(nx3674)) ;
    inv02 ix2640 (.Y (nx2639), .A (NOT_src_2_2)) ;
    xnor2 ix1183 (.Y (nx1182), .A0 (nx2643), .A1 (nx1180)) ;
    xnor2 ix2644 (.Y (nx2643), .A0 (nx2637), .A1 (nx2645)) ;
    xnor2 ix2646 (.Y (nx2645), .A0 (nx1172), .A1 (nx1174)) ;
    nor02_2x ix1173 (.Y (nx1172), .A0 (nx1162), .A1 (nx2649)) ;
    aoi22 ix2650 (.Y (nx2649), .A0 (nx3870), .A1 (nx3674), .B0
(nx3822), .B1 (nx3682)) ;
    xnor2 ix2656 (.Y (nx2655), .A0 (nx1312), .A1 (nx1314)) ;
    xnor2 ix1313 (.Y (nx1312), .A0 (nx1288), .A1 (nx2663)) ;
    mux21_ni ix1289 (.Y (nx1288), .A0 (nx1040), .A1 (nx1174), .S0
(nx2645)) ;
    xnor2 ix2664 (.Y (nx2663), .A0 (nx1306), .A1 (nx1308)) ;
    xnor2 ix1307 (.Y (nx1306), .A0 (nx1162), .A1 (nx2451)) ;
    xnor2 ix2672 (.Y (nx2671), .A0 (nx1450), .A1 (nx1452)) ;
    xnor2 ix1451 (.Y (nx1450), .A0 (nx1410), .A1 (nx2677)) ;
    mux21_ni ix1411 (.Y (nx1410), .A0 (nx1288), .A1 (nx1308), .S0
(nx2663)) ;
    xnor2 ix2678 (.Y (nx2677), .A0 (nx1444), .A1 (nx1446)) ;
    xnor2 ix1445 (.Y (nx1444), .A0 (nx1418), .A1 (nx2457)) ;
    xnor2 ix2685 (.Y (nx2684), .A0 (nx1652), .A1 (nx1654)) ;
    xnor2 ix1653 (.Y (nx1652), .A0 (nx1600), .A1 (nx2689)) ;
    mux21_ni ix1601 (.Y (nx1600), .A0 (nx1410), .A1 (nx1446), .S0
(nx2677)) ;
    xnor2 ix2690 (.Y (nx2689), .A0 (nx1646), .A1 (nx1648)) ;
    xnor2 ix1647 (.Y (nx1646), .A0 (nx1608), .A1 (nx2475)) ;
    xnor2 ix2698 (.Y (nx2697), .A0 (nx1844), .A1 (nx1846)) ;
    xnor2 ix1845 (.Y (nx1844), .A0 (nx1774), .A1 (nx2703)) ;
    mux21_ni ix1775 (.Y (nx1774), .A0 (nx1600), .A1 (nx1648), .S0
(nx2689)) ;
    xnor2 ix2704 (.Y (nx2703), .A0 (nx1838), .A1 (nx1840)) ;
    xnor2 ix1839 (.Y (nx1838), .A0 (nx1782), .A1 (nx2501)) ;
    xnor2 ix2009 (.Y (nx2008), .A0 (nx1942), .A1 (nx2713)) ;
    mux21_ni ix1943 (.Y (nx1942), .A0 (nx1774), .A1 (nx1840), .S0
(nx2703)) ;
    xnor2 ix2714 (.Y (nx2713), .A0 (nx2002), .A1 (nx2004)) ;
    xnor2 ix2003 (.Y (nx2002), .A0 (nx1950), .A1 (nx2535)) ;
    xnor2 ix2139 (.Y (nx2138), .A0 (nx2064), .A1 (nx2423)) ;
    mux21_ni ix2065 (.Y (nx2064), .A0 (nx1942), .A1 (nx2004), .S0
(nx2713)) ;
    xnor2 ix2253 (.Y (nx2252), .A0 (nx2204), .A1 (nx2725)) ;
    mux21_ni ix2205 (.Y (nx2204), .A0 (nx2126), .A1 (nx2080), .S0
(nx2128)) ;
    xnor2 ix2726 (.Y (nx2725), .A0 (nx2246), .A1 (nx2248)) ;
    xnor2 ix2247 (.Y (nx2246), .A0 (nx2212), .A1 (nx2731)) ;
```

```
    mux21_ni ix2213 (.Y (nx2212), .A0 (nx2120), .A1 (nx2088), .S0
(nx2122)) ;
    xnor2 ix2732 (.Y (nx2731), .A0 (nx2240), .A1 (nx2242)) ;
    xnor2 ix2241 (.Y (nx2240), .A0 (nx2220), .A1 (nx2735)) ;
    mux21_ni ix2221 (.Y (nx2220), .A0 (nx2114), .A1 (nx2096), .S0
(nx2116)) ;
    xnor2 ix2736 (.Y (nx2735), .A0 (nx868), .A1 (nx2236)) ;
    xnor2 ix2237 (.Y (nx2236), .A0 (nx2228), .A1 (nx2745)) ;
    oai32 ix2229 (.Y (nx2228), .A0 (nx2741), .A1 (nx3610), .A2
(nx3594), .B0 (nx2591), .B1 (nx2595)) ;
    xnor2 ix2746 (.Y (nx2745), .A0 (nx2230), .A1 (nx2232)) ;
    xnor2 ix2752 (.Y (nx2751), .A0 (nx2196), .A1 (nx2252)) ;
    mux21_ni ix2197 (.Y (nx2196), .A0 (nx2132), .A1 (nx2072), .S0
(nx2134)) ;
    xnor2 ix2357 (.Y (nx2356), .A0 (nx2320), .A1 (nx2756)) ;
    mux21_ni ix2321 (.Y (nx2320), .A0 (nx2212), .A1 (nx2242), .S0
(nx2731)) ;
    xnor2 ix2757 (.Y (nx2756), .A0 (nx2350), .A1 (nx2352)) ;
    xnor2 ix2351 (.Y (nx2350), .A0 (nx2328), .A1 (nx2760)) ;
    mux21_ni ix2329 (.Y (nx2328), .A0 (nx2220), .A1 (nx2236), .S0
(nx2735)) ;
    xnor2 ix2761 (.Y (nx2760), .A0 (nx2344), .A1 (nx2346)) ;
    xnor2 ix2345 (.Y (nx2344), .A0 (nx2336), .A1 (nx2764)) ;
    mux21_ni ix2337 (.Y (nx2336), .A0 (nx2228), .A1 (nx2232), .S0
(nx2745)) ;
    xnor2 ix2765 (.Y (nx2764), .A0 (nx2338), .A1 (nx2340)) ;
    xnor2 ix2771 (.Y (nx2770), .A0 (nx2312), .A1 (nx2356)) ;
    mux21_ni ix2313 (.Y (nx2312), .A0 (nx2204), .A1 (nx2248), .S0
(nx2725)) ;
    xnor2 ix2445 (.Y (nx2444), .A0 (nx2424), .A1 (nx2775)) ;
    mux21_ni ix2425 (.Y (nx2424), .A0 (nx2328), .A1 (nx2346), .S0
(nx2760)) ;
    xnor2 ix2776 (.Y (nx2775), .A0 (nx2438), .A1 (nx2440)) ;
    xnor2 ix2439 (.Y (nx2438), .A0 (nx2432), .A1 (nx2779)) ;
    mux21_ni ix2433 (.Y (nx2432), .A0 (nx2336), .A1 (nx2340), .S0
(nx2764)) ;
    xnor2 ix2780 (.Y (nx2779), .A0 (nx876), .A1 (nx2434)) ;
    xnor2 ix2785 (.Y (nx2784), .A0 (nx2416), .A1 (nx2444)) ;
    mux21_ni ix2417 (.Y (nx2416), .A0 (nx2320), .A1 (nx2352), .S0
(nx2756)) ;
    xnor2 ix2521 (.Y (nx2520), .A0 (nx2512), .A1 (nx2789)) ;
    mux21_ni ix2513 (.Y (nx2512), .A0 (nx2432), .A1 (nx2434), .S0
(nx2779)) ;
    xnor2 ix2790 (.Y (nx2789), .A0 (nx2514), .A1 (nx2516)) ;
    xnor2 ix2794 (.Y (nx2793), .A0 (nx2504), .A1 (nx2520)) ;
    mux21_ni ix2505 (.Y (nx2504), .A0 (nx2424), .A1 (nx2440), .S0
(nx2775)) ;
    mux21_ni ix2581 (.Y (nx2580), .A0 (nx2512), .A1 (nx2516), .S0
(nx2789)) ;
    xnor2 ix2525 (.Y (nx2524), .A0 (nx2496), .A1 (nx2793)) ;
    xnor2 ix2449 (.Y (nx2448), .A0 (nx2408), .A1 (nx2784)) ;
    nand04 ix2705 (.Y (nx2704), .A0 (nx2801), .A1 (nx2803), .A2
(nx2807), .A3 (nx2809)) ;
    xnor2 ix2804 (.Y (nx2803), .A0 (nx2805), .A1 (nx2751)) ;
    mux21_ni ix2806 (.Y (nx2805), .A0 (nx2423), .A1 (nx2609), .S0
(nx2138)) ;
```

```
    xnor2 ix2810 (.Y (nx2809), .A0 (nx1934), .A1 (nx2008)) ;
    xor2 ix2812 (.Y (nx2811), .A0 (nx2666), .A1 (nx3686)) ;
    aoi21 ix2815 (.Y (nx2814), .A0 (nx1684), .A1 (nx874), .B0
(nx876)) ;
    aoi21 ix2818 (.Y (nx2817), .A0 (nx1344), .A1 (nx850), .B0
(nx852)) ;
    aoi21 ix2821 (.Y (nx2820), .A0 (nx1080), .A1 (nx830), .B0
(nx832)) ;
    aoi21 ix2824 (.Y (nx2823), .A0 (nx2825), .A1 (nx24), .B0(nx4)) ;
    inv01 ix2826 (.Y (nx2825), .A (NOT_src_cy)) ;
    inv02 ix2845 (.Y (nx2844), .A (NOT_src_1_7)) ;
    oai22 ix2695 (.Y (nx2694), .A0 (nx3558), .A1 (nx2847), .B0
(nx2902), .B1 (nx2910)) ;
    aoi22 ix2855 (.Y (nx2854), .A0 (nx3886), .A1 (nx1250), .B0
(nx3632), .B1 (nx1374)) ;
    aoi22 ix2862 (.Y (nx2861), .A0 (nx3848), .A1 (nx1014), .B0
(nx3648), .B1 (nx1118)) ;
    aoi22 ix2869 (.Y (nx2868), .A0 (nx3666), .A1 (nx3674), .B0
(NOT_src_cy), .B1 (nx46)) ;
    nand02 ix1007 (.Y (nx1006), .A0 (nx2868), .A1 (nx3600)) ;
    nand02 ix1243 (.Y (nx1242), .A0 (nx2861), .A1 (nx3588)) ;
    aoi22 ix2894 (.Y (nx2893), .A0 (nx3838), .A1 (nx1382), .B0
(nx3626), .B1 (nx1554)) ;
    nand02 ix1555 (.Y (nx1554), .A0 (nx2854), .A1 (nx3576)) ;
    aoi22 ix2897 (.Y (nx2896), .A0 (nx3844), .A1 (nx1736), .B0
(nx3612), .B1 (nx2640)) ;
    nand02 ix2903 (.Y (nx2902), .A0 (nx3656), .A1 (nx140)) ;
    nand02 ix2906 (.Y (nx2905), .A0 (nx3640), .A1 (nx132)) ;
    oai222 ix2681 (.Y (int_des_cy), .A0 (nx2914), .A1 (nx2916), .B0
(nx3606), .B1 (nx2941), .C0 (nx3564), .C1 (nx2943)) ;
    inv02 ix2915 (.Y (nx2914), .A (NOT_op_code_3)) ;
    aoi332 ix2917 (.Y (nx2916), .A0 (nx2658), .A1 (nx1494), .A2
(nx3768), .B0 (nx2896), .B1 (nx3554), .B2 (nx3854), .C0 (nx2666),
.C1 (nx34)) ;
    nand02 ix2659 (.Y (nx2658), .A0 (nx1492), .A1 (nx1480)) ;
    nor02_2x ix1493 (.Y (nx1492), .A0 (nx2920), .A1 (nx2928)) ;
    nand02 ix2923 (.Y (nx2922), .A0 (nx3692), .A1 (nx1468)) ;
    nand02 ix2926 (.Y (nx2925), .A0 (nx3886), .A1 (nx952)) ;
    aoi21 ix953 (.Y (nx952), .A0 (nx3594), .A1 (nx3600), .B0
(nx3588)) ;
    nor02ii ix2929 (.Y (nx2928), .A0 (nx1474), .A1 (nx2931)) ;
    aoi21 ix1475 (.Y (nx1474), .A0 (nx2925), .A1 (nx3576),
.B0(nx1468)) ;
    oai21 ix2932 (.Y (nx2931), .A0 (nx1468), .A1 (nx3692),
.B0(nx2922)) ;
    oai21 ix1481 (.Y (nx1480), .A0(nx3564),.A1(nx2922),
.B0(NOT_src_cy));
    inv02 ix2937 (.Y (nx2936), .A (NOT_op_code_1)) ;
    inv01 ix2940 (.Y (nx2939), .A (NOT_op_code_2)) ;
    oai221 ix2023 (.Y (int_des_2_0), .A0 (nx2809), .A1 (nx3706), .B0
(nx2950), .B1 (nx2974), .C0 (nx2976)) ;
    xor2 ix2951 (.Y (nx2950), .A0 (nx2952), .A1 (nx1922)) ;
    mux21_ni ix2953 (.Y (nx2952), .A0 (nx3564), .A1 (nx2954), .S0
(nx1866)) ;
    mux21_ni ix2955 (.Y (nx2954), .A0 (nx3570), .A1 (nx2956), .S0
(nx1674)) ;
```

