UNIVERSIDADE POSITIVO
NÚCLEO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Raphael D'Oliveira Fogaça

Processador RISC Didático

Trabalho de Conclusão de Curso

Professor Marcelo Micosz Gonçalves
Orientador

Curitiba, novembro de 2009.

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais Eveli e Mario, que me proporcionaram muitas oportunidades no decorrer da minha vida e sempre lutaram junto a mim para que todos os meus obstáculos fossem superados.

A toda a minha família que sempre esteve ao meu lado nos momentos de dificuldade que passei no decorrer desses anos de vida.

Ao meu amigo, orientador e professor Marcelo Mickoz Gonçalves, que sempre com sabedoria e apoio me ajudou no desenvolvimento do projeto.

A todos os professores do curso de Engenharia da Computação, que compartilharam conhecimentos tanto no âmbito profissional quanto nas experiências de vida.

Agradeço também a Monica Dalmolin por toda força e apoio apresentado nos últimos 5 anos, que me impulsionaram para a conclusão dos meus estudos.

E por fim a todos os meus amigos e colegas que de alguma forma, influenciaram e ajudaram na minha formação acadêmica, profissional e moldaram meu caráter ao longo da minha vida.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

NCET	Núcleo de Ciências Exatas e Tecnológicas
FPGA	*Field Programmable Gate Array*
VHDL	*Very High Speed Integrated Circuit*
ROM	*Read Only Memory*
RAM	*Random Acess Memory*
CPU	Unidade central de processamento
ASIC	*Application Specific Integrated Circuits*
VHSIC	*Very High Speed Integrated Circuits*
I/O	*Input/ output*
E/S	Entrada/Saída
LED	*Light-Emitting Diode*
CPU	*Central Processor Unit*
ULA	Unidade Lógica Aritimética
RISC	*Reduced Instruction Set Computer*
CISC	*Complex Instruction Set Computer*
LCD	*Liquid-crystal display*
USB	*Universal Serial Bus*
SRAM	*Static Random Access Memory*
SDRAM	*Synchronous Dynamic Randow Access Memory*
SD	*Secure Digital*
CODEC	*Coder-Decoder*
VGA	*Video Graphics Array*
DAC	*Digital-to-analog Converter*
IrDA	*Infrared Data Association*
AHDL	*Altera Hardware Description Language*
PC	*Program Counter*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

W	Watts

RESUMO

O projeto consiste em desenvolver um processador RISC (*Reduced Instruction Set Computer*) didático com interface de entrada (teclado e/ou mouse) e porta paralela para simular a comunicação com algum dispositivo externo.

Para a implementação do projeto foi utilizado o ambiente Quartus II, em linguagem VHDL/AHDL, onde ocorre a simulação do processador RISC. Que foi embarcado em um Kit FPGA (circuito integrado passível de configuração por *software* para simulação de circuitos digitais, processadores, decodificadores, etc.) Altera DE2.

No escopo do projeto foi utilizado um processador RISC base (apresentado durante a disciplina de microprocessadores do curso de Engenharia da Computação da Universidade Positivo, ministrada pelo Professor Valfredo Pilla Júnior), e incluídos alguns novos módulos para atender as novas funcionalidades exigidas no projeto.

Em sua totalidade o projeto atendeu as expectativas, funcionando todos os recursos previstos na proposta inicial.

Palavras chave: Microprocessador, RISC, *Software*, FPGA

Didactic RISC Processor

ABSTRACT

The project is to develop a didactic RISC processor (Reduced Instruction Set Computer) with input interface (keyboard and/or mouse), parallel port to simulate communication with an external device.

For the implementation of the project was use the Quartus II environment, on VHDL/AHDL language, where the simulation of processor occurs. This processor was be shipped in FPGA (integrated circuit capable of configuration software for simulation of digital circuits, processors, decoders, and others) kit DE2 of Altera.

The project scope was used on a RISC processor (made during the microprocessors discipline of Computer Engineering course in Positivo University, taught by Professor Valfredo Pilla Júnior), and included some new modules to meet the new features required in the project.

In its entirety the project has exceeded expectations, running all the resources proposed in initially.

Keywords: Microcontrolador, RISC, Software, FPGA

CAPÍTULO 1 - INTRODUÇÃO

O projeto consistiu em criar um processador RISC didático embarcado no kit FPGA Altera DE2 com um dispositivo de entrada (teclado e/ou mouse) onde, é possível introduzir ações a serem efetuadas pelo processador através de uma comunicação do dispositivo de entrada com o mesmo. No processador também está disponível uma porta paralela para simular uma comunicação externa com outro dispositivo.

O ambiente de desenvolvimento do processador RISC é o Quartus II da Altera, e o Kit FPGA da altera DE2 foi programado para simular um processador RISC.

Após a conclusão e processamento das instruções no desenvolvimento do hardware, as informações dos resultados são mostrados em uma interface LCD presente no kit FPGA da Altera DE2.

1.1 Histórico

O primeiro processador do mundo nasceu juntamente com o primeiro computador, o Eniac (*Eletronic Numerical Integrator and Calculator*), construído por John Von Neuman em 1946 (MATOS/RASKIN, 2009).

O processador é a unidade principal do computador; ele controla o fluxo dos programas, executa operações lógicas e aritméticas, acessa a memória, faz solicitações aos periféricos, confunde-se com a CPU (MATOS/RASKIN, 2009).

Em 1971, na Intel Corporation, Ted Hoff construiu um processador que tinha todas as unidades reunidas em um só chip, o 4004, o primeiro microprocessador (MATOS/RASKIN, 2009).

A diferença básica entre o processador tradicional e o microprocessador é o fato de este último poder ser produzido na linha de montagem, em larga escala, diminuindo drasticamente o custo por causa do preço e do pouco calor dissipado; os microprocessadores se espalharam pelo mundo, conquistaram o mercado. (MATOS/RASKIN, 2009).

A idéia original do projeto RISC, de produzir máquinas com um conjunto reduzido de instruções, é, em última análise, uma volta ao início da computação, pois os primeiros computadores digitais tinham poucas instruções. (MATOS/RASKIN, 2009).

Um projeto de pesquisa da IBM identificou que a maioria das instruções eram usadas com pouca freqüência. Cerca de 20% delas eram usadas 80% das vezes. Os próprios desenvolvedores de sistemas operacionais habituaram-se a determinados subconjuntos de instruções, tendendo a ignorar as demais, principalmente as mais complexas (MATOS/RASKIN, 2009).

A proposta RISC foi então implementar todo um conjunto de instruções em um único chip (MATOS/RASKIN, 2009).

Desde o Eniac, os processadores utilizavam conjuntos de instruções (*instruction sets*) bastante complexos. Esse tipo de arquitetura, por sua difícil execução, exige que o processador analise as instruções e execute pequenas sub-rotinas gravadas dentro do próprio processador (MATOS/RASKIN, 2009).