```
    mux21_ni ix2957 (.Y (nx2956), .A0 (nx3576), .A1 (nx2958), .S0
(nx1516)) ;
    mux21_ni ix2959 (.Y (nx2958), .A0 (nx3582), .A1 (nx2960), .S0
(nx1334)) ;
    mux21_ni ix2961 (.Y (nx2960), .A0 (nx3588), .A1 (nx2962), .S0
(nx1200)) ;
    mux21_ni ix2963 (.Y (nx2962), .A0 (nx3594), .A1 (nx2964), .S0
(nx1070)) ;
    aoi32 ix2965 (.Y (nx2964), .A0 (int_src_3_0), .A1 (nx3676), .A2
(nx968), .B0 (nx3682), .B1 (int_src_3_1)) ;
    nand02_2x ix2975 (.Y (nx2974), .A0 (nx2914), .A1
(NOT_op_code_0)) ;
    aoi322 ix2977 (.Y (nx2976), .A0 (nx1902), .A1 (nx2979), .A2
(nx3534), .B0 (nx3676), .B1 (nx1914), .C0 (nx3165), .C1 (nx788)) ;
    nand02 ix1903 (.Y (nx1902), .A0 (nx2952), .A1 (nx3666)) ;
    nor02_2x ix79 (.Y (nx78), .A0 (nx3786), .A1 (NOT_op_code_0)) ;
    nor02_2x ix1915 (.Y (nx1914), .A0 (nx786), .A1 (nx3163)) ;
    mux21_ni ix787 (.Y (nx786), .A0 (nx3612), .A1 (nx782), .S0
(nx638)) ;
    mux21_ni ix783 (.Y (nx782), .A0 (nx778), .A1 (nx3620),
.S0(nx3155)) ;
    mux21_ni ix779 (.Y (nx778), .A0 (nx774), .A1 (nx3628),
.S0(nx3151)) ;
    mux21_ni ix775 (.Y (nx774), .A0 (nx3632), .A1 (nx770), .S0
(nx692)) ;
    mux21_ni ix771 (.Y (nx770), .A0 (nx766), .A1 (nx3642),
.S0(nx3142)) ;
    mux21_ni ix767 (.Y (nx766), .A0 (nx3648), .A1 (nx762), .S0
(nx728)) ;
    mux21_ni ix763 (.Y (nx762), .A0 (nx758), .A1 (nx3656),
.S0(nx2996)) ;
    oai22 ix749 (.Y (nx748), .A0 (nx3600), .A1 (nx2999), .B0
(nx594), .B1 (nx3710)) ;
    mux21_ni ix3002 (.Y (nx3001), .A0 (nx3003), .A1 (nx3130), .S0
(nx3132)) ;
    nand02 ix3004 (.Y (nx3003), .A0 (nx3612), .A1 (nx614)) ;
    mux21_ni ix615 (.Y (nx614), .A0 (nx610), .A1 (nx3626),
.S0(nx3126)) ;
    mux21_ni ix611 (.Y (nx610), .A0 (nx606), .A1 (nx3634),
.S0(nx3122)) ;
    mux21_ni ix607 (.Y (nx606), .A0 (nx3640), .A1 (nx602), .S0
(nx550)) ;
    mux21_ni ix603 (.Y (nx602), .A0 (nx598), .A1 (nx3650),
.S0(nx3113)) ;
    mux21_ni ix599 (.Y (nx598), .A0 (nx3656), .A1 (nx594), .S0
(nx584)) ;
    nand02 ix581 (.Y (nx580), .A0 (nx3546), .A1 (nx3890)) ;
    oai21 ix491 (.Y (nx490), .A0 (nx480), .A1 (nx2908), .B0
(nx3088)) ;
    aoi32 ix3018 (.Y (nx3017), .A0 (nx366), .A1 (nx3078),
.A2(nx132), .B0 (nx298), .B1 (nx3720)) ;
    mux21_ni ix367 (.Y (nx366), .A0 (nx362), .A1 (nx3640),
.S0(nx3074)) ;
    mux21_ni ix363 (.Y (nx362), .A0 (nx358), .A1 (nx3650),
.S0(nx3070)) ;
```

```
    mux21_ni ix359 (.Y (nx358), .A0 (nx3656), .A1 (nx354), .S0
(nx344)) ;
    nand02 ix341 (.Y (nx340), .A0 (nx3544), .A1 (nx3700)) ;
    oai21 ix289 (.Y (nx288), .A0 (nx278), .A1 (nx2905), .B0
(nx3056)) ;
    aoi32 ix3030 (.Y (nx3029), .A0 (nx208), .A1 (nx3040),
.A2(nx140), .B0 (nx184), .B1 (nx3718)) ;
    mux21_ni ix209 (.Y (nx208), .A0 (nx204), .A1 (nx3658),
.S0(nx3033)) ;
    oai32 ix185 (.Y (nx184), .A0 (nx3043), .A1 (nx164), .A2
(nx2902), .B0 (nx3046), .B1 (nx174)) ;
    oai21 ix3047 (.Y (nx3046), .A0 (nx2902), .A1 (nx3668),
.B0(nx3844)) ;
    aoi21 ix3052 (.Y (nx3051), .A0 (nx3040), .A1 (nx140), .B0
(nx214)) ;
    nor03_2x ix215 (.Y (nx214), .A0 (nx2905), .A1
(nx3054),.A2(nx3040)) ;
    nand03 ix3057 (.Y (nx3056), .A0 (nx132), .A1 (nx276), .A2
(nx278)) ;
    mux21_ni ix277 (.Y (nx276), .A0 (nx272), .A1 (nx3648),
.S0(nx3066)) ;
    mux21_ni ix273 (.Y (nx272), .A0 (nx268), .A1 (nx3658),
.S0(nx3061)) ;
    xnor2 ix237 (.Y (nx236), .A0 (nx204), .A1 (nx3033)) ;
    mux21_ni ix327 (.Y (nx326), .A0 (nx256), .A1 (nx322), .S0
(nx3544)) ;
    xnor2 ix323 (.Y (nx322), .A0 (nx268), .A1 (nx3061)) ;
    mux21_ni ix309 (.Y (nx308), .A0 (nx240), .A1 (nx304), .S0
(nx3544)) ;
    xnor2 ix305 (.Y (nx304), .A0 (nx272), .A1 (nx3066)) ;
    oai32 ix299 (.Y (nx298), .A0 (nx3081), .A1 (nx278), .A2
(nx2905), .B0 (nx3716), .B1 (nx3544)) ;
    aoi21 ix3084 (.Y (nx3083), .A0 (nx3078), .A1 (nx132), .B0
(nx372)) ;
    nor03_2x ix373 (.Y (nx372), .A0 (nx2908), .A1
(nx3086),.A2(nx3078)) ;
    nand04 ix3089 (.Y (nx3088), .A0 (nx3614), .A1 (nx3618), .A2
(nx478), .A3 (nx480)) ;
    mux21_ni ix479 (.Y (nx478), .A0 (nx474), .A1 (nx3632),
.S0(nx3109)) ;
    mux21_ni ix475 (.Y (nx474), .A0 (nx470), .A1 (nx3642),
.S0(nx3105)) ;
    mux21_ni ix471 (.Y (nx470), .A0 (nx3650), .A1 (nx466), .S0
(nx436)) ;
    mux21_ni ix467 (.Y (nx466), .A0 (nx462), .A1 (nx3660),
.S0(nx3095)) ;
    mux21_ni ix3102 (.Y (nx3101), .A0 (nx3103), .A1 (nx3024), .S0
(nx3720)) ;
    xnor2 ix3104 (.Y (nx3103), .A0 (nx354), .A1 (nx344)) ;
    mux21_ni ix417 (.Y (nx416), .A0 (nx412), .A1 (nx326), .S0
(nx3720)) ;
    xnor2 ix413 (.Y (nx412), .A0 (nx358), .A1 (nx3070)) ;
    mux21_ni ix399 (.Y (nx398), .A0 (nx394), .A1 (nx308), .S0
(nx3720)) ;
    xnor2 ix395 (.Y (nx394), .A0 (nx362), .A1 (nx3074)) ;
```

```
    mux21_ni ix567 (.Y (nx566), .A0 (nx450), .A1 (nx562), .S0
(nx3546)) ;
    xnor2 ix563 (.Y (nx562), .A0 (nx462), .A1 (nx3095)) ;
    mux21_ni ix3119 (.Y (nx3118), .A0 (nx3101), .A1 (nx3120), .S0
(nx3546)) ;
    xnor2 ix3121 (.Y (nx3120), .A0 (nx466), .A1 (nx436)) ;
    mux21_ni ix531 (.Y (nx530), .A0 (nx416), .A1 (nx526), .S0
(nx3546)) ;
    xnor2 ix527 (.Y (nx526), .A0 (nx470), .A1 (nx3105)) ;
    mux21_ni ix513 (.Y (nx512), .A0 (nx398), .A1 (nx508), .S0
(nx3546)) ;
    xnor2 ix509 (.Y (nx508), .A0 (nx474), .A1 (nx3109)) ;
    oai32 ix501 (.Y (nx500), .A0 (nx3135), .A1 (nx480), .A2
(nx2908), .B0 (nx3714), .B1 (nx3546)) ;
    xnor2 ix3141 (.Y (nx3140), .A0 (nx594), .A1 (nx584)) ;
    xnor2 ix705 (.Y (nx704), .A0 (nx598), .A1 (nx3113)) ;
    xnor2 ix3150 (.Y (nx3149), .A0 (nx602), .A1 (nx550)) ;
    xnor2 ix669 (.Y (nx668), .A0 (nx606), .A1 (nx3122)) ;
    xnor2 ix651 (.Y (nx650), .A0 (nx610), .A1 (nx3126)) ;
    aoi32 ix3161 (.Y (nx3160), .A0 (nx614), .A1 (nx3132), .A2
(nx124), .B0 (nx500), .B1 (nx3712)) ;
    nand02 ix3164 (.Y (nx3163), .A0 (nx3786), .A1 (NOT_op_code_0)) ;
    oai21 ix3166 (.Y (nx3165), .A0 (nx3890), .A1 (nx3606), .B0
(nx46)) ;
    nand02 ix2147 (.Y (int_des_2_1), .A0 (nx3169), .A1 (nx3178)) ;
    aoi22 ix3170 (.Y (nx3169), .A0 (nx2054), .A1 (nx74), .B0
(nx2140), .B1 (nx3524)) ;
    xnor2 ix2055 (.Y (nx2054), .A0 (nx3172), .A1 (nx2052)) ;
    aoi22 ix3173 (.Y (nx3172), .A0 (nx3890), .A1 (nx3482), .B0
(nx1900), .B1 (nx1922)) ;
    xnor2 ix2141 (.Y (nx2140), .A0 (nx2609), .A1 (nx2138)) ;
    aoi322 ix3179 (.Y (nx3178), .A0 (nx2028), .A1 (nx3181), .A2
(nx3534), .B0 (nx748), .B1 (nx1914), .C0 (nx2024), .C1 (nx788)) ;
    nand02 ix3182 (.Y (nx3181), .A0 (nx3880), .A1 (nx1904)) ;
    xnor2 ix2025 (.Y (nx2024), .A0 (nx758), .A1 (nx2996)) ;
    oai221 ix2263 (.Y (int_des_2_2), .A0 (nx2803), .A1 (nx3706), .B0
(nx3186), .B1 (nx2974), .C0 (nx3191)) ;
    xor2 ix3187 (.Y (nx3186), .A0 (nx3188), .A1 (nx2176)) ;
    mux21_ni ix3189 (.Y (nx3188), .A0 (nx3660), .A1 (nx3172), .S0
(nx2052)) ;
    aoi322 ix3192 (.Y (nx3191), .A0 (nx2152), .A1 (nx3194), .A2
(nx3534), .B0 (nx726), .B1 (nx1914), .C0 (nx2148), .C1 (nx788)) ;
    nand02 ix2153 (.Y (nx2152), .A0 (nx3181), .A1 (nx3652)) ;
    xor2 ix2149 (.Y (nx2148), .A0 (nx762), .A1 (nx728)) ;
    nand02 ix2367 (.Y (int_des_2_3), .A0 (nx3199), .A1 (nx3206)) ;
    aoi22 ix3200 (.Y (nx3199), .A0 (nx2294), .A1 (nx74), .B0
(nx2360), .B1 (nx3524)) ;
    xnor2 ix2295 (.Y (nx2294), .A0 (nx3202), .A1 (nx2292)) ;
    mux21_ni ix3203 (.Y (nx3202), .A0 (nx3652), .A1 (nx3188), .S0
(nx2176)) ;
    xnor2 ix2361 (.Y (nx2360), .A0 (nx2304), .A1 (nx2770)) ;
    aoi322 ix3207 (.Y (nx3206), .A0 (nx2268), .A1 (nx3209), .A2
(nx3534), .B0 (nx708), .B1 (nx1914), .C0 (nx2264), .C1 (nx788)) ;
    nand02 ix3210 (.Y (nx3209), .A0 (nx3870), .A1 (nx2154)) ;
    xnor2 ix2265 (.Y (nx2264), .A0 (nx766), .A1 (nx3142)) ;
```

```
    ao221 ix2455 (.Y (int_des_2_4), .A0 (nx2448), .A1 (nx3524), .B0
(nx2398), .B1 (nx74), .C0 (nx2386)) ;
    xnor2 ix2399 (.Y (nx2398), .A0 (nx3215), .A1 (nx2396)) ;
    mux21_ni ix3216 (.Y (nx3215), .A0 (nx3644), .A1 (nx3202), .S0
(nx2292)) ;
    oai322 ix2387 (.Y (nx2386), .A0 (nx3219), .A1 (nx2374), .A2
(nx3222), .B0 (nx3147), .B1 (nx3224), .C0 (nx3226), .C1 (nx3228)) ;
    xnor2 ix3227 (.Y (nx3226), .A0 (nx770), .A1 (nx692)) ;
    nand03 ix3229 (.Y (nx3228), .A0 (nx786), .A1 (nx3786), .A2
(NOT_op_code_0)       ) ;
    nand02 ix2531 (.Y (int_des_2_5), .A0 (nx3231), .A1 (nx3237)) ;
    aoi22 ix3232 (.Y (nx3231), .A0 (nx2486), .A1 (nx74), .B0
(nx2524), .B1 (nx3524)) ;
    xnor2 ix2487 (.Y (nx2486), .A0 (nx3234), .A1 (nx2484)) ;
    mux21_ni ix3235 (.Y (nx3234), .A0 (nx3634), .A1 (nx3215), .S0
(nx2396)) ;
    aoi322 ix3238 (.Y (nx3237), .A0 (nx2460), .A1 (nx3240), .A2
(nx3534), .B0 (nx672), .B1 (nx1914), .C0 (nx2456), .C1 (nx788)) ;
    oai21 ix2461 (.Y (nx2460), .A0 (nx3636), .A1 (nx3209), .B0
(nx3628)) ;
    nand02 ix3241 (.Y (nx3240), .A0 (nx3802), .A1 (nx2374)) ;
    xnor2 ix2457 (.Y (nx2456), .A0 (nx774), .A1 (nx3151)) ;
    nand02 ix2591 (.Y (int_des_2_6), .A0 (nx3244), .A1 (nx3253)) ;
    aoi22 ix3245 (.Y (nx3244), .A0 (nx2562), .A1 (nx74), .B0
(nx2584), .B1 (nx3524)) ;
    xnor2 ix2563 (.Y (nx2562), .A0 (nx3247), .A1 (nx2560)) ;
    mux21_ni ix3248 (.Y (nx3247), .A0 (nx3628), .A1 (nx3234), .S0
(nx2484)) ;
    xnor2 ix2585 (.Y (nx2584), .A0 (nx2572), .A1 (nx3251)) ;
    xnor2 ix3252 (.Y (nx3251), .A0 (nx890), .A1 (nx2580)) ;
    aoi322 ix3254 (.Y (nx3253), .A0 (nx2536), .A1 (nx3256), .A2
(nx3534), .B0 (nx654), .B1 (nx1914), .C0 (nx2532), .C1 (nx788)) ;
    nand02 ix2537 (.Y (nx2536), .A0 (nx3240), .A1 (nx3620)) ;
    xnor2 ix2533 (.Y (nx2532), .A0 (nx778), .A1 (nx3155)) ;
    ao221 ix2633 (.Y (int_des_2_7), .A0 (nx2626), .A1 (nx3524), .B0
(nx2616), .B1 (nx3534), .C0 (nx2614)) ;
    mux21_ni ix2627 (.Y (nx2626), .A0 (nx2572), .A1 (nx2580), .S0
(nx3251)) ;
    oai221 ix2615 (.Y (nx2614), .A0 (nx3722), .A1 (nx3224), .B0
(nx3264), .B1 (nx2974), .C0 (nx3270)) ;
    xnor2 ix3265 (.Y (nx3264), .A0 (nx3266), .A1 (nx3268)) ;
    mux21_ni ix3267 (.Y (nx3266), .A0 (nx3620), .A1 (nx3247), .S0
(nx2560)) ;
    nand04 ix801 (.Y (int_des_1_0), .A0 (nx3276), .A1 (nx3281), .A2
(nx3290), .A3 (nx3297)) ;
    aoi22 ix3277 (.Y (nx3276), .A0 (nx3676), .A1 (nx792), .B0
(nx3768), .B1 (nx788)) ;
    aoi222 ix3282 (.Y (nx3281), .A0 (nx24), .A1 (nx3862), .B0
(nx3606), .B1 (nx3538), .C0 (nx100), .C1 (nx3860)) ;
    nor03_2x ix107 (.Y (nx106), .A0 (nx3163), .A1 (nx3554), .A2
(nx3558)) ;
    xnor2 ix101 (.Y (nx100), .A0 (NOT_src_cy), .A1 (nx3530)) ;
    aoi21 ix29 (.Y (nx28), .A0 (nx3608), .A1 (nx3668), .B0 (nx4)) ;
    aoi332 ix3291 (.Y (nx3290), .A0 (nx3682), .A1 (nx2914), .A2
(nx3768), .B0 (nx3554), .B1 (nx3854), .B2 (nx82), .C0 (nx56), .C1
(nx3532)) ;
```

```
    oai22 ix83 (.Y (nx82), .A0 (NOT_src_cy), .A1 (nx3222), .B0
(nx3566), .B1 (nx2974)) ;
    xnor2 ix57 (.Y (nx56), .A0 (NOT_src_cy), .A1 (nx3165)) ;
    nor02_2x ix69 (.Y (nx68), .A0 (nx3163), .A1 (nx3295)) ;
    aoi332 ix3298 (.Y (nx3297), .A0 (nx3524), .A1 (nx3854), .A2
(nx4), .B0 (nx3530), .B1 (nx2914), .B2 (nx34), .C0 (nx10), .C1
(nx20)) ;
    or02 ix11 (.Y (nx10), .A0 (nx3676), .A1 (int_src_3_0)) ;
    aoi21 ix21 (.Y (nx20), .A0 (int_src_3_0), .A1 (nx3676), .B0
(nx3301)) ;
    nand04 ix981 (.Y (int_des_1_1), .A0 (nx3305), .A1 (nx3315), .A2
(nx3319), .A3 (nx3326)) ;
    aoi222 ix3306 (.Y (nx3305), .A0 (nx970), .A1 (nx3528), .B0
(nx964), .B1 (nx3860), .C0 (nx956), .C1 (nx792)) ;
    xnor2 ix971 (.Y (nx970), .A0 (nx3308), .A1 (nx968)) ;
    nand02 ix3309 (.Y (nx3308), .A0 (int_src_3_0), .A1 (nx3676)) ;
    aoi222 ix3316 (.Y (nx3315), .A0 (nx3676), .A1 (nx3552), .B0
(nx3602), .B1 (nx3538), .C0 (nx824), .C1 (nx3862)) ;
    nor02_2x ix939 (.Y (nx938), .A0 (nx3786), .A1 (nx3295)) ;
    aoi222 ix3320 (.Y (nx3319), .A0 (nx3548), .A1 (nx3550), .B0
(nx3848), .B1 (nx3770), .C0 (nx920), .C1 (nx3532)) ;
    nor02_2x ix927 (.Y (nx926), .A0 (nx3284), .A1 (nx2974)) ;
    xnor2 ix921 (.Y (nx920), .A0 (nx2868), .A1 (nx3856)) ;
    aoi332 ix3327 (.Y (nx3326), .A0 (nx812), .A1 (nx3329), .A2
(nx3778), .B0 (nx3880), .B1 (nx3876), .B2 (nx3776), .C0 (nx3772),
.C1 (nx906)) ;
    nand02 ix3330 (.Y (nx3329), .A0 (nx4), .A1 (nx802)) ;
    nor04 ix3336 (.Y (nx3335), .A0 (nx3530), .A1 (nx3724), .A2
(nx3858), .A3 (nx3726)) ;
    aoi21 ix3340 (.Y (nx3339), .A0 (nx3570), .A1 (nx3622), .B0
(nx876)) ;
    nand04 ix1093 (.Y (int_des_1_2), .A0 (nx3354), .A1 (nx3359), .A2
(nx3362), .A3 (nx3365)) ;
    aoi222 ix3355 (.Y (nx3354), .A0 (nx830), .A1 (nx3862), .B0
(nx1082), .B1 (nx3860), .C0 (nx1072), .C1 (nx3528)) ;
    xnor2 ix1083 (.Y (nx1082), .A0 (nx1080), .A1 (nx3780)) ;
    xnor2 ix1073 (.Y (nx1072), .A0 (nx2964), .A1 (nx1070)) ;
    aoi222 ix3360 (.Y (nx3359), .A0 (nx3876), .A1 (nx3552), .B0
(nx3596), .B1 (nx3538), .C0 (nx1054), .C1 (nx3778)) ;
    xnor2 ix1055 (.Y (nx1054), .A0 (nx3329), .A1 (nx1052)) ;
    aoi222 ix3363 (.Y (nx3362), .A0 (nx3732), .A1 (nx3550), .B0
(nx3828), .B1 (nx3770), .C0 (nx1016), .C1 (nx3532)) ;
    aoi222 ix3366 (.Y (nx3365), .A0 (nx792), .A1 (nx996), .B0
(nx832), .B1 (nx3776), .C0 (nx3546), .C1 (nx906)) ;
    oai22 ix997 (.Y (nx996), .A0 (nx3596), .A1 (nx3368), .B0
(nx950), .B1 (nx3370)) ;
    nor02_2x ix3371 (.Y (nx3370), .A0 (nx952), .A1 (int_src_ac)) ;
    nand04 ix1223 (.Y (int_des_1_3), .A0 (nx3374), .A1 (nx3379), .A2
(nx3382), .A3 (nx3386)) ;
    aoi222 ix3375 (.Y (nx3374), .A0 (nx842), .A1 (nx3862), .B0
(nx1212), .B1 (nx3860), .C0 (nx1202), .C1 (nx3528)) ;
    xnor2 ix1203 (.Y (nx1202), .A0 (nx2962), .A1 (nx1200)) ;
    aoi222 ix3380 (.Y (nx3379), .A0 (nx3848), .A1 (nx3552), .B0
(nx3590), .B1 (nx3538), .C0 (nx1184), .C1 (nx3778)) ;
    xor2 ix1185 (.Y (nx1184), .A0 (nx1156), .A1 (nx1182)) ;
```

```
    aoi222 ix3383 (.Y (nx3382), .A0 (nx3730), .A1 (nx3550), .B0
(nx3886), .B1 (nx3770), .C0 (nx1128), .C1 (nx3532)) ;
    xnor2 ix1129 (.Y (nx1128), .A0 (nx2861), .A1 (nx3782)) ;
    aoi21 ix3387 (.Y (nx3386), .A0 (nx844), .A1 (nx3776), .B0
(nx1114)) ;
    oai22 ix1115 (.Y (nx1114), .A0 (nx3720), .A1 (nx3389), .B0
(nx3392), .B1 (nx3394)) ;
    nand04 ix3390 (.Y (nx3389), .A0 (nx902), .A1 (nx3786), .A2
(NOT_op_code_0),.A3 (nx3768)) ;
    nand02 ix903 (.Y (nx902), .A0 (nx3335), .A1 (nx3343)) ;
    nand02 ix3393 (.Y (nx3392), .A0 (nx3526), .A1 (nx3768)) ;
    aoi32 ix3395 (.Y (nx3394), .A0 (nx1096), .A1 (nx3397), .A2
(int_src_ac), .B0 (nx3828), .B1 (nx3370)) ;
    nand04 ix1357 (.Y (int_des_1_4), .A0 (nx3400), .A1 (nx3405), .A2
(nx3408), .A3 (nx3411)) ;
    aoi222 ix3401 (.Y (nx3400), .A0 (nx850), .A1 (nx3862), .B0
(nx1346), .B1 (nx3860), .C0 (nx1336), .C1 (nx3528)) ;
    xnor2 ix1347 (.Y (nx1346), .A0 (nx1344), .A1 (nx3784)) ;
    xnor2 ix1337 (.Y (nx1336), .A0 (nx2960), .A1 (nx1334)) ;
    aoi222 ix3406 (.Y (nx3405), .A0 (nx3828), .A1 (nx3552), .B0
(nx3584), .B1 (nx3538), .C0 (nx1318), .C1 (nx3778)) ;
    xnor2 ix1319 (.Y (nx1318), .A0 (nx1280), .A1 (nx2655)) ;
    aoi222 ix3409 (.Y (nx3408), .A0 (nx3728), .A1 (nx3550), .B0
(nx3838), .B1 (nx3770), .C0 (nx1252), .C1 (nx3532)) ;
    aoi332 ix3412 (.Y (nx3411), .A0 (nx1226), .A1 (nx2925), .A2
(nx792), .B0 (nx3864), .B1 (nx3886), .B2 (nx3776), .C0 (nx3544), .C1
(nx906)) ;
    nand04 ix1529 (.Y (int_des_1_5), .A0 (nx3415), .A1 (nx3422), .A2
(nx3426), .A3 (nx3430)) ;
    aoi222 ix3416 (.Y (nx3415), .A0 (nx1496), .A1 (nx792), .B0
(nx1504), .B1 (nx3860), .C0 (nx1518), .C1 (nx3528)) ;
    xnor2 ix1497 (.Y (nx1496), .A0 (nx1474), .A1 (nx3418)) ;
    nor02_2x ix3419 (.Y (nx3418), .A0 (nx1492), .A1 (nx1480)) ;
    xnor2 ix1519 (.Y (nx1518), .A0 (nx2958), .A1 (nx1516)) ;
    aoi222 ix3423 (.Y (nx3422), .A0 (nx3578), .A1 (nx3538), .B0
(nx866), .B1 (nx3862), .C0 (nx1456), .C1 (nx3778)) ;
    xnor2 ix1457 (.Y (nx1456), .A0 (nx1402), .A1 (nx2671)) ;
    aoi222 ix3427 (.Y (nx3426), .A0 (nx3726), .A1 (nx3550), .B0
(nx3886), .B1 (nx3552), .C0 (nx1384), .C1 (nx3532)) ;
    xnor2 ix1385 (.Y (nx1384), .A0 (nx2854), .A1 (nx872)) ;
    aoi222 ix3431 (.Y (nx3430), .A0 (nx868), .A1 (nx3776), .B0
(nx3692), .B1 (nx3770), .C0 (nx220), .C1 (nx906)) ;
    nand04 ix1697 (.Y (int_des_1_6), .A0 (nx3435), .A1 (nx3440), .A2
(nx3443), .A3 (nx3446)) ;
    aoi222 ix3436 (.Y (nx3435), .A0 (nx1686), .A1 (nx102), .B0
(nx874), .B1 (nx112), .C0 (nx1676), .C1 (nx3528)) ;
    xnor2 ix1677 (.Y (nx1676), .A0 (nx2956), .A1 (nx1674)) ;
    aoi222 ix3441 (.Y (nx3440), .A0 (nx3838), .A1 (nx3552), .B0
(nx3572), .B1 (nx3538), .C0 (nx1658), .C1 (nx3778)) ;
    xnor2 ix1659 (.Y (nx1658), .A0 (nx1592), .A1 (nx2684)) ;
    aoi222 ix3444 (.Y (nx3443), .A0 (nx3858), .A1 (nx3550), .B0
(nx3844), .B1 (nx3770), .C0 (nx1564), .C1 (nx68)) ;
    aoi222 ix3447 (.Y (nx3446), .A0 (nx174), .A1 (nx906), .B0
(nx876), .B1 (nx3776), .C0 (nx792), .C1 (nx1544)) ;
    oai22 ix1545 (.Y (nx1544), .A0 (nx2931), .A1 (nx3449), .B0
(nx1490), .B1 (nx3418)) ;
```

```
    nor02_2x ix3450 (.Y (nx3449), .A0 (nx1474), .A1 (nx3418)) ;
    nand04 ix1889 (.Y (int_des_1_7), .A0 (nx3455), .A1 (nx3460), .A2
(nx3463), .A3 (nx3467)) ;
    aoi222 ix3456 (.Y (nx3455), .A0 (nx1878), .A1 (nx102), .B0
(nx888), .B1 (nx112), .C0 (nx1868), .C1 (nx3528)) ;
    xnor2 ix1869 (.Y (nx1868), .A0 (nx2954), .A1 (nx1866)) ;
    aoi222 ix3461 (.Y (nx3460), .A0 (nx3692), .A1 (nx3552), .B0
(nx3566), .B1 (nx3540), .C0 (nx1850), .C1 (nx3778)) ;
    xnor2 ix1851 (.Y (nx1850), .A0 (nx1766), .A1 (nx2697)) ;
    aoi222 ix3464 (.Y (nx3463), .A0 (nx3724), .A1 (nx3550), .B0
(nx890), .B1 (nx3776), .C0 (nx1738), .C1 (nx68)) ;
    aoi222 ix3468 (.Y (nx3467), .A0 (nx3768), .A1 (nx1720), .B0
(nx150), .B1 (nx906), .C0 (nx792), .C1 (nx1712)) ;
    oai22 ix1721 (.Y (nx1720), .A0 (NOT_src_cy), .A1 (nx3222), .B0
(nx3608), .B1 (nx2974)) ;
    aoi21 ix151 (.Y (nx150), .A0 (nx3566), .A1 (nx3890), .B0
(nx2902)) ;
    oai32 ix1713 (.Y (nx1712), .A0 (nx3472), .A1 (nx1492), .A2
(nx3474), .B0 (nx2920), .B1 (nx1494)) ;
    and02 ix3473 (.Y (nx3472), .A0 (nx2928), .A1 (nx2920)) ;
    inv01 ix3257 (.Y (nx3256), .A (nx2538)) ;
    inv01 ix2802 (.Y (nx2801), .A (nx2360)) ;
    inv01 ix2189 (.Y (nx2188), .A (nx2805)) ;
    inv01 ix3195 (.Y (nx3194), .A (nx2154)) ;
    inv01 ix2808 (.Y (nx2807), .A (nx2140)) ;
    inv01 ix2742 (.Y (nx2741), .A (nx2108)) ;
    inv01 ix3225 (.Y (nx3224), .A (nx1914)) ;
    inv01 ix2980 (.Y (nx2979), .A (nx1904)) ;
    inv01 ix1901 (.Y (nx1900), .A (nx2952)) ;
    inv01 ix1495 (.Y (nx1494), .A (nx3418)) ;
    inv01 ix1491 (.Y (nx1490), .A (nx2928)) ;
    inv01 ix3475 (.Y (nx3474), .A (nx1480)) ;
    inv01 ix1383 (.Y (nx1382), .A (nx2854)) ;
    inv01 ix2882 (.Y (nx2881), .A (nx1250)) ;
    inv01 ix1041 (.Y (nx1040), .A (nx2637)) ;
    inv01 ix3398 (.Y (nx3397), .A (nx952)) ;
    inv01 ix2944 (.Y (nx2943), .A (nx938)) ;
    inv02 ix907 (.Y (nx906), .A (nx3389)) ;
    inv01 ix815 (.Y (nx814), .A (nx3329)) ;
    inv02 ix793 (.Y (nx792), .A (nx3392)) ;
    inv02 ix789 (.Y (nx788), .A (nx3228)) ;
    inv01 ix727 (.Y (nx726), .A (nx3138)) ;
    inv01 ix3136 (.Y (nx3135), .A (nx478)) ;
    inv01 ix3087 (.Y (nx3086), .A (nx366)) ;
    inv01 ix3082 (.Y (nx3081), .A (nx276)) ;
    inv01 ix221 (.Y (nx220), .A (nx3051)) ;
    inv01 ix3055 (.Y (nx3054), .A (nx208)) ;
    inv01 ix175 (.Y (nx174), .A (nx3038)) ;
    inv01 ix125 (.Y (nx124), .A (nx3130)) ;
    inv02 ix3223 (.Y (nx3222), .A (nx78)) ;
    inv01 ix75 (.Y (nx74), .A (nx2974)) ;
    inv01 ix35 (.Y (nx34), .A (nx3284)) ;
    buf02 ix3481 (.Y (nx3482), .A (int_src_3_7)) ;
    buf02 ix3483 (.Y (nx3484), .A (int_src_3_7)) ;
    buf04 ix3485 (.Y (nx3486), .A (int_des_1_7)) ;
    buf04 ix3487 (.Y (nx3488), .A (int_des_1_6)) ;
```