Acreditando que essas sub-rotinas ou microcódigos fossem contraproducentes, Jonh Cocke, da IBM, teve a idéia de construir um processador mais simples, que não necessitasse de microcódigo, deixando o trabalho pesado para os programas. Estava criada a filosofia do computador com conjunto reduzido de instruções (*Reduced Instruction Set Computer - RISC*), um processador menor, mais barato, mais frio. Com o tempo, este termo acabou por se generalizar e denomina todas as máquinas que obedecessem a um conjunto específico de princípios de arquitetura (MATOS/RASKIN, 2009).

Como era preciso identificar os outros computadores não RISC, foi cunhado o termo CISC (*Complex Instruction Set Computer*). Apesar de inventada em 1974, a filosofia RISC só chegou ao mercado em 1985, pelas mãos da *Sun Microsystems*, com o Sparc.

Hoje temos como microprocessadores CISC toda a plataforma Intel, dos tradicionais ao Pentium. Do outro lado estão o consórcio PowerPC, MIPS, HP e Digital, cada qual com seu chip RISC. (MATOS/RASKIN, 2009).

O PowerPC é o maior desafio aos processadores Intel, que detêm 70% do mercado mundial. O DEC Alpha AXP é o processador RISC de mais alto desempenho rodando em 150MHz ou mais. As outras plataformas RISC populares são o SPARC e o PA-RISC, que geralmente rodam em sistemas operacionais baseados em Unix. (MATOS/RASKIN, 2009).

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

2.1 Organização do sistema

Os computadores apresentam cinco unidades essenciais: unidade lógica e aritmética (ULA), a unidade de memória, a unidade de controle, a unidade de entrada e a unidade de saída. Representado na Figura 1.

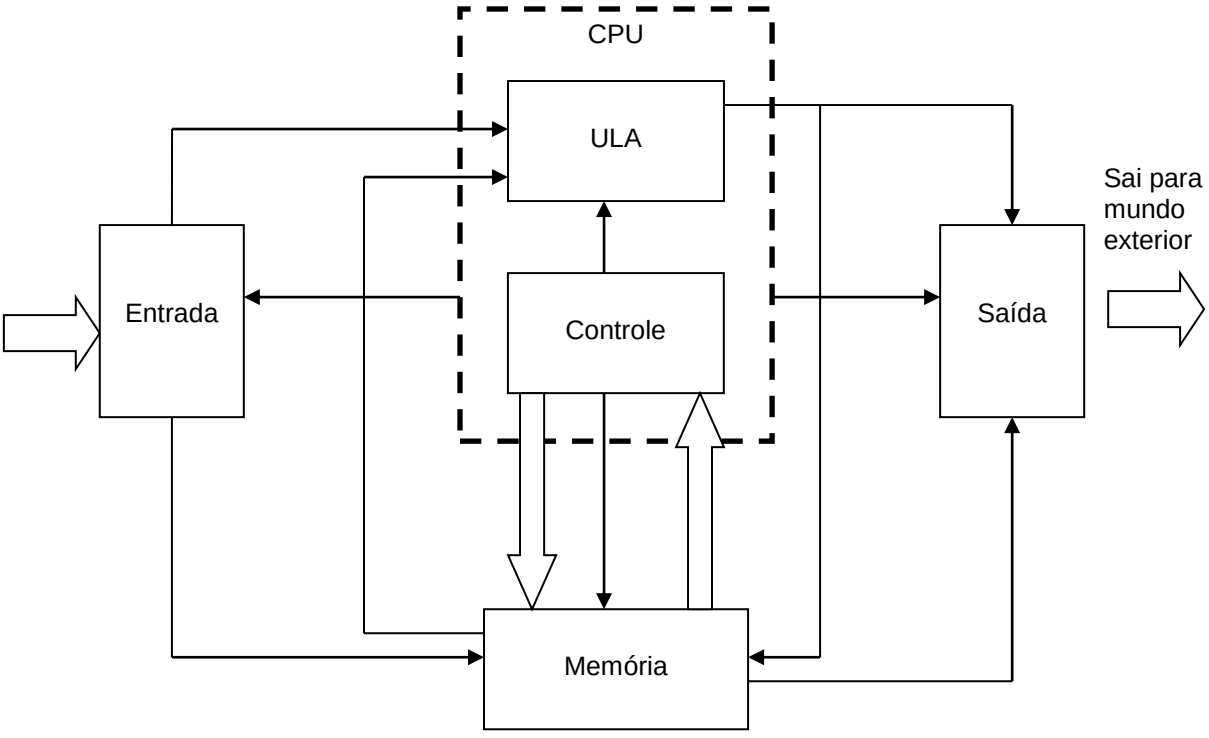


Figura 1 - Organização do sistema

FONTE: O Autor

- Unidade lógica aritmética (ULA) – local onde as operações lógicas e aritmetricas são realizadas.
    - Lógicas: comparação entre maior, menor e igual e booleanas (*And*, *Or*, *Xor*)
    - Aritméticas: Adição, Subtração, Incrementa Multiplicação e Divisão.

- Unidade de memória – a memória armazena grupos de dígitos binários, palavras, que representão instruções que compoêm os programas que o computador executará, manipulando os dados. Além disso a memória também pode armazenar resultados intermediários assim como operações aritméticas. A unidade de controle gerencia a operação da memória que sinaliza uma operação de leitura ou escrita.
    - ROM (*Read Only Memory*) – (memória somente de leitura). Já pelo nome, é possível perceber que esse tipo de memória só permite leitura, ou seja, suas informações são gravadas uma única vez e após isso não podem ser alteradas ou apagadas, somente acessadas. Em outras palavras, são memórias cujo conteúdo é gravado permanentemente (ALECRIM, 2009).
    - RAM (*Random Acess Memory*) – (memória de acesso aleatório). Este tipo de memória permite tanto a leitura como a gravação e regravação de dados. No entanto, assim que elas deixam de ser alimentadas eletricamente, ou seja, quando o usuário desliga o computador, a memória RAM perde todos os seus dados (ALECRIM, 2009).

- Unidade de entrada – utilizada para colocar programas e dados na unidade de memória antes de iniciar o processo . A unidade de entrada consiste em dispositivos que são usados para obter informações e dados externos ao computador e coloca-lo na memoria ou na ULA. A unidade de controle gerencia as informações de entrada.

- Unidade de saída - São dispositivos usados para transferir dados e informações do computador para o mundo exterior. Os dispositivos de saída

são acionados sob o comando da unidade de controle e recebem dados da memória ou da ULA.

- Interfaceamento – Dispositivos de entrada e saída são denominados de periféricos e interfaceamento é especificado como a transmissão digital de informação entre um computador e seus periféricos de modo compativel e sincronizados.