```
buf04 ix3489 (.Y (nx3490), .A (int_des_1_5)) ;
buf04 ix3491 (.Y (nx3492), .A (int_des_1_4)) ;
buf04 ix3493 (.Y (nx3494), .A (int_des_1_3)) ;
buf04 ix3495 (.Y (nx3496), .A (int_des_1_2)) ;
buf04 ix3497 (.Y (nx3498), .A (int_des_1_1)) ;
buf04 ix3499 (.Y (nx3500), .A (int_des_1_0)) ;
buf04 ix3501 (.Y (nx3502), .A (int_des_2_7)) ;
buf04 ix3503 (.Y (nx3504), .A (int_des_2_6)) ;
buf04 ix3505 (.Y (nx3506), .A (int_des_2_5)) ;
buf04 ix3507 (.Y (nx3508), .A (int_des_2_4)) ;
buf04 ix3509 (.Y (nx3510), .A (int_des_2_3)) ;
buf04 ix3511 (.Y (nx3512), .A (int_des_2_2)) ;
buf04 ix3513 (.Y (nx3514), .A (int_des_2_1)) ;
buf04 ix3515 (.Y (nx3516), .A (int_des_2_0)) ;
buf04 ix3517 (.Y (nx3518), .A (int_des_cy)) ;
buf04 ix3519 (.Y (nx3520), .A (int_des_ac)) ;
buf04 ix3521 (.Y (nx3522), .A (int_des_ov)) ;
inv02 ix3523 (.Y (nx3524), .A (nx3706)) ;
inv02 ix3525 (.Y (nx3526), .A (nx3706)) ;
inv02 ix3527 (.Y (nx3528), .A (nx3301)) ;
buf02 ix3529 (.Y (nx3530), .A (nx28)) ;
nor02_2x ix3531 (.Y (nx3532), .A0 (nx3163), .A1 (nx3295)) ;
inv02 ix3533 (.Y (nx3534), .A (nx3222)) ;
buf02 ix3537 (.Y (nx3538), .A (nx106)) ;
buf02 ix3539 (.Y (nx3540), .A (nx106)) ;
buf02 ix3543 (.Y (nx3544), .A (nx288)) ;
buf02 ix3545 (.Y (nx3546), .A (nx490)) ;
inv01 ix3547 (.Y (nx3548), .A (nx3324)) ;
buf02 ix3549 (.Y (nx3550), .A (nx926)) ;
inv02 ix3551 (.Y (nx3552), .A (nx2943)) ;
inv02 ix3553 (.Y (nx3554), .A (nx2939)) ;
inv02 ix3557 (.Y (nx3558), .A (nx3854)) ;
inv02 ix3563 (.Y (nx3564), .A (nx3844)) ;
inv02 ix3565 (.Y (nx3566), .A (nx3844)) ;
inv02 ix3569 (.Y (nx3570), .A (nx3884)) ;
inv02 ix3571 (.Y (nx3572), .A (nx3884)) ;
inv02 ix3575 (.Y (nx3576), .A (nx3838)) ;
inv02 ix3577 (.Y (nx3578), .A (nx3838)) ;
inv02 ix3581 (.Y (nx3582), .A (nx3886)) ;
inv02 ix3583 (.Y (nx3584), .A (nx3698)) ;
inv02 ix3587 (.Y (nx3588), .A (nx3828)) ;
inv02 ix3589 (.Y (nx3590), .A (nx3828)) ;
inv02 ix3593 (.Y (nx3594), .A (nx3848)) ;
inv02 ix3595 (.Y (nx3596), .A (nx3848)) ;
inv02 ix3599 (.Y (nx3600), .A (nx3876)) ;
inv02 ix3601 (.Y (nx3602), .A (nx3876)) ;
inv02 ix3605 (.Y (nx3606), .A (nx3868)) ;
inv02 ix3607 (.Y (nx3608), .A (nx3868)) ;
inv02 ix3609 (.Y (nx3610), .A (nx3816)) ;
inv02 ix3611 (.Y (nx3612), .A (nx3816)) ;
inv02 ix3613 (.Y (nx3614), .A (nx3816)) ;
inv02 ix3617 (.Y (nx3618), .A (nx3808)) ;
inv02 ix3619 (.Y (nx3620), .A (nx3808)) ;
inv02 ix3621 (.Y (nx3622), .A (nx3808)) ;
inv02 ix3625 (.Y (nx3626), .A (nx3802)) ;
inv02 ix3627 (.Y (nx3628), .A (nx3802)) ;
```

```
    inv02 ix3631 (.Y (nx3632), .A (nx3794)) ;
    inv02 ix3633 (.Y (nx3634), .A (nx3794)) ;
    inv02 ix3635 (.Y (nx3636), .A (nx3794)) ;
    inv02 ix3639 (.Y (nx3640), .A (nx3870)) ;
    inv02 ix3641 (.Y (nx3642), .A (nx3798)) ;
    inv02 ix3643 (.Y (nx3644), .A (nx3798)) ;
    inv02 ix3647 (.Y (nx3648), .A (nx3822)) ;
    inv02 ix3649 (.Y (nx3650), .A (nx3822)) ;
    inv02 ix3651 (.Y (nx3652), .A (nx3822)) ;
    inv02 ix3655 (.Y (nx3656), .A (nx3880)) ;
    inv02 ix3657 (.Y (nx3658), .A (nx3834)) ;
    inv02 ix3659 (.Y (nx3660), .A (nx3834)) ;
    inv02 ix3665 (.Y (nx3666), .A (nx3890)) ;
    inv02 ix3667 (.Y (nx3668), .A (nx3704)) ;
    inv02 ix3673 (.Y (nx3674), .A (NOT_src_1_0)) ;
    inv02 ix3675 (.Y (nx3676), .A (NOT_src_1_0)) ;
    inv02 ix3677 (.Y (nx3678), .A (NOT_src_1_0)) ;
    inv02 ix3681 (.Y (nx3682), .A (NOT_src_1_1)) ;
    inv02 ix3683 (.Y (nx3684), .A (NOT_src_1_1)) ;
    buf02 ix3685 (.Y (nx3686), .A (nx2814)) ;
    inv02 ix3691 (.Y (nx3692), .A (NOT_src_1_6)) ;
    inv02 ix3693 (.Y (nx3694), .A (NOT_src_1_6)) ;
    inv02 ix3695 (.Y (nx3696), .A (NOT_src_1_4)) ;
    inv02 ix3697 (.Y (nx3698), .A (NOT_src_1_4)) ;
    inv02 ix3699 (.Y (nx3700), .A (NOT_src_2_0)) ;
    inv02 ix3701 (.Y (nx3702), .A (nx3894)) ;
    inv02 ix3703 (.Y (nx3704), .A (nx3894)) ;
    inv02 ix3705 (.Y (nx3706), .A (nx2)) ;
    inv02 ix3709 (.Y (nx3710), .A (nx3772)) ;
    inv02 ix3711 (.Y (nx3712), .A (nx3772)) ;
    buf02 ix3713 (.Y (nx3714), .A (nx3017)) ;
    buf02 ix3715 (.Y (nx3716), .A (nx3029)) ;
    inv01 ix3717 (.Y (nx3718), .A (nx220)) ;
    buf02 ix3719 (.Y (nx3720), .A (nx3083)) ;
    buf02 ix3721 (.Y (nx3722), .A (nx3160)) ;
    inv01 ix3723 (.Y (nx3724), .A (nx894)) ;
    inv01 ix3725 (.Y (nx3726), .A (nx872)) ;
    inv01 ix3727 (.Y (nx3728), .A (nx856)) ;
    inv01 ix3729 (.Y (nx3730), .A (nx848)) ;
    inv01 ix3731 (.Y (nx3732), .A (nx836)) ;
    mux21 ix2723 (.Y (nx2722), .A0 (nx2811), .A1 (nx2409), .S0
(nx3854)) ;
    and04 ix1163 (.Y (nx1162), .A0 (nx3868), .A1 (nx3876), .A2
(nx3822), .A3 (nx3870)) ;
    and04 ix1293 (.Y (nx1292), .A0 (nx3868), .A1 (nx3876), .A2
(nx3864), .A3 (nx3870)) ;
    and02 ix833 (.Y (nx832), .A0 (nx3822), .A1 (nx3848)) ;
    and02 ix1441 (.Y (nx1440), .A0 (nx3824), .A1 (nx3828)) ;
    and04 ix1423 (.Y (nx1422), .A0 (nx3868), .A1 (nx3876), .A2
(nx3802), .A3 (nx3864)) ;
    and02 ix1435 (.Y (nx1434), .A0 (nx3870), .A1 (nx3848)) ;
    and02 ix1643 (.Y (nx1642), .A0 (nx3824), .A1 (nx3886)) ;
    and02 ix845 (.Y (nx844), .A0 (nx3872), .A1 (nx3828)) ;
    and04 ix1621 (.Y (nx1620), .A0 (nx3868), .A1 (nx3878), .A2
(nx3808), .A3 (nx3802)) ;
    and02 ix1633 (.Y (nx1632), .A0 (nx3864), .A1 (nx3850)) ;
```

```
    and02 ix1835 (.Y (nx1834), .A0 (nx3824), .A1 (nx3838)) ;
    and04 ix1805 (.Y (nx1804), .A0 (nx3816), .A1 (nx3878), .A2
(nx3808), .A3 (nx3868)) ;
    and02 ix1817 (.Y (nx1816), .A0 (nx3804), .A1 (nx3850)) ;
    and02 ix1823 (.Y (nx1822), .A0 (nx3864), .A1 (nx3830)) ;
    and02 ix1829 (.Y (nx1828), .A0 (nx3872), .A1 (nx3888)) ;
    and02 ix1999 (.Y (nx1998), .A0 (nx3824), .A1 (nx3884)) ;
    and02 ix853 (.Y (nx852), .A0 (nx3866), .A1 (nx3888)) ;
    and02 ix1801 (.Y (nx1800), .A0 (nx3816), .A1 (nx3878)) ;
    and02 ix1977 (.Y (nx1976), .A0 (nx3810), .A1 (nx3850)) ;
    and02 ix1983 (.Y (nx1982), .A0 (nx3804), .A1 (nx3830)) ;
    and02 ix1993 (.Y (nx1992), .A0 (nx3872), .A1 (nx3838)) ;
    and02 ix1803 (.Y (nx1802), .A0 (nx3810), .A1 (nx3678)) ;
    and02 ix2107 (.Y (nx2106), .A0 (nx3816), .A1 (nx3850)) ;
    and02 ix2109 (.Y (nx2108), .A0 (nx3810), .A1 (nx3830)) ;
    and02 ix2115 (.Y (nx2114), .A0 (nx3804), .A1 (nx3888)) ;
    and02 ix2121 (.Y (nx2120), .A0 (nx3866), .A1 (nx3840)) ;
    and02 ix2127 (.Y (nx2126), .A0 (nx3872), .A1 (nx3884)) ;
    and02 ix2133 (.Y (nx2132), .A0 (nx3824), .A1 (nx3844)) ;
    and02 ix1181 (.Y (nx1180), .A0 (nx3890), .A1 (nx3830)) ;
    and02 ix1051 (.Y (nx1050), .A0 (nx3890), .A1 (nx3850)) ;
    and02 ix1043 (.Y (nx1042), .A0 (nx3824), .A1 (nx3678)) ;
    and02 ix803 (.Y (nx802), .A0 (nx3880), .A1 (nx3878)) ;
    and02 ix1175 (.Y (nx1174), .A0 (nx3880), .A1 (nx3850)) ;
    and02 ix1315 (.Y (nx1314), .A0 (nx3892), .A1 (nx3888)) ;
    and02 ix1309 (.Y (nx1308), .A0 (nx3880), .A1 (nx3830)) ;
    and02 ix1453 (.Y (nx1452), .A0 (nx3892), .A1 (nx3840)) ;
    and02 ix1447 (.Y (nx1446), .A0 (nx3880), .A1 (nx3888)) ;
    and02 ix1655 (.Y (nx1654), .A0 (nx3892), .A1 (nx3884)) ;
    and02 ix1649 (.Y (nx1648), .A0 (nx3882), .A1 (nx3840)) ;
    and02 ix1847 (.Y (nx1846), .A0 (nx3892), .A1 (nx3844)) ;
    and02 ix1841 (.Y (nx1840), .A0 (nx3882), .A1 (nx3884)) ;
    and02 ix2005 (.Y (nx2004), .A0 (nx3882), .A1 (nx3846)) ;
    and02 ix869 (.Y (nx868), .A0 (nx3804), .A1 (nx3840)) ;
    and02 ix2231 (.Y (nx2230), .A0 (nx3818), .A1 (nx3830)) ;
    and02 ix2233 (.Y (nx2232), .A0 (nx3810), .A1 (nx3888)) ;
    and02 ix2243 (.Y (nx2242), .A0 (nx3866), .A1 (nx3884)) ;
    and02 ix2249 (.Y (nx2248), .A0 (nx3872), .A1 (nx3846)) ;
    and02 ix2339 (.Y (nx2338), .A0 (nx3818), .A1 (nx3888)) ;
    and02 ix2341 (.Y (nx2340), .A0 (nx3810), .A1 (nx3840)) ;
    and02 ix2347 (.Y (nx2346), .A0 (nx3804), .A1 (nx3694)) ;
    and02 ix2353 (.Y (nx2352), .A0 (nx3794), .A1 (nx3846)) ;
    and02 ix877 (.Y (nx876), .A0 (nx3810), .A1 (nx3694)) ;
    and02 ix2435 (.Y (nx2434), .A0 (nx3818), .A1 (nx3840)) ;
    and02 ix2441 (.Y (nx2440), .A0 (nx3804), .A1 (nx3846)) ;
    and02 ix2515 (.Y (nx2514), .A0 (nx3818), .A1 (nx3694)) ;
    and02 ix2517 (.Y (nx2516), .A0 (nx3810), .A1 (nx3846)) ;
    and02 ix891 (.Y (nx890), .A0 (nx3818), .A1 (nx3846)) ;
    ao22 ix2667 (.Y (nx2666), .A0 (nx3818), .A1 (nx3846), .B0
(nx3738), .B1 (nx888)) ;
    inv01 ix3737 (.Y (nx3738), .A (nx3686)) ;
    ao22 ix1685 (.Y (nx1684), .A0 (nx3804), .A1 (nx3840), .B0
(nx3740), .B1 (nx866)) ;
    inv01 ix3739 (.Y (nx3740), .A (nx2817)) ;
    ao22 ix1345 (.Y (nx1344), .A0 (nx3872), .A1 (nx3830), .B0
(nx3742), .B1 (nx842)) ;
```

```
    inv01 ix3741 (.Y (nx3742), .A (nx2820)) ;
    ao22 ix1081 (.Y (nx1080), .A0 (nx3882), .A1 (nx3878), .B0
(nx3744), .B1 (nx824)) ;
    inv01 ix3743 (.Y (nx3744), .A (nx2823)) ;
    or02 ix25 (.Y (nx24), .A0 (nx3678), .A1 (nx3892)) ;
    and02 ix5 (.Y (nx4), .A0 (nx3892), .A1 (nx3678)) ;
    or02 ix831 (.Y (nx830), .A0 (nx3850), .A1 (nx3824)) ;
    or02 ix851 (.Y (nx850), .A0 (nx3696), .A1 (nx3794)) ;
    or02 ix875 (.Y (nx874), .A0 (nx3694), .A1 (nx3812)) ;
    xor2 ix2848 (.Y (nx2847), .A0 (nx1736), .A1 (nx2896)) ;
    oai22 ix2850 (.Y (nx1736), .A0 (nx3812), .A1 (nx3746), .B0
(NOT_src_1_6), .B1 (nx2893)) ;
    ao22 ix1251 (.Y (nx1250), .A0 (NOT_src_2_3), .A1 (nx1242), .B0
(nx3832), .B1 (nx3748)) ;
    inv01 ix3747 (.Y (nx3748), .A (nx2861)) ;
    ao22 ix1015 (.Y (nx1014), .A0 (NOT_src_2_1), .A1 (nx1006), .B0
(nx3878), .B1 (nx3750)) ;
    inv01 ix3749 (.Y (nx3750), .A (nx2868)) ;
    or02 ix47 (.Y (nx46), .A0 (nx3678), .A1 (nx3894)) ;
    or02 ix1119 (.Y (nx1118), .A0 (nx1014), .A1 (nx3852)) ;
    or02 ix1375 (.Y (nx1374), .A0 (nx1250), .A1 (nx3696)) ;
    nor02ii ix1729 (.Y (nx3746), .A0 (nx3694), .A1 (nx2893)) ;
    or02 ix2641 (.Y (nx2640), .A0 (nx1736), .A1 (nx2844)) ;
    nor02ii ix141 (.Y (nx140), .A0 (nx2905), .A1 (NOT_src_2_2)) ;
    nor02ii ix133 (.Y (nx132), .A0 (nx2908), .A1 (NOT_src_2_4)) ;
    or03 ix2909 (.Y(nx2908),.A0(nx3806),.A1 (nx3818), .A2 (nx3812));
    or02 ix2911 (.Y (nx2910), .A0 (nx3892), .A1 (nx3854)) ;
    mux21_ni ix2737 (.Y (int_des_ac), .A0 (nx1344), .A1 (nx2881),
.S0 (nx3854)) ;
    xor2 ix2921 (.Y (nx2920), .A0 (nx2922), .A1 (nx2844)) ;
    nor02ii ix1469 (.Y (nx1468), .A0 (nx2925), .A1 (nx3842)) ;
    and02 ix89 (.Y (nx88), .A0 (nx2939), .A1 (NOT_op_code_1)) ;
    or02 ix2942 (.Y (nx2941), .A0 (nx3786), .A1 (NOT_op_code_2)) ;
    xor2 ix969 (.Y (nx968), .A0 (int_src_3_1), .A1 (nx3878)) ;
    xor2 ix1071 (.Y (nx1070), .A0 (int_src_3_2), .A1 (nx3852)) ;
    xor2 ix1201 (.Y (nx1200), .A0 (int_src_3_3), .A1 (nx3832)) ;
    xor2 ix1335 (.Y (nx1334), .A0 (int_src_3_4), .A1 (nx3696)) ;
    xor2 ix1517 (.Y (nx1516), .A0 (int_src_3_5), .A1 (nx3842)) ;
    xor2 ix1675 (.Y (nx1674), .A0 (int_src_3_6), .A1 (nx3694)) ;
    xor2 ix1867 (.Y (nx1866), .A0 (nx3482), .A1 (nx2844)) ;
    xor2 ix1923 (.Y (nx1922), .A0 (nx3482), .A1 (nx3702)) ;
    or02 ix759 (.Y (nx758), .A0 (nx3894), .A1 (nx3678)) ;
    xor2 ix2997 (.Y (nx2996), .A0 (nx3882), .A1 (nx748)) ;
    and02 ix3000 (.Y (nx2999), .A0 (nx3772), .A1 (nx3704)) ;
    or02 ix595 (.Y (nx594), .A0 (nx3684), .A1 (nx3894)) ;
    xor2 ix585 (.Y (nx584), .A0 (nx3882), .A1 (nx3012)) ;
    xor2 ix3013 (.Y (nx3012), .A0 (nx3852), .A1 (nx580)) ;
    xor2 ix481 (.Y (nx480), .A0 (nx3806), .A1 (nx3714)) ;
    or02 ix355 (.Y (nx354), .A0 (nx3832), .A1 (nx3894)) ;
    xor2 ix345 (.Y (nx344), .A0 (nx3834), .A1 (nx3024)) ;
    xor2 ix3025 (.Y (nx3024), .A0 (nx3696), .A1 (nx340)) ;
    xor2 ix279 (.Y (nx278), .A0 (nx3872), .A1 (nx3716)) ;
    or02 ix205 (.Y (nx204), .A0 (nx3842), .A1 (nx3894)) ;
    xor2 ix3034 (.Y (nx3033), .A0 (nx3834), .A1 (nx192)) ;
    xor2 ix193 (.Y (nx192), .A0 (nx3694), .A1 (nx3036)) ;
    nor02ii ix3037 (.Y (nx3036), .A0 (nx3038), .A1 (nx3702)) ;
```

```
    ao221 ix3039 (.Y (nx3038), .A0 (nx3834), .A1 (NOT_src_1_7), .B0
(nx3702), .B1 (NOT_src_1_6), .C0 (nx3752)) ;
    inv01 ix3751 (.Y (nx3752), .A (nx140)) ;
    xor2 ix3041 (.Y (nx3040), .A0 (nx3826), .A1 (nx184)) ;
    and02 ix3044 (.Y (nx3043), .A0 (NOT_src_1_6), .A1 (nx3704)) ;
    xor2 ix165 (.Y (nx164), .A0 (nx3834), .A1 (nx3046)) ;
    or02 ix269 (.Y (nx268), .A0 (nx3696), .A1 (nx3792)) ;
    xor2 ix3062 (.Y (nx3061), .A0 (nx3834), .A1 (nx256)) ;
    xor2 ix257 (.Y (nx256), .A0 (nx3842), .A1 (nx3064)) ;
    and02 ix3065 (.Y (nx3064), .A0 (nx220), .A1 (nx3704)) ;
    xor2 ix3067 (.Y (nx3066), .A0 (nx3826), .A1 (nx240)) ;
    mux21_ni ix241 (.Y (nx240), .A0 (nx192),.A1(nx236),.S0 (nx220));
    xor2 ix3071 (.Y (nx3070), .A0 (nx3826), .A1 (nx326)) ;
    xor2 ix3075 (.Y (nx3074), .A0 (nx3874), .A1 (nx308)) ;
    xor2 ix3079 (.Y (nx3078), .A0 (nx3794), .A1 (nx298)) ;
    or02 ix463 (.Y (nx462), .A0 (nx3852), .A1 (nx3792)) ;
    xor2 ix3096 (.Y (nx3095), .A0 (nx3836), .A1 (nx450)) ;
    xor2 ix451 (.Y (nx450), .A0 (nx3832), .A1 (nx3098)) ;
    nor02ii ix3099 (.Y (nx3098), .A0 (nx3720), .A1 (nx3704)) ;
    xor2 ix437 (.Y (nx436), .A0 (nx3826), .A1 (nx3101)) ;
    xor2 ix3106 (.Y (nx3105), .A0 (nx3798), .A1 (nx416)) ;
    xor2 ix3110 (.Y (nx3109), .A0 (nx3794), .A1 (nx398)) ;
    xor2 ix3114 (.Y (nx3113), .A0 (nx3826), .A1 (nx566)) ;
    xor2 ix551 (.Y (nx550), .A0 (nx3798), .A1 (nx3118)) ;
    xor2 ix3123 (.Y (nx3122), .A0 (nx3796), .A1 (nx530)) ;
    xor2 ix3127 (.Y (nx3126), .A0 (nx3806), .A1 (nx512)) ;
    or02 ix3131 (.Y (nx3130), .A0 (nx3820), .A1 (nx3812)) ;
    xor2 ix3133 (.Y (nx3132), .A0 (nx3812), .A1 (nx500)) ;
    xor2 ix729 (.Y (nx728), .A0 (nx3826), .A1 (nx3138)) ;
    mux21_ni ix3139(.Y(nx3138),.A0(nx3012),.A1(nx3140),.S0(nx3772));
    xor2 ix3143 (.Y (nx3142), .A0 (nx3798), .A1 (nx708)) ;
    mux21_ni ix709 (.Y (nx708),.A0(nx566),.A1(nx704), .S0 (nx3772));
    xor2 ix693 (.Y (nx692), .A0 (nx3796), .A1 (nx3147)) ;
    mux21_ni ix3148 (.Y(nx3147),.A0(nx3118),.A1(nx3149),.S0nx3772));
    xor2 ix3152 (.Y (nx3151), .A0 (nx3806), .A1 (nx672)) ;
    mux21_ni ix673 (.Y (nx672), .A0(nx530),.A1(nx668),.S0 (nx3774));
    xor2 ix3156 (.Y (nx3155), .A0 (nx3812), .A1 (nx654)) ;
    mux21_ni ix655 (.Y (nx654), .A0(nx512),.A1(nx650),.S0 (nx3774));
    xor2 ix639 (.Y (nx638), .A0 (nx3820), .A1 (nx3722)) ;
    xor2 ix2053 (.Y (nx2052), .A0 (nx3482), .A1 (nx3836)) ;
    nor02ii ix3 (.Y (nx2), .A0 (NOT_op_code_0), .A1 (nx3786)) ;
    or02 ix2029 (.Y (nx2028), .A0 (nx1904), .A1 (nx3836)) ;
    nor02ii ix1905 (.Y (nx1904), .A0 (nx2952), .A1 (nx3704)) ;
    xor2 ix2177 (.Y (nx2176), .A0 (nx3482), .A1 (nx2639)) ;
    xor2 ix2293 (.Y (nx2292), .A0 (nx3482), .A1 (nx3798)) ;
    or02 ix2269 (.Y (nx2268), .A0 (nx2154), .A1 (nx3798)) ;
    nor02ii ix2155 (.Y (nx2154), .A0 (nx3181), .A1 (nx3826)) ;
    xor2 ix2397 (.Y (nx2396), .A0 (nx3482), .A1 (nx3796)) ;
    nor02ii ix3220 (.Y (nx3219), .A0 (nx3796), .A1 (nx3209)) ;
    nor02ii ix2375 (.Y (nx2374), .A0 (nx3209), .A1 (nx3796)) ;
    xor2 ix2485 (.Y (nx2484), .A0 (nx3484), .A1 (nx3806)) ;
    xor2 ix2561 (.Y (nx2560), .A0 (nx3484), .A1 (nx3812)) ;
    xor2 ix2617 (.Y (nx2616), .A0 (nx2538), .A1 (nx3820)) ;
    nor02ii ix2539 (.Y (nx2538), .A0 (nx3240), .A1 (nx3814)) ;
    xor2 ix3269 (.Y (nx3268), .A0 (nx3484), .A1 (NOT_src_2_7)) ;
```

```
    or04 ix3271 (.Y (nx3270), .A0 (nx3754), .A1 (nx638), .A2
(nx3820), .A3 (nx3163)) ;
    inv01 ix3753 (.Y (nx3754), .A (nx782)) ;
    nor02ii ix113 (.Y (nx112), .A0 (nx3284), .A1 (nx78)) ;
    or02 ix3285 (.Y (nx3284), .A0 (nx2939), .A1 (nx2936)) ;
    nor02ii ix103 (.Y (nx102), .A0 (nx3284), .A1 (nx2)) ;
    or02 ix3296 (.Y (nx3295), .A0 (nx2939), .A1 (NOT_op_code_1)) ;
    nand03 ix17 (.Y (nx3301), .A0(nx2914),.A1(nx2939),.A2 (nx2936));
    xor2 ix965 (.Y (nx964), .A0 (nx2823), .A1 (nx3856)) ;
    oai21 ix829 (.Y (nx3324), .A0(nx3684),.A1(nx3836),.B0 (nx3756));
    inv01 ix3755 (.Y (nx3756), .A (nx802)) ;
    xnor2 ix957 (.Y (nx956), .A0 (nx3684), .A1 (nx3370)) ;
    or02 ix825 (.Y (nx824), .A0 (nx3684), .A1 (nx3836)) ;
    nor02ii ix91 (.Y (nx90), .A0 (nx3788), .A1 (nx3768)) ;
    ao22 ix813 (.Y (nx812), .A0 (nx3836), .A1 (nx3678), .B0
(nx3704), .B1 (nx3684)) ;
    nor02ii ix817 (.Y (nx816), .A0 (nx3295), .A1 (nx2)) ;
    and03 ix805 (.Y (nx804), .A0 (nx2), .A1 (nx2939), .A2(nx2936)) ;
    oai21 ix3338 (.Y (nx894), .A0(nx2844),.A1(nx3820),.B0 (nx3758));
    inv01 ix3757 (.Y (nx3758), .A (nx890)) ;
    oai21 ix3342 (.Y (nx872), .A0(nx3842),.A1(nx3806),.B0 (nx3760));
    inv01 ix3759 (.Y (nx3760), .A (nx868)) ;
    and04 ix3344 (.Y (nx3343), .A0 (nx3784), .A1 (nx3782), .A2
(nx3780), .A3 (nx3856)) ;
    oai21 ix3346 (.Y (nx856), .A0(nx3698),.A1(nx3796),.B0 (nx3762));
    inv01 ix3761 (.Y (nx3762), .A (nx852)) ;
    oai21 ix3348 (.Y (nx848), .A0(nx3832),.A1(nx3800),.B0 (nx3764));
    inv01 ix3763 (.Y (nx3764), .A (nx844)) ;
    oai21 ix3350 (.Y (nx836), .A0(nx3852),.A1(nx2639),.B0 (nx3766));
    inv01 ix3765 (.Y (nx3766), .A (nx832)) ;
    xor2 ix1017 (.Y (nx1016), .A0 (nx1014), .A1 (nx3780)) ;
    nor02ii ix3369 (.Y (nx3368), .A0 (nx3370), .A1 (NOT_src_1_1)) ;
    or02 ix951 (.Y (nx950), .A0 (nx3852), .A1 (nx3684)) ;
    or02 ix843 (.Y (nx842), .A0 (nx3832), .A1 (nx3800)) ;
    xor2 ix1213 (.Y (nx1212), .A0 (nx2820), .A1 (nx3782)) ;
    or03 ix1097(.Y(nx1096),.A0(nx3852), .A1 (nx3684), .A2 (nx3832));
    xor2 ix1253 (.Y (nx1252), .A0 (nx1250), .A1 (nx3784)) ;
    or02 ix1227 (.Y (nx1226), .A0 (nx952), .A1 (nx3698)) ;
    xor2 ix1505 (.Y (nx1504), .A0 (nx2817), .A1 (nx872)) ;
    or02 ix867 (.Y (nx866), .A0 (nx3842), .A1 (nx3806)) ;
    xor2 ix1687 (.Y (nx1686), .A0 (nx1684), .A1 (nx3858)) ;
    xor2 ix1565 (.Y (nx1564), .A0 (nx2893), .A1 (nx3858)) ;
    xor2 ix1879 (.Y (nx1878), .A0 (nx3686), .A1 (nx894)) ;
    or02 ix889 (.Y (nx888), .A0 (nx2844), .A1 (nx3820)) ;
    xor2 ix1739 (.Y (nx1738), .A0 (nx1736), .A1 (nx894)) ;
    nor02ii ix3535 (.Y (nx3536), .A0 (nx3284), .A1 (nx2)) ;
    nor02ii ix3541 (.Y (nx3542), .A0 (nx3284), .A1 (nx78)) ;
    buf02 ix3767 (.Y (nx3768), .A (nx88)) ;
    buf02 ix3769 (.Y (nx3770), .A (nx90)) ;
    inv02 ix3771 (.Y (nx3772), .A (nx3001)) ;
    inv02 ix3773 (.Y (nx3774), .A (nx3001)) ;
    buf02 ix3775 (.Y (nx3776), .A (nx804)) ;
    buf02 ix3777 (.Y (nx3778), .A (nx816)) ;
    inv02 ix3779 (.Y (nx3780), .A (nx3732)) ;
    inv01 ix3781 (.Y (nx3782), .A (nx3730)) ;
    inv01 ix3783 (.Y (nx3784), .A (nx3728)) ;
```