- Unidade de controle – responsável por comandar todas as outras unidades fornecendo sinal de controle e temporização. A unidade de controle busca uma instrução na memória enviando um endereço e um comando de leitura para a unidade de memória. A palavra é armazenada na posição da memória e então transferida para a unidade de controle. A palavra então é codificada pelos circuitos lógicos onde é determinado a instrução a ser invocada. A unidade de controle envia o sinal apropriado para as unidades restantes a fim de executar a operação específica, essa ação é conhecida como busca/executa, como mostra a Figura 2.

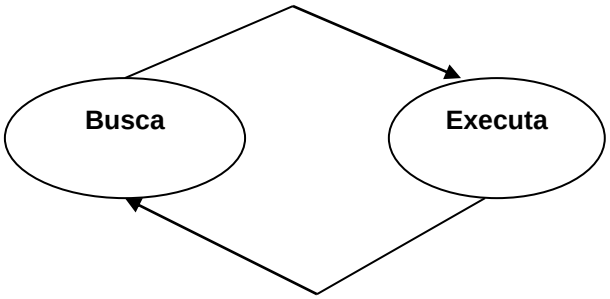


Figura 2 - Unidade de controle

FONTE: O Autor

- Unidade central de processamento (CPU) – A CPU tem um conjunto de registradores que realiza funções específicas. Estes registradores possibilitam o armazenamento temporário dos dados dentro da CPU sem necessidade de acessar a memória externa. A ULA e a unidade de controle são mostradas combinadas em uma unidade denominada CPU.

2.2 Arquitetura RISC

A arquitetura RISC é constituída por um pequeno conjunto de instruções simples que são executadas diretamente pelo hardware, sem a intervenção de um interpretador (microcódigo), ou seja, as instruções são executadas em apenas uma microinstrução (MATOS/RASKIN, 2009).

A arquitetura RISC só veio a s tornar viável com o avanço tecnológico de softwares e o desenvolvimento de novos compiladores otimizados.

Existe um conjunto de características que permite uma definição de arquitetura básica RISC, são elas:

- *Datapath* (ULA, registradores e os barramentos que fazem sua conexão); uma das maiores características das máquinas RISC é utilizar apenas uma instrução por ciclo do *datapath* (uma instrução é similar a uma microinstrução);
- Projeto carrega/armazena, ou seja, as referências à memória são feitas por instruções especiais de *load/store*;
- Inexistência de microcódigo; sendo assim, a complexidade está no compilador;
- Instruções de formato fixo, permitindo uso consistente do formato e facilitando a decodificação de instruções por controle fixo, o que torna mais rápido os dutos de controle;
- Conjunto reduzido de instruções, facilitando a organização da Unidade de Controle de modo que esta tenha uma interpretação simples e rápida;
- Utilização de múltiplos conjuntos de registradores.

A especificação de formatos e o número de instruções são os pontos mais enfocados quando se fala de arquitetura RISC, mas uma generalização bem feita da teoria RISC vai muito além, indicando uma predisposição para estabelecer livremente compromissos de projeto através das fronteiras arquitetura/implementação e tempo de compilação/tempo de execução, de modo a maximizar o desempenho medido em algum contexto específico (MATOS/RASKIN, 2009).

O principal objetivo de uma máquina RISC é executar uma instrução por ciclo; como o acesso à memória utiliza mais ciclos, a solução foi criar um grande número de registradores. Este número de registradores tem grande impacto na performance das máquinas RISC, que só são possíveis devido sua simplicidade de projeto (MATOS/RASKIN, 2009).

O crescimento do número de modos de endereçamento em uma máquina implica na perda de velocidade e também no aumento da complexidade. Porém, apesar do nome, uma máquina RISC poderia ter muitas instruções, contanto que elas executassem em um único ciclo de via de dados e tivessem o formato fixo. O comprometimento nesse caso seria o de construir uma unidade de decodificação bastante complexa, já que ele cresceria exponencialmente com o número de instruções (SICA, 1999).

2.3 Kit de Desenvolvimento DE2

O kit de desenvolvimento da Altera DE2, figura 3, possui caracteristicas que possibilitam a criação e implementação de uma vasta gama de aplicações, desde simples circuitos até projetos muito complexos multimidia.

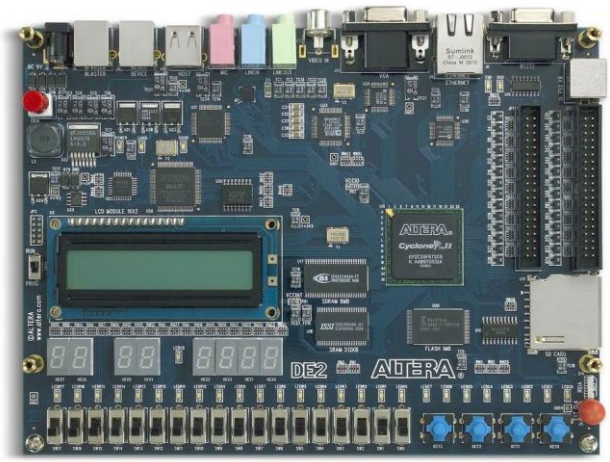


Figura 3 – Kit DE2 de Desenvolvimento (ALTERA, 2009)

A placa do kit de Desenvolvimento DE2 da Altera é composta pelos seguintes componentes (ALTERA, 2009):

- Altera Cyclone II EP2C35F6722C6 FPGA
- Altera EPCS16 – Dispositivo de configuração serial
- USB Blaster para programação
- 512 Kbytes SRAM (*Static Random Access Memory*)
- 8 Mbytes SDRAM (*Synchronous Dynamic Randow Access Memory*)
- 4 Mbytes para *Flash Memory*
- Soquete para SD *Card*
- Botões
- Interruptores
- LEDs (*Light-Emitting Diode*) vermelhos
- LEDs verdes
- Osciladores de 50 Mhz, 27 Mhz e conector para oscilador externo
- CODEC (*Coder-Decoder*) de áudio de 24 bits
- VGA (*Video Graphics Array*) DAC (*Digital-to-analog Converter*) de 10 bits
- TV Decoder (NTSC/PAL)
- Controlador Ethernet 10/100
- Controlador USB com conectores do Tipo A e do Tipo B

- Transceiver IrDA (*Infrared Data Association*)
- Transceiver RS-232
- Conector PS/2
- Dois barramentos expansores de 40 pinos com proteção de diodos
- Display LCD de 2 linhas

2.4 O Dispositivo FPGA

O FPGA (Field Programmable Gate Array) é um circuito integrado passível configuração por *software* e serve para implementar circuitos digitais, como processadores, interfaces, controladores e decodificadores. Basicamente, consiste de um arranjo fortemente condensado de blocos idênticos de pequenos circuitos, compostos por algumas portas lógicas e *flip-flops*, com alguns sinais de interface. As conexões entre as saídas de determinados blocos com as entradas de outros são programáveis através de um protocolo simples e de fácil implementação (MENDONÇA/ZELENOVISKY, 2008).

O FPGA possui três conjuntos de elementos de configuração. O primeiro consiste de vários circuitos idênticos, compostos por alguns *flip-flops* e lógica combinacional extra, sendo conhecidos por CLBs (*Configuration Logical Blocks*). Os CLBs possuem os elementos funcionais para a construção de lógicas pelo usuário. Geralmente, são constituídos por um conjunto de 2 a 4 flip-flops e por lógicas combinacionais. O segundo consiste de circuitos de interfaceamento das saídas dos CLBs com o exterior do FPGA, chamados de IOBs (*Input Output Blocks*). Eles são constituídos por buffers bidirecionais com saída em alta-impedância. Através da programação adequada de um IOB, configura-se um pino do FPGA para funcionar como entrada, saída, bidirecional ou coletor-aberto. O terceiro grupo é composto pelas interconexões. Os recursos de interconexões possuem trilhas para conectar as entradas e saídas dos CLBs e IOBs para as redes apropriadas. Geralmente, a configuração é estabelecida por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA entre os CLBs e os IOBs. O processo de escolha das interconexões é chamado de roteamento (MENDONÇA/ZELENOVISKY, 2008).

Para melhor visualização da FPGA a Figura 4 mostra sua arquitetura básica.

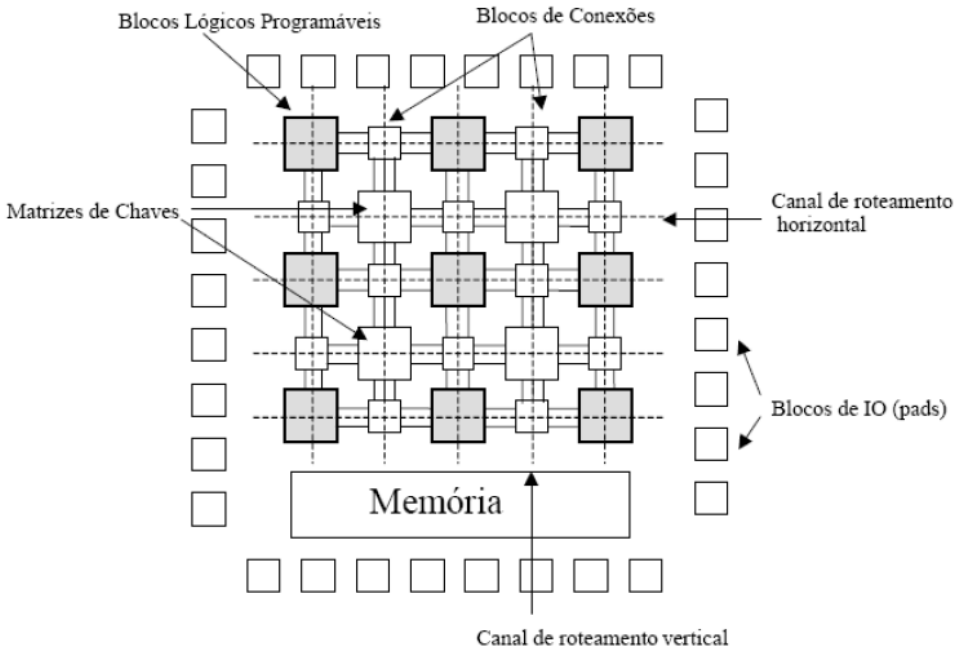


Figura 4 - FPGA Arquitetura Básica (JUNQUEIRA, 2007).

2.5 VHDL/AHDL

O VHDL/AHDL são formas de se descrever, através de um programa, o comportamento de um circuito ou componente digital (GIACOMINI, 2000).

A história do VHDL começo nas décadas de 70 e 80 onde foi posto em prática um programa do Departamento de Defesa (DoD) americano, para desenvolvimento de circuitos integrados de alta velocidade, denominado VHSIC e originado da preocupação de domínio das tecnologias envolvidas. Desde o princípio, anteviu-se o problema de representação dos projetos segundo uma linguagem que fosse comum às várias empresas envolvidas e que permitisse uma documentação fechada e clara para projetos de complexidade crescente. Nesta mesma época, já havia no mercado de computação a chamada crise do *software*, caracterizada pela dificuldade de gestão, documentação e sistematização do ciclo de vida do *software* (que envolvem, em termos gerais, todas as atividades de relativas à sua criação e uso). O problema surgiu inicialmente no *software* porque era nele que se desenvolviam as aplicações, que tornavam o produto particular para um uso específico. Isso fazia com que um grande volume de projetos fossem conduzidos, quase sempre com muita complexidade e pouco método. No hardware a questão era mais bem controlada, pois os projetos eram, em geral, mais genéricos, especialmente para o *hardware* digital (GIACOMINI, 2000).

Os projetos de microprocessadores, que eram os mais complexos, visavam na maior parte dos casos o uso geral. Um mesmo microprocessador seria utilizado em várias aplicações, seria reproduzido e testado aos milhões. Isso significava que o investimento em um único projeto poderia ser muito maior, inclusive em termos de especificação e testes. A evolução da metodologia de projeto de *software* foi então natural, em função dos argumentos sustentados pela crise. Hoje ainda é grande o desenvolvimento de novos métodos e a busca por uma linguagem descritiva universal, como o UML, para especificação e projeto de software orientado a objetos (GIACOMINI, 2000).

Em 1987, o VHDL foi normalizado pelo IEEE, tornando-se um padrão mundial, ao lado do Verilog, uma alternativa também expressiva no mercado de projetos de hardware. Hoje, praticamente todas as ferramentas de desenvolvimento de hardware computadorizadas aceitam essas linguagens como entrada, de forma que um projeto baseado em VHDL ou Verilog pode ser implementado com qualquer tecnologia (GIACOMINI, 2000).

O VHDL apresenta as seguintes vantagens:

- Código facilmente reutilizável;
- Projeto independente da tecnologia;
- Redução do tempo de projeto;
- Eliminação de erros de baixo nível;
- Portabilidade.

Suas principais desvantagens:

- Hardware gerado é menos otimizado;
- Nem todos os comandos são sintetizáveis.

2.6 Quartus II

O ambiente de desenvolvimento utilizado é o Quartus II que se mostra prático e fácil para o desenvolvimento e edição em AHDL/VHDL.

O Quartus II é uma ferramenta voltada para desenvolvimento de projetos em componentes de alta densidade. O software oferece recursos e metodologias para implementação e gerenciamento de projetos.

A Figura 5 mostra a janela do Quartus II.