```
    inv02 ix3785 (.Y (nx3786), .A (nx2914)) ;
    inv02 ix3787 (.Y (nx3788), .A (nx2914)) ;
    inv01 ix3791 (.Y (nx3792), .A (nx3700)) ;
    inv02 ix3793 (.Y (nx3794), .A (NOT_src_2_4)) ;
    inv02 ix3795 (.Y (nx3796), .A (NOT_src_2_4)) ;
    inv02 ix3797 (.Y (nx3798), .A (NOT_src_2_3)) ;
    inv02 ix3799 (.Y (nx3800), .A (NOT_src_2_3)) ;
    inv02 ix3801 (.Y (nx3802), .A (NOT_src_2_5)) ;
    inv02 ix3803 (.Y (nx3804), .A (NOT_src_2_5)) ;
    inv02 ix3805 (.Y (nx3806), .A (NOT_src_2_5)) ;
    inv02 ix3807 (.Y (nx3808), .A (NOT_src_2_6)) ;
    inv02 ix3809 (.Y (nx3810), .A (NOT_src_2_6)) ;
    inv02 ix3811 (.Y (nx3812), .A (NOT_src_2_6)) ;
    inv02 ix3813 (.Y (nx3814), .A (NOT_src_2_6)) ;
    inv02 ix3815 (.Y (nx3816), .A (NOT_src_2_7)) ;
    inv02 ix3817 (.Y (nx3818), .A (NOT_src_2_7)) ;
    inv02 ix3819 (.Y (nx3820), .A (NOT_src_2_7)) ;
    inv02 ix3821 (.Y (nx3822), .A (NOT_src_2_2)) ;
    inv02 ix3823 (.Y (nx3824), .A (NOT_src_2_2)) ;
    inv02 ix3825 (.Y (nx3826), .A (NOT_src_2_2)) ;
    inv02 ix3827 (.Y (nx3828), .A (NOT_src_1_3)) ;
    inv02 ix3829 (.Y (nx3830), .A (NOT_src_1_3)) ;
    inv02 ix3831 (.Y (nx3832), .A (NOT_src_1_3)) ;
    inv02 ix3833 (.Y (nx3834), .A (NOT_src_2_1)) ;
    inv02 ix3835 (.Y (nx3836), .A (NOT_src_2_1)) ;
    inv02 ix3837 (.Y (nx3838), .A (NOT_src_1_5)) ;
    inv02 ix3839 (.Y (nx3840), .A (NOT_src_1_5)) ;
    inv02 ix3841 (.Y (nx3842), .A (NOT_src_1_5)) ;
    inv02 ix3843 (.Y (nx3844), .A (NOT_src_1_7)) ;
    inv02 ix3845 (.Y (nx3846), .A (NOT_src_1_7)) ;
    inv02 ix3847 (.Y (nx3848), .A (NOT_src_1_2)) ;
    inv02 ix3849 (.Y (nx3850), .A (NOT_src_1_2)) ;
    inv02 ix3851 (.Y (nx3852), .A (NOT_src_1_2)) ;
    inv02 ix3853 (.Y (nx3854), .A (NOT_op_code_1)) ;
    inv02 ix3855 (.Y (nx3856), .A (nx3548)) ;
   buf02 ix3857 (.Y (nx3858), .A (nx3339)) ;
   buf02 ix3859 (.Y (nx3860), .A (nx3536)) ;
   buf02 ix3861 (.Y (nx3862), .A (nx3542)) ;
    inv02 ix3863 (.Y (nx3864), .A (NOT_src_2_4)) ;
    inv02 ix3865 (.Y (nx3866), .A (NOT_src_2_4)) ;
    inv02 ix3867 (.Y (nx3868), .A (NOT_src_1_0)) ;
    inv02 ix3869 (.Y (nx3870), .A (NOT_src_2_3)) ;
    inv02 ix3871 (.Y (nx3872), .A (NOT_src_2_3)) ;
    inv02 ix3873 (.Y (nx3874), .A (NOT_src_2_3)) ;
    inv02 ix3875 (.Y (nx3876), .A (NOT_src_1_1)) ;
    inv02 ix3877 (.Y (nx3878), .A (NOT_src_1_1)) ;
    inv02 ix3879 (.Y (nx3880), .A (NOT_src_2_1)) ;
    inv02 ix3881 (.Y (nx3882), .A (NOT_src_2_1)) ;
    inv02 ix3883 (.Y (nx3884), .A (NOT_src_1_6)) ;
    inv02 ix3885 (.Y (nx3886), .A (NOT_src_1_4)) ;
    inv02 ix3887 (.Y (nx3888), .A (NOT_src_1_4)) ;
    inv02 ix3889 (.Y (nx3890), .A (nx3896)) ;
    inv02 ix3891 (.Y (nx3892), .A (nx3896)) ;
    inv02 ix3893 (.Y (nx3894), .A (nx3700)) ;
    inv02 ix3895 (.Y (nx3896), .A (nx3700)) ;
endmodule
```

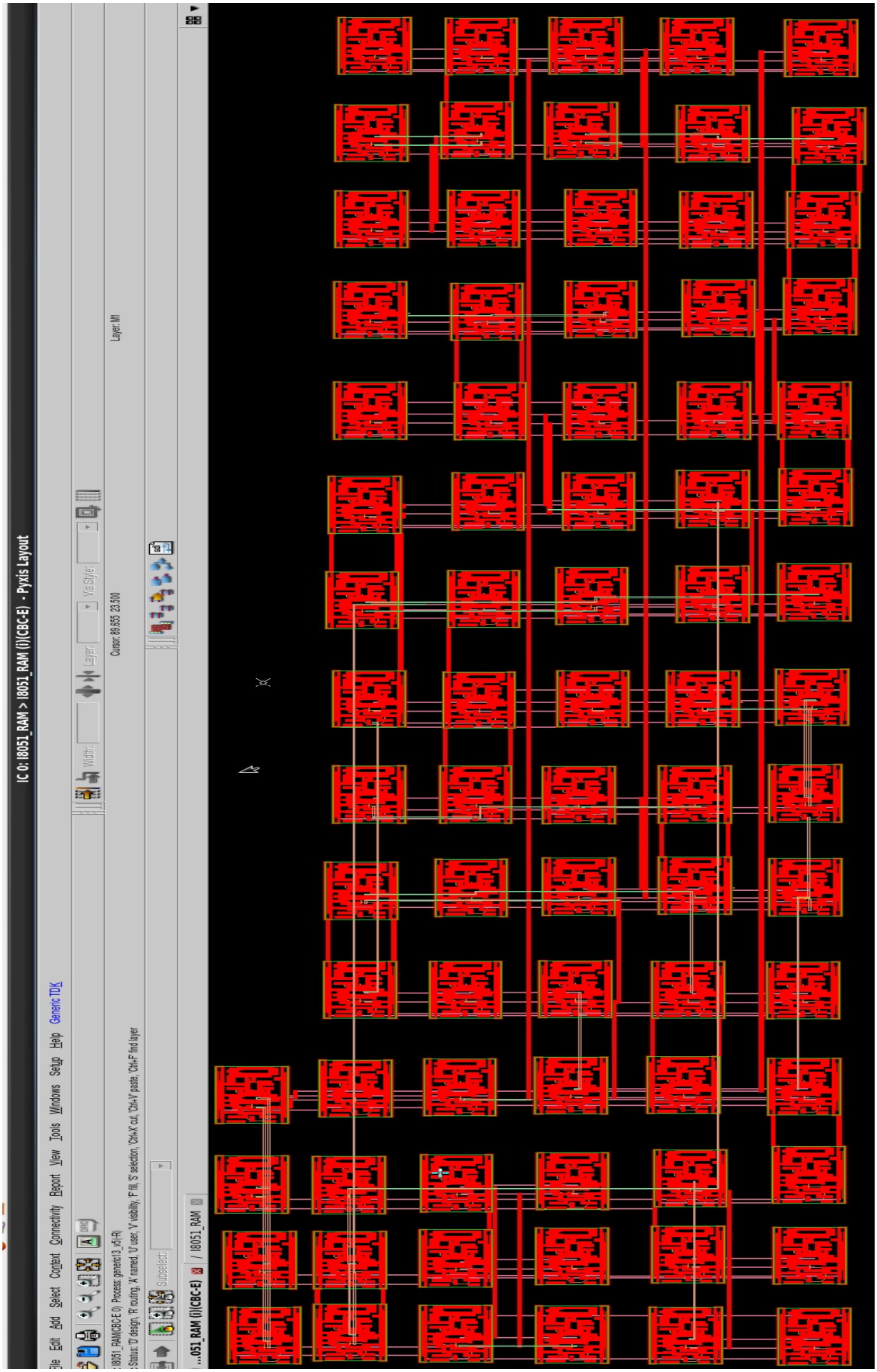
APÊNDICE E – ARQUIVO DE MAPEAMENTO DO BLOCO “ULA” DO 8051

```
// Template file for mapping primitive gates and modules to EDDM
components.
// v10.2_3.0     Tue Jun 25 02:33:04 PDT 2013
p portin $MGC_IC_GENERIC_LIB/portin X
p portout $MGC_IC_GENERIC_LIB/portout X
p portbi $MGC_IC_GENERIC_LIB/portbi X
p ripper $MGC_IC_GENERIC_LIB/rip/1X1 bundle wire
p pagein $MGC_IC_GENERIC_LIB/offpag.in IN
p pageout $MGC_IC_GENERIC_LIB/offpag.out OUT
p netcon $MGC_IC_GENERIC_LIB/netcon IN OUT
p ground $MGC_IC_GENERIC_LIB/vss X
p power $MGC_IC_GENERIC_LIB/vdd X
//
m mux21_ni $GDKGATES/mux21 Y A0 A1 S0
m nor04 $GDKGATES/nor04 Y A0 A1 A2 A3
m or03 $GDKGATES/or03 Y A0 A1 A2
m xnor2 $GDKGATES/xnor2 Y A0 A1
m nor02_2x $GDKGATES/nor02_2x Y A0 A1
m aoi22 $GDKGATES/aoi22 Y A0 A1 B0 B1
m oai21 $GDKGATES/oai21 Y A0 A1 B0
m nand02 $GDKGATES/nand02 Y A0 A1
m nand03 $GDKGATES/nand03 Y A0 A1 A2
m inv02 $GDKGATES/inv02 Y A
m oai32 $GDKGATES/oai32 Y A0 A1 A2 B0 B1
m nand04 $GDKGATES/nand04 Y A0 A1 A2 A3
m xor2 $GDKGATES/xor2 Y A0 A1
m aoi21 $GDKGATES/aoi21 Y A0 A1 B0
m inv01 $GDKGATES/inv01 Y A
m oai22 $GDKGATES/oai22 Y A0 A1 B0 B1
m oai222 $GDKGATES/oai222 Y A0 A1 B0 B1 C0 C1
m aoi332 $GDKGATES/aoi332 Y A0 A1 A2 B0 B1 B2 C0 C1
m nor02ii $GDKGATES/nor02ii Y A0 A1
m oai221 $GDKGATES/oai221 Y A0 A1 B0 B1 C0
m aoi32 $GDKGATES/aoi32 Y A0 A1 A2 B0 B1
m nand02_2x $GDKGATES/nand02_2x Y A0 A1
m aoi322 $GDKGATES/aoi322 Y A0 A1 A2 B0 B1 C0 C1
m nor03_2x $GDKGATES/nor03_2x Y A0 A1 A2
m ao221 $GDKGATES/ao221 Y A0 A1 B0 B1 C0
m oai322 $GDKGATES/oai322 Y A0 A1 A2 B0 B1 C0 C1
m aoi222 $GDKGATES/aoi222 Y A0 A1 B0 B1 C0 C1
m or02 $GDKGATES/or02 Y A0 A1
m and02 $GDKGATES/and02 Y A0 A1
m buf02 $GDKGATES/buf02 Y A
m buf04 $GDKGATES/buf04 Y A
m mux21 $GDKGATES/mux21 Y A0 A1 S0
m and04 $GDKGATES/and04 Y A0 A1 A2 A3
m ao22 $GDKGATES/ao22 Y A0 A1 B0 B1
m or04 $GDKGATES/or04 Y A0 A1 A2 A3
m and03 $GDKGATES/and03 Y A0 A1 A2
```

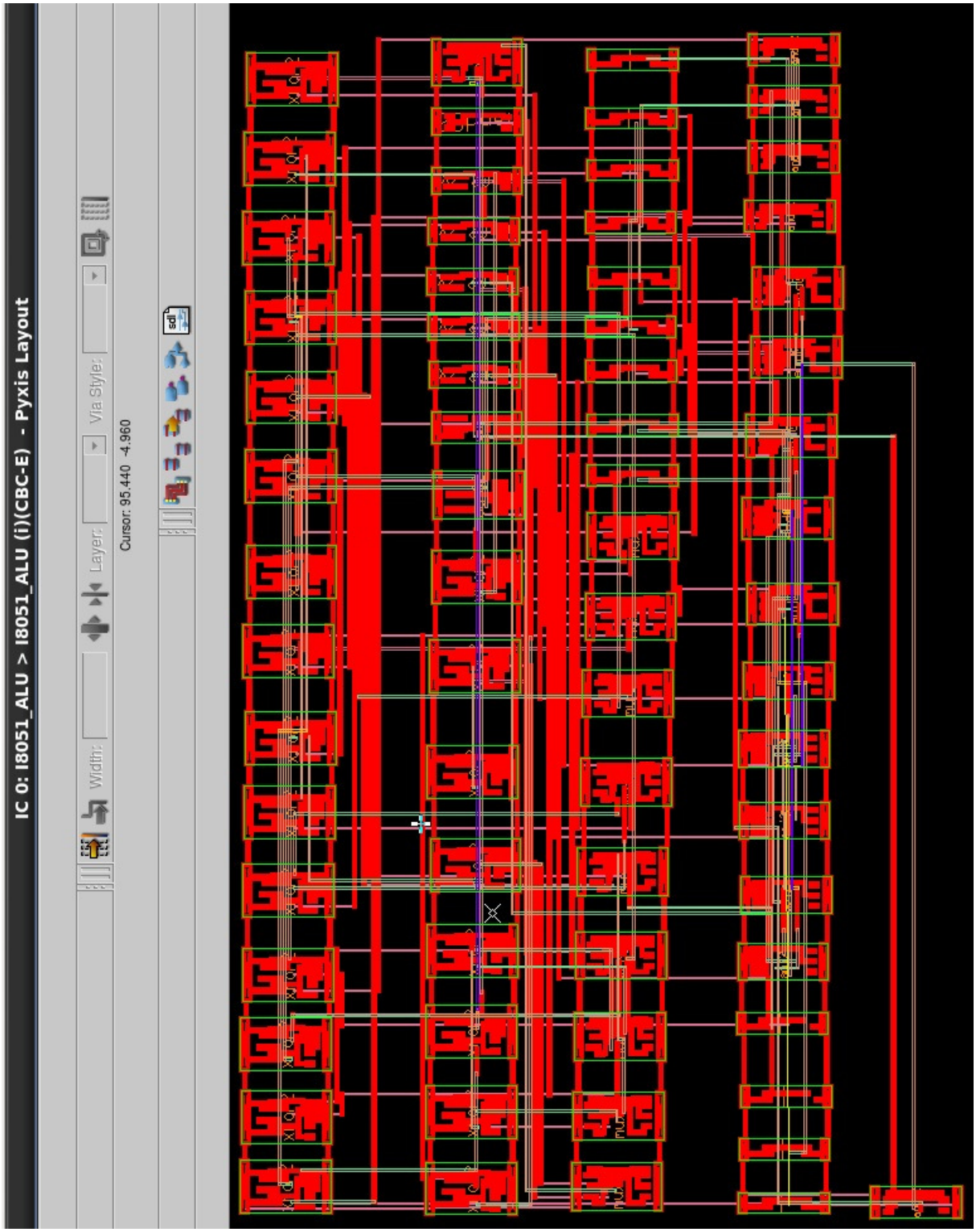
APÊNDICE F – ARQUIVO DE MAPEAMENTO DO BLOCO “RAM” DO 8051