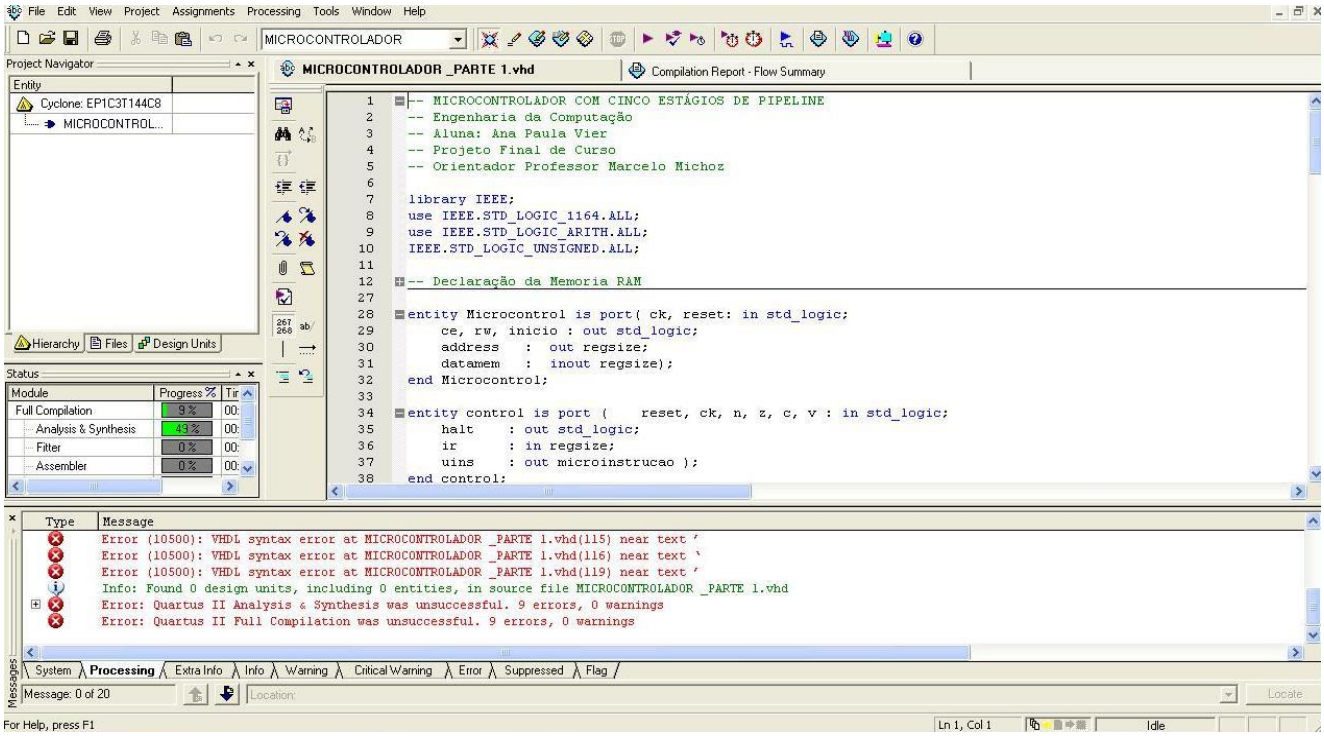


Figura 5 - Área de desenvolvimento do Quartus II

FONTE: O Autor

CAPÍTULO 3 – ESPECIFICAÇÃO DO PROJETO

3.1 O projeto

A implementação deste projeto é baseada num processador RISC desenvolvido em AHDL/VHDL no Quartus II. O objetivo é o desenvolvimento de uma descrição do processador RISC com comunicação com o PC via interface de entrada (teclado e/ou mouse).

A Figura 6 mostra a Arquitetura básica de um processador RISC.

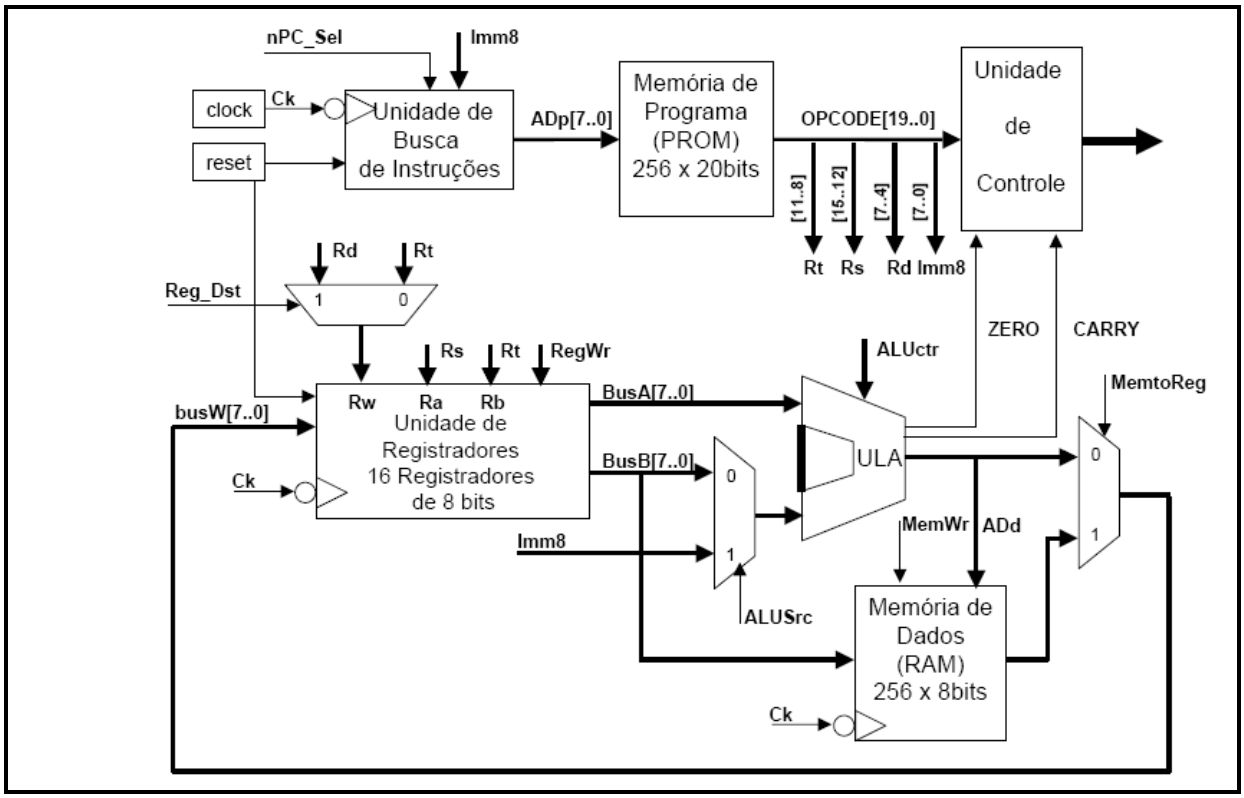


Figura 6 – Estrutura do Processador Usada como Base no Projeto (JÚNIOR, 2006)

Para ser verificado o funcionamento do projeto em sua etapa final, basta seguirmos uma sequência de etapas, como mostra a figura 7:

1. Conectar ao Kit de desenvolvimento DE2 um Computador com o Quartus II previamente instalado e efetuar a gravação do processador na FPGA do Kit.
2. Conectar ao Kit um Teclado PS2 convencional, através da porta específica presente no Kit, para interação com o programa.
3. Visualização da simulação da porta paralela pelas Chaves e LED's
4. Resultado das operações efetuadas pelo teclado visualizado no display LCD.

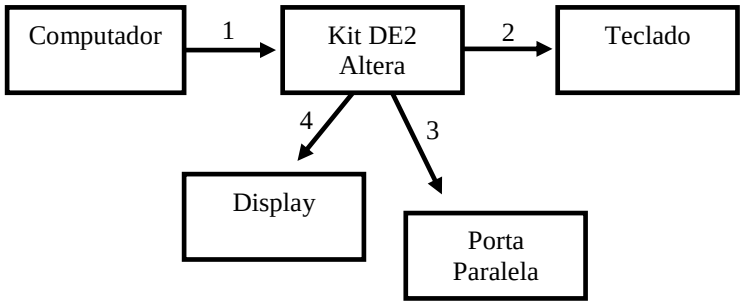


Figura 7 – Diagrama de Funcionamento do Projeto
FONTE: O Autor

3.2 Hardware

A implementação desenvolvida neste projeto é um processador com arquitetura RISC, o diagrama da figura 6 mostra a estrutura lógica do mesmo, embarcado no FPGA presente no Kit de Desenvolvimento DE2.

A principal característica da arquitetura RISC é o fato de utilizar apenas uma instrução por cilco de *clock*. As principais características da arquitetura RISC, segundo Matos e Raskin (2009), são:

- *Datapath* (ULA, registradores e os barramentos que fazem sua conexão); uma das maiores características das máquinas RISC é utilizar apenas uma instrução por ciclo do *datapath* (uma instrução é similar a uma microinstrução);

- Projeto carrega/armazena, ou seja, as referências à memória são feitas por instruções especiais de *load/store;*
- Instruções de formato fixo, permitindo uso consistente do formato e facilitando a decodificação de instruções por controle fixo, o que torna mais rápido os dutos de controle;
- Conjunto reduzido de instruções, facilitando a organização da Unidade de Controle de modo que esta tenha uma interpretação simples e rápida;
- Utilização de múltiplos conjuntos de registradores.

3.3 Estágio de Busca

O Estágio de busca, Figura 8, é responsável pelo fornecimento de instruções para o processador RISC, a partir da ordem definida no programa. Para determinar o valor da próxima instrução deve verificar se, conforme Brisolara e Pilla (2000):

- A próxima instrução a ser buscada segue a última instrução no código do programa, onde é seqüencial, e o PC é incrementado de 4 em 4.
- A próxima instrução a ser buscada é determinada por uma instrução de desvio.
- Observa-se que o segundo item é necessário esperar pelo resultado do cálculo do desvio, no caso de um desvio condicional.

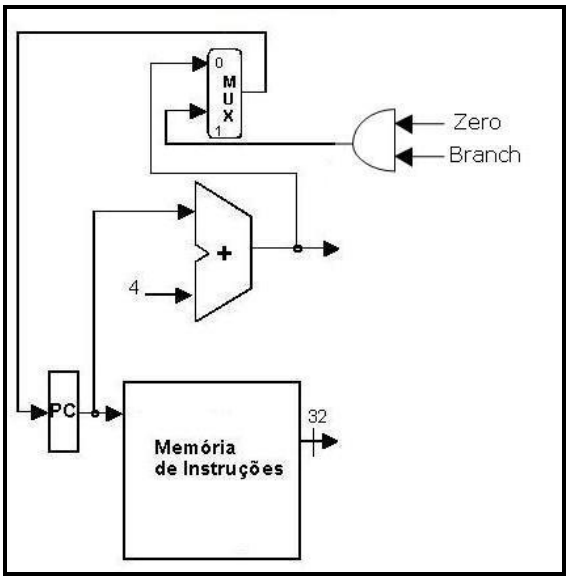


Figura 8 - Estágio de Busca (BRISOLARA/PILLA, 2000)

Na Figura 8, observa-se que a fonte do próximo PC (*Program Counter*) é determinada pelo AND entre Zero e *Branch* (BRISOLARA/PILLA, 2000).

O sinal Zero determina se o resultado da ULA na última operação foi zero. O sinal *Branch* depende da instrução no estágio de Execução, sendo 1(um) quando esta for um desvio. Assim, quando estes dois sinais são iguais a 1(um), o resultado da soma do próximo PC com o deslocamento na instrução de desvio determina o valor a ser armazenado no PC (BRISOLARA/PILLA, 2000).

Em caso contrário, o PC recebe o endereço da próxima instrução no fluxo seqüencial de instruções (PC+4). O somador para calcular o PC+4 encontra-se dentro do estágio de Busca (BRISOLARA/PILLA, 2000).

3.4 Estágio de Decodificação

O estágio de Decodificação é responsável pela interpretação das instruções e pelo gerenciamento do banco de registradores (BRISOLARA/PILLA, 2000).

A Figura 9 mostra os componentes do estágio de Decodificação. Além do banco de registradores com duas leituras e uma escrita por ciclo, implementado como um componente parametrizado (BRISOLARA/PILLA, 2000).

O banco de registradores deve ser implementado como uma entidade separada, podendo ser determinado na instanciação o número de registradores. Cada registrador é inicializado com o valor correspondente ao seu índice no banco de registradores, para diminuir o tamanho necessário aos programas de teste (BRISOLARA/PILLA, 2000).

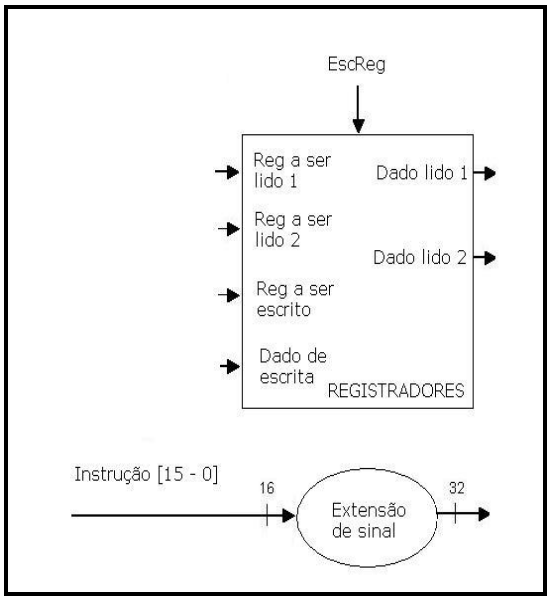


Figura 9 - Estágio de Decodificação (BRISOLARA/PILLA, 2000)

3.5 Estágio de Execução

No estágio de execução, Brisolara e Pilla (2000), verificaram duas unidades de execução:

- Unidade para cálculo de endereço de *branch* (apenas soma)
- Unidade Lógica e Aritmética (*and*, or, add e sub).