```
// Template file for mapping primitive gates and modules to EDDM
components.
// v10.2_3.0     Tue Jun 25 02:33:04 PDT 2013
p portin $MGC_IC_GENERIC_LIB/portin X
p portout $MGC_IC_GENERIC_LIB/portout X
p portbi $MGC_IC_GENERIC_LIB/portbi X
p ripper $MGC_IC_GENERIC_LIB/rip/1X1 bundle wire
p pagein $MGC_IC_GENERIC_LIB/offpag.in IN
p pageout $MGC_IC_GENERIC_LIB/offpag.out OUT
p netcon $MGC_IC_GENERIC_LIB/netcon IN OUT
p ground $MGC_IC_GENERIC_LIB/ground X
p power $MGC_IC_GENERIC_LIB/vdd X
//
m dffr $GDKGATES/dffr Q QB D CLK R
m mux21_ni $GDKGATES/mux21 Y A0 A1 S0
m nand02 $GDKGATES/nand02 Y A0 A1
m nand02_2x $GDKGATES/nand02_2x Y A0 A1
m inv02 $GDKGATES/inv02 Y A
m nand03 $GDKGATES/nand03 Y A0 A1 A2
m nor02_2x $GDKGATES/nor02_2x Y A0 A1
m aoi21 $GDKGATES/aoi21 Y A0 A1 B0
m aoi43 $GDKGATES/aoi43 Y A0 A1 A2 A3 B0 B1 B2
m nand04 $GDKGATES/nand04 Y A0 A1 A2 A3
m aoi22 $GDKGATES/aoi22 Y A0 A1 B0 B1
m aoi222 $GDKGATES/aoi222 Y A0 A1 B0 B1 C0 C1
m oai21 $GDKGATES/oai21 Y A0 A1 B0
m nor03_2x $GDKGATES/nor03_2x Y A0 A1 A2
m nor04 $GDKGATES/nor04 Y A0 A1 A2 A3
m aoi221 $GDKGATES/aoi221 Y A0 A1 B0 B1 C0
m aoi32 $GDKGATES/aoi32 Y A0 A1 A2 B0 B1
m nand03_2x $GDKGATES/nand03_2x Y A0 A1 A2
m oai222 $GDKGATES/oai222 Y A0 A1 B0 B1 C0 C1
m oai22 $GDKGATES/oai22 Y A0 A1 B0 B1
m or04 $GDKGATES/or04 Y A0 A1 A2 A3
m or02 $GDKGATES/or02 Y A0 A1
m and03 $GDKGATES/and03 Y A0 A1 A2
m inv01 $GDKGATES/inv01 Y A
m buf04 $GDKGATES/buf04 Y A
m buf02 $GDKGATES/buf02 Y A
m inv04 $GDKGATES/inv04 Y A
m ao32 $GDKGATES/ao32 Y A0 A1 A2 B0 B1
m nor02ii $GDKGATES/nor02ii Y A0 A1
m and02 $GDKGATES/and02 Y A0 A1
m or03 $GDKGATES/or03 Y A0 A1 A2
m and04 $GDKGATES/and04 Y A0 A1 A2 A3
m mux21 $GDKGATES/mux21 Y A0 A1 S0
m ao221 $GDKGATES/ao221 Y A0 A1 B0 B1 C0
m ao22 $GDKGATES/ao22 Y A0 A1 B0 B1
```

APÊNDICE G – LEIAUTE DO BLOCO “RAM” DO 8051



APÊNDICE H – LEIAUTE DO BLOCO “ULA” DO 8051



APÊNDICE I - TUTORIAL DE UTILIZAÇÃO DO *SOFTWARE PYXIS LAYOUT* PARA GERAÇÃO AUTOMÁTICA DE LEIAUTES DE CIRCUITOS INTEGRADOS

Assim que um objeto de leiaute for criado no *Project Navigator* do *Mentor Graphics* GDK, deve-se dar um duplo clique nesta célula.

Aparecerá a janela conforme a Figura I.1.

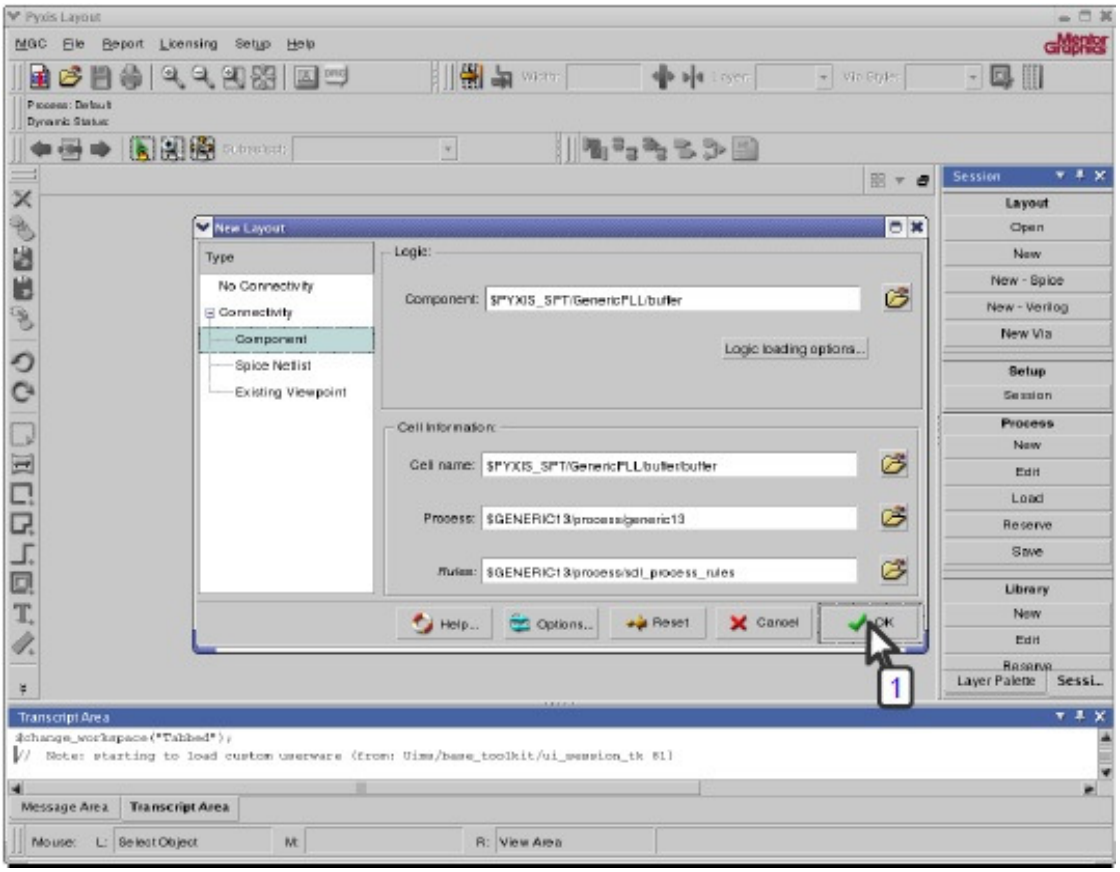


Figura I. 1 – Janela de criação inicial de um leiaute no *Pyxis Layout.*

Apenas clicar no botão OK e o esquemático será exibido no *Pyxis Layout.*

Para iniciar o processo de SDL, clicar no ícone, como demonstra a Figura I.2 (seta número 1). Selecionar *Prompt User* (seta número 2) e clicar no botão OK.

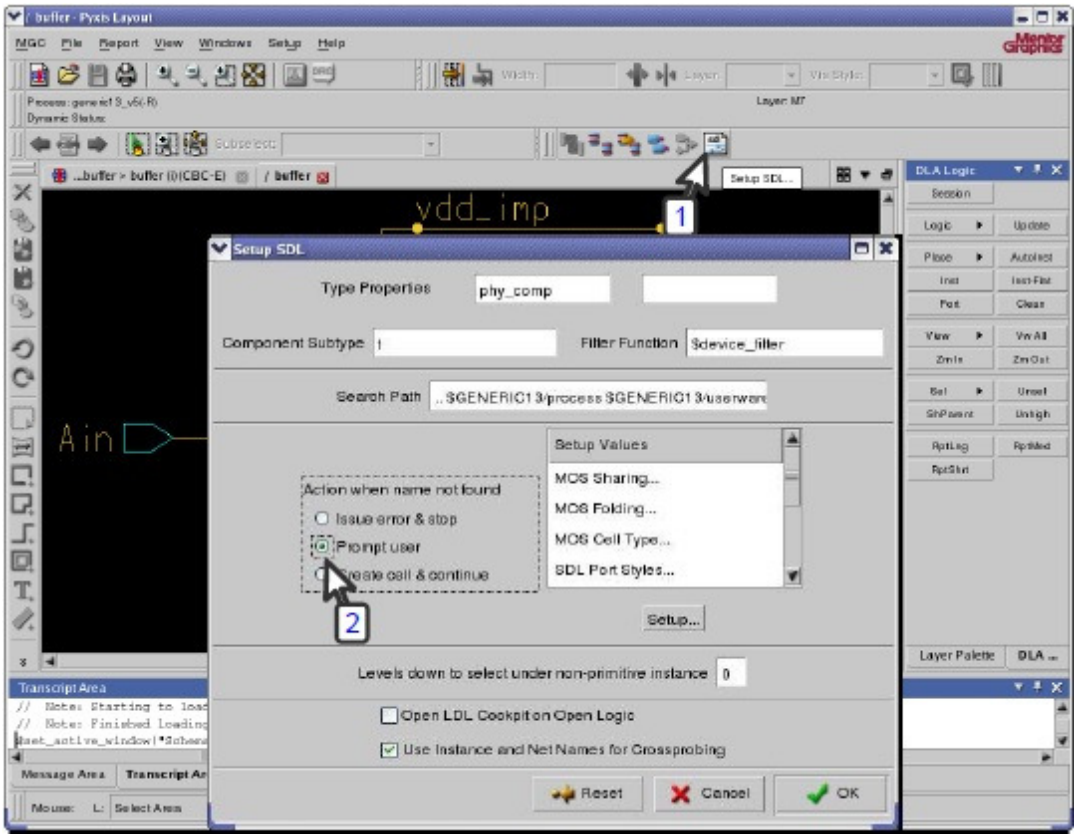


Figura I. 2 – Janela de configuração do SDL no *Pyxys Layout.*

Será aberta uma nova aba ao lado da aba do esquemático, que será utilizada para a criação do leiaute. Selecionando esta aba, acessar o menu *Windows*, e então a opção *LDL Cockpit*.