Os sinais de controle dos multiplexadores são detalhes importantes. O sinal ALUscr é gerado pelo bloco de controle para selecionar o segundo operando da ULA,

este pode ser o dado lido de um registrador, no caso das instruções *Rtype* e de *branch* ou um endereço no caso das instruções de *load* e *store* (BRISOLARA/PILLA, 2000).

Os outros dois multiplexadores servem para permitir o *forwarding* dos dados. Agora, além do dado lido dos registradores, o operando da ULA pode ser o resultado de um estágio de execução de uma instrução anterior que escreve em um registrador que a instrução que está sendo executada lê, permitindo que o valor possa ser repassado para instrução corrente antes do estágio de escrita dos registradores (*write-back*) da instrução anterior ocorra, ou ainda o resultado de uma busca na memória no qual o registrador destino é utilizado pela instrução corrente. Os sinais de controle dos multiplexadores, AluSelA e AluSelB, são gerados. Na figura 10 tem os detalhes do estágio de execução (BRISOLARA/PILLA, 2000).

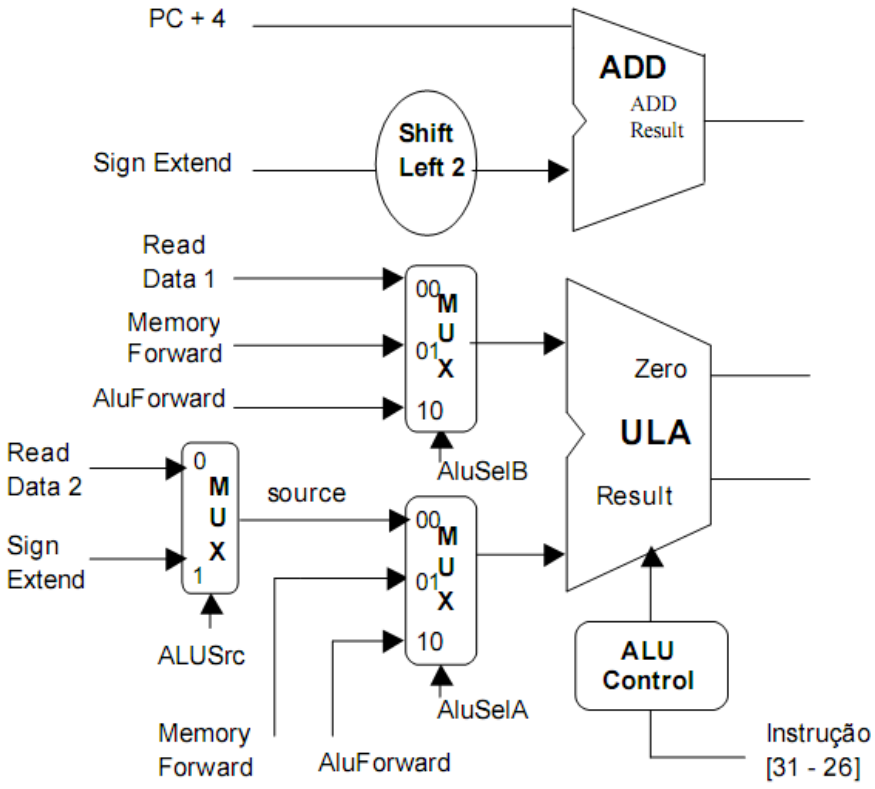


Figura 10 - Estágio de Execução (BRISOLARA/PILLA, 2000).

3.6 Controle da ULA

Para controlar a ULA (Unidade lógica e aritmética), tem como entrada o opcode, onde os quatro primeiros bits mais significativos da palavra de instrução e mais os quatro bits menos significativos da instrução, gerando a partir destes um sinal de quatro bits para a ULA. Este sinal, denominado func, determina qual a operação que a ULA executará (BRISOLARA/PILLA, 2000).

Na Tabela 1, pode-se observar os códigos e suas operações correspondentes executadas pela ULA da estrutura do processador usada como base (BRISOLARA/PILLA, 2000).

Tabela 1 – Operação da ULA

Código	Operação
0000	AND
0001	OR
0010	ADD
0011	SUB
0111	**

Fonte: (BRISOLARA/PILLA, 2000).

Na Tabela 2, observa-se a tabela-verdade para a geração do sinal de função da ULA a partir de um sinal conhecido como ALUOp, gerado do opcode, e dos 4 bits menos significativos da instrução, utilizados apenas para as instruções lógicas e aritméticas.

Analisando a tabela 2, obtem-se o circuito de controle da ULA.

Tabela 2 – Tabela verdade para operação da ULA

ULAop		**Instrução lógica e aritmetica**								**Operação ULA**			
ALUop1	**ALUop2**	**F7**	**F6**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**				
0	0	X	X	X	X	X	X	X	X	0	1	0	1
X	1	X	X	X	X	X	X	X	X	1	1	0	1
1	X	X	X	0	0	0	0	1	0	0	0	1	0
1	X	X	X	0	0	0	1	0	1	1	0	1	1
1	X	X	X	0	1	0	0	0	1	0	1	0	1
1	X	X	X	0	1	1	1	0	1	1	1	0	0
1	X	X	X	1	0	1	0	1	0	0	0	1	0
1	X	X	X	1	0	1	1	1	0	1	0	1	1
1	X	X	X	1	1	0	0	1	0	0	1	0	1

Fonte: (BRISOLARA/PILLA, 2000).

3.7 Sinais de Interface

A tabela 3 descreve alguns dos principais sinais de interface do projeto.

Tabela 3: Sinais de controle

Sinais de Controle	Descrição
EscReg	Sinal que habilita a escrita o banco de registradores
MenToReg	sinal que indica que o dado a ser escrito no registrador de destino é um dado vindo da memória;
AluSrc	sinal que indica qual o segundo operando da ULA, podendo este ser o valor lido do segundo registrador, ou um deslocamento de 16 bits, vindo da palavra de instrução e estendido para 32 bits que servirá para o cálculo do endereço para as instruções de load ou store.
EscMem	sinal que habilita a escrita na memória, gerado na instrução de store.
LerMem	sinal que habilita a leitura da memória, gerado na instrução de load.
Branch	sinal que indica uma instrução de branch e juntamente com o sinal de zero obtido após o estágio de execução determina se ocorrerá ou não o salto.
SelReg	sinal utilizado para selecionar o registrado destino.

Fonte: (BRISOLARA/PILLA, 2000).

3.8 Estágio de Memória

O estágio de Memória, Figura 11, implementado através de um componente lpq_ram_qd da Altera, permite a leitura e a escrita no mesmo ciclo. A escrita é habilitada apenas no final do ciclo e somente se o sinal EscMem, que indica instrução de *store*, estiver ativo (BRISOLARA/PILLA, 2000).

A memória de dados foi implementada através de um bloco lpm_ram_dq da Altera. Este bloco permite dois acessos de leitura e um acesso de escrita no mesmo ciclo (BRISOLARA/PILLA, 2000).