Nesta nova janela (*LDL Cockpit*) aparecerão todos os dispositivos presentes no esquemático, como ilustra a Figura I.3. Selecione todos os dispositivos (ou faça este procedimento clicando em um dispositivo por vez), clicar com o botão direito do *mouse*, e clicar na opção *Place Instance* (seta número um da Figura I.3). Os dispositivos devem ser posicionados um a um na aba de leiaute.

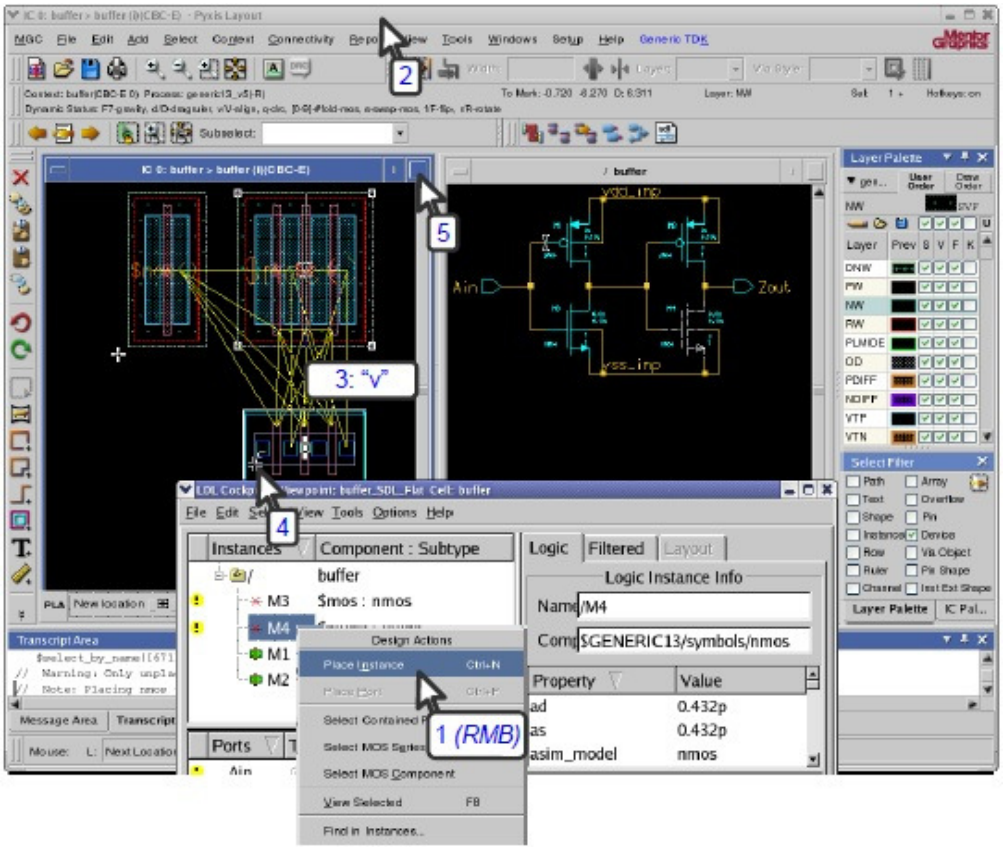


Figura I. 3 – Janela do *Pyxis Layout* para posicionamento dos dispositivos no leiaute.

Opcionalmente, neste momento, pode-se invocar a ferramenta que busca erros de posicionamento conforme as regras de projetos (DRC). Para este procedimento, acessar o menu *Setup, Toolbars, Calibre RealTime*, conforme ilustra a Figura I.4.

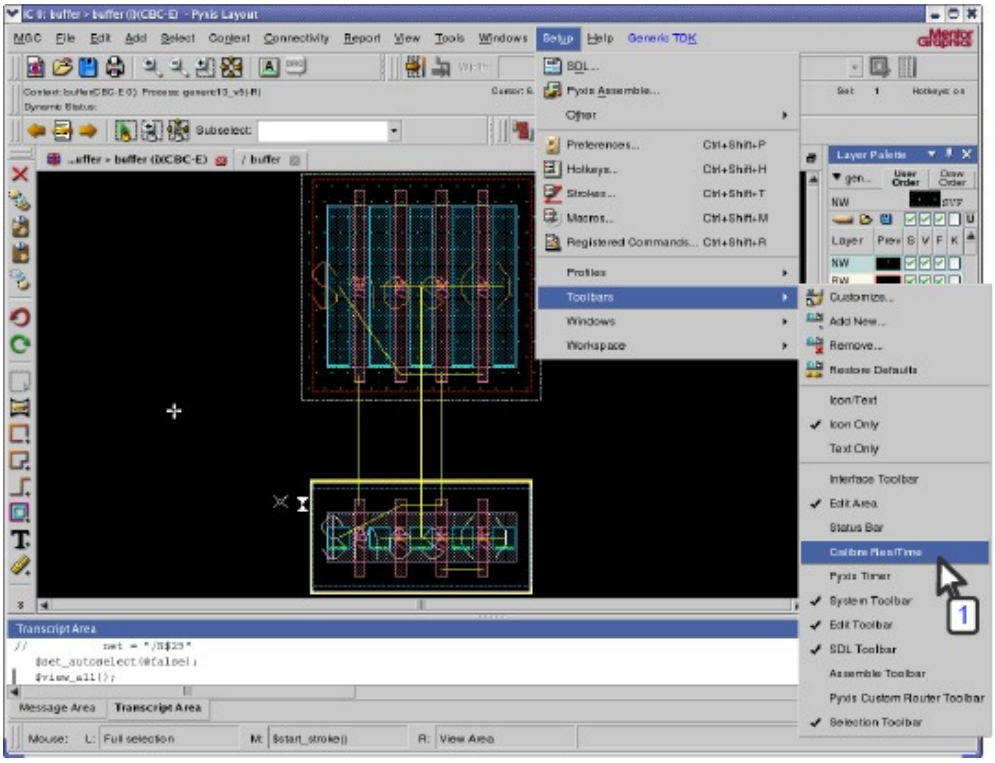


Figura I. 4 – Caminho para selecionar a ferramenta de DRC no *Pyxis Layout.*

Clicar no ícone *Run DRC* (seta número 1 da Figura I.5) para iniciar a análise. Os erros detectados serão evidenciados no leiaute, percorrendo os ícones e opções na barra, ao lado do ícone *Run DRC* (setas 2 e 4 da Figura I.5).

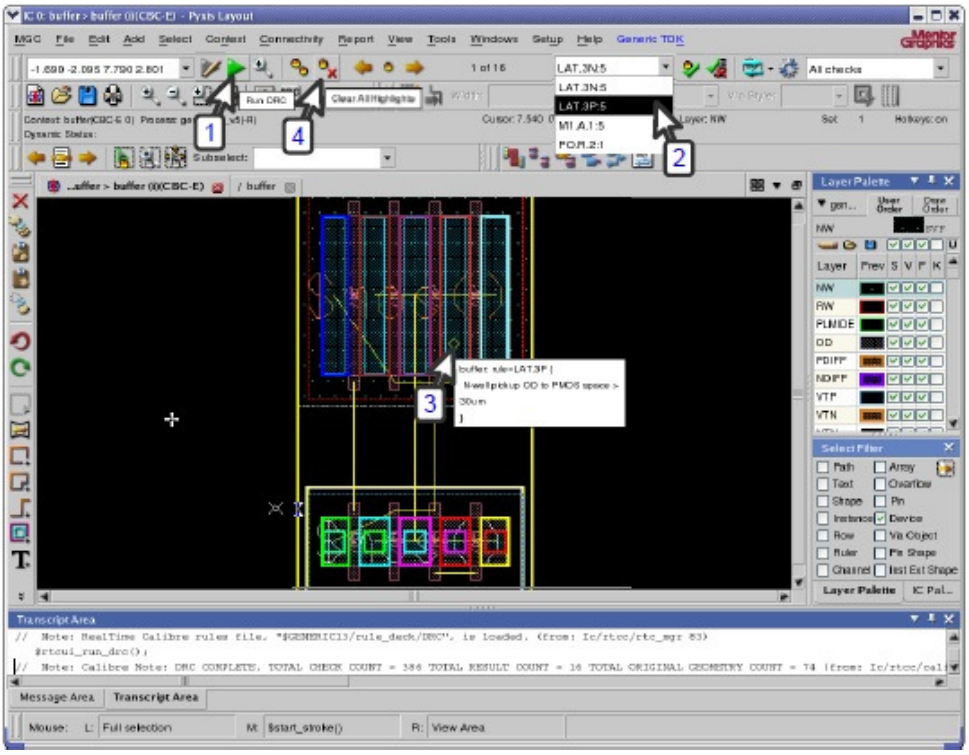


Figura I. 5 – Ícones que devem ser percorridos para análise e evidência de erros de DRC no *Pyxis Layout.*

Os erros podem ser corrigidos editando o leiaute.

Após a etapa de edição do leiaute, clicar na aba do esquemático (seta número 1 da Figura I.6), e na paleta de botões localizada à direita da tela, clicar na opção *Ports* (seta número 2 da Figura I.6).

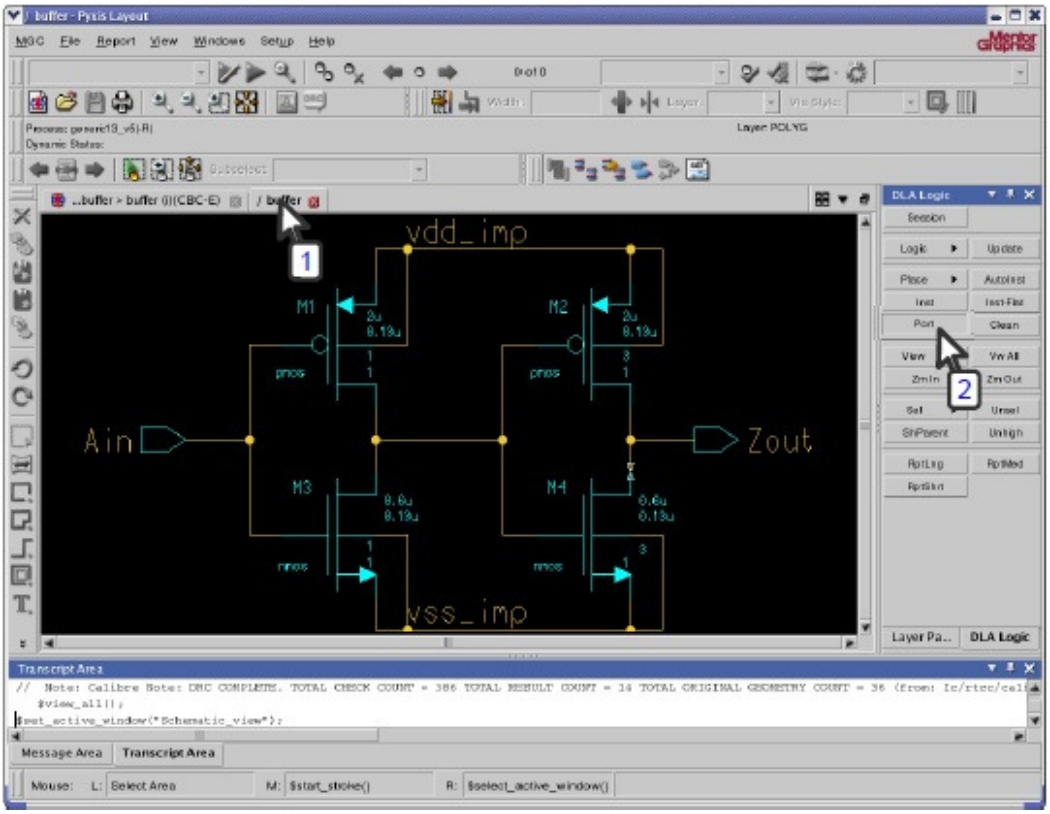


Figura I. 6 – Janela com a visualização de um esquemático no *Pyxis Layout.*

Nesta etapa, serão posicionados os terminais ou portas do projeto no leiaute, referente à alimentação, entradas e saídas.

Posicionadas as portas, a próxima etapa é de autorroteamento (*Autoroute*). Na paleta de botões à direita da tela, clicar na opção *Plan & Place, Route* e finalmente dentro da área denominada *ARoute* clicar na opção *Run*.

Para finalizar o leiaute, opcionalmente pode-se utilizar o *Calibre LVS*, que vai conferir se o leiaute gerado está em conformidade com o esquemático originário do projeto.

Acessar o menu *Tools, Calibre, Run LVS* (seta número 1 da Figura I.7). Na janela do *Calibre,* acessar *Setup, LVS Options* (seta número 2 da Figura I.7).

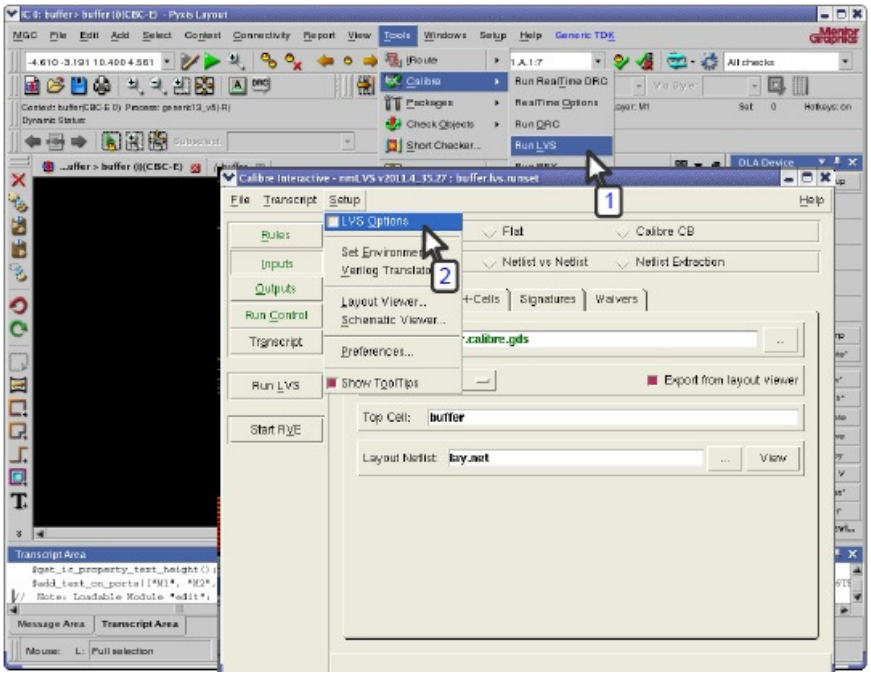


Figura I. 7 – Janela do *Calibre LVS* no *Pyxis Layout.*

Colocar os nomes referente aos terminais de tensão de alimentação do circuito (por exemplo VDD e VSS) nos campos *Power nets* e *Ground nets* (setas número 1 e 2 da Figura I.8). Então clicar em *Run LVS* (seta número 3 da Figura I.8).

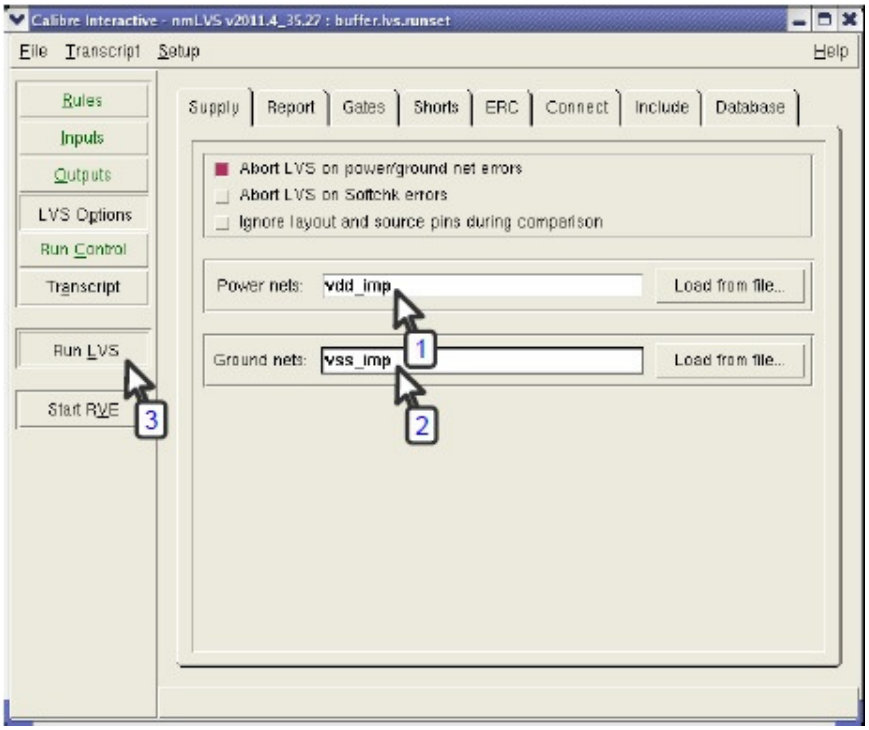


Figura I. 8 – Janela de configuração do *Calibre LVS* no *Pyxis Layout.*