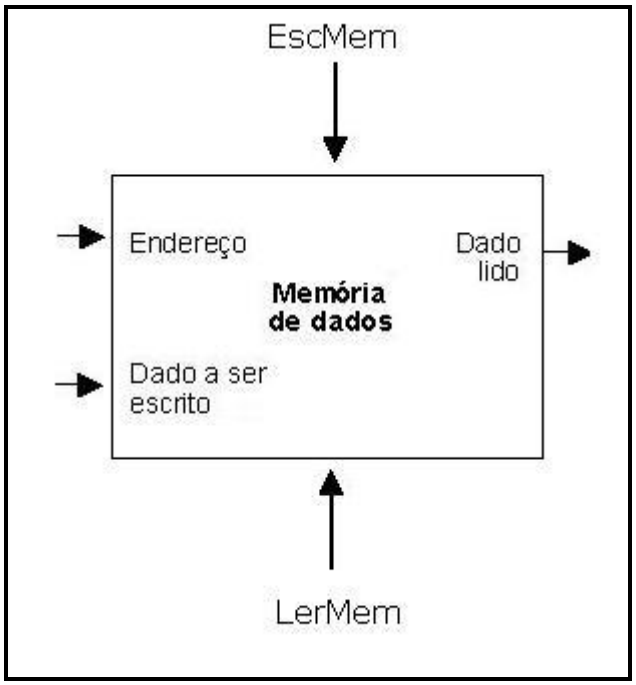


Figura 11 - Estágio de Memória (BRISOLARA/PILLA, 2000)

3.9 Estágio WriteBack

O estágio de *Writeback* alimenta o estágio de decodificação com o valor a ser escrito no banco de registradores. Quando o dado a ser escrito é resultado de uma instrução de leitura da memória (LerMem), o dado proveniente da memória é selecionado. Caso contrário, será passado para o estágio de decodificação o resultado da ULA (BRISOLARA/PILLA, 2000).

3.10 Requisitos de Funcionamento

Software:

- Altera Quartus II 8.1 *Web Edition* ou superior
- Sistema operacional Windows XP/Vista

Hardware:

- Microcomputador com 1GHz (ou superior)
- 256 Mb de memória (ou mais)
- 4Gb de espaço disponível

O material utilizado para o desenvolvimento do projeto:

- Kit FPGA – Altera DE2
- Software Quartus II 8.1 *Web Edition*
- Microcomputador AMD Athon64 X2 5200+ com 4Gb de memória e 500Gb de espaço disponível

3.11 Viabilidade Técnico-econômica

Para o desenvolvimento do projeto, os principais recursos foram disponibilizados pelo Curso de Engenharia da Computação. Porém para contabilizar os custos gerados pelo projeto foram listados na Tabela 4, sem levar em consideração ferramentas de desenvolvimento como computador e softwares utilizados. Foi considerado o tempo de trabalho durante o desenvolvimento do projeto, desde a pesquisa até a conclusão da documentação.

Tabela 4 – Custos do Desenvolvimento do Projeto

Descrição	**Quantidade**	**Valor Unitário**	**Total**
Horas de trabalho	650	R$ 25,00	R$ 16.250,00
Kit DE2 Altera	1	R$ 1.800,00	R$ 1.800,00
Teclado PS2	1	R$ 20,00	R$ 20,00
Total de Mão-de-obra			R$ 16.250,00
Total de Materiais			R$ 1.820,00
Total de Custos do Projeto			**R$ 18.070,00**

Fonte: (O Autor)

CAPÍTULO 4 – DESENVOLVIMENTO E IMPLEMENTAÇÃO

O objetivo da implementação desse projeto é criar um processador RISC capaz de executar programas em Assembly desde que sejam utilizadas as funções definidas em sua unidade de controle. O processador deve ter uma interface de entrada, no caso o teclado, para interagir com o programa, que deverá mostrar seus resultados no display, presente no Kit de desenvolvimento DE2 da Altera, e simular uma porta paralela, o que é feito através das chaves e *LED's* também presentes no kit. Para o funcionamento correto do projeto foram desenvolvidos alguns novos blocos no processador RISC usado como base, bem como algumas mudanças na arquitetura do mesmo.

4.1 Principais Módulos de Hardware Utilizados

4.1.1 Módulo Display LCD

Para que fosse possível a leitura dos dados do processador no display LCD, foi utilizado um *driver* de inicialização do mesmo elaborado por Navalbi (2007), Figura 12, com algumas alterações para o funcionamento correto no projeto. O display presente no kit é o CFAH160B-TMC-JP da *Crystalfontz America.*

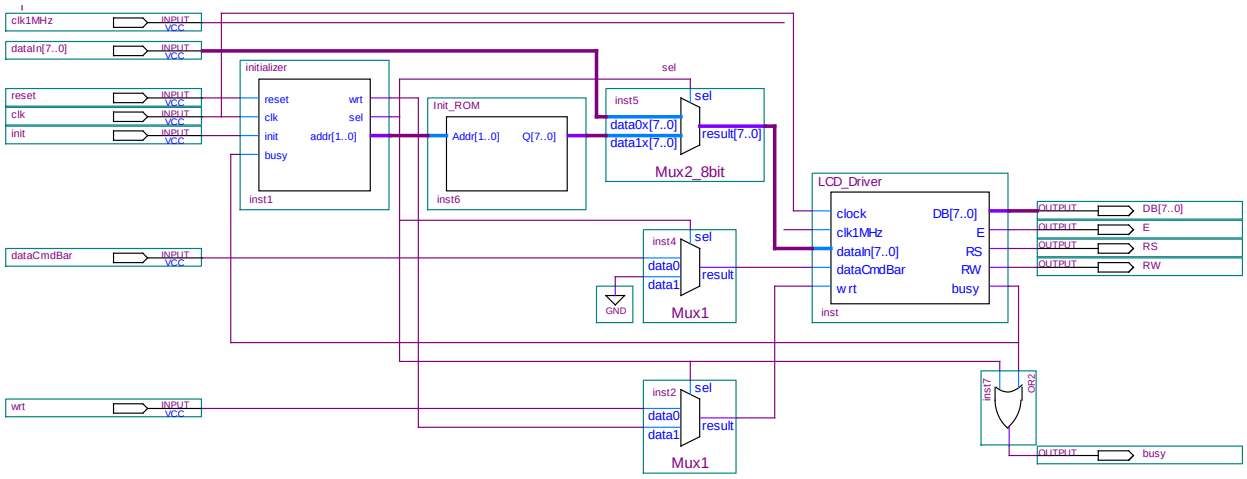


Figura 12 – Esquematico do Driver de Inicialização do Display LCD (NAVALBI, 2007)

Após criado o *driver* LCD constatou-se que para que a leitura fosse possível no display seria necessário diminuir o *clock* que chegava do processador. Foi desenvolvido então um divisor de *clock* acoplado no *driver* de inicialização do display LCD.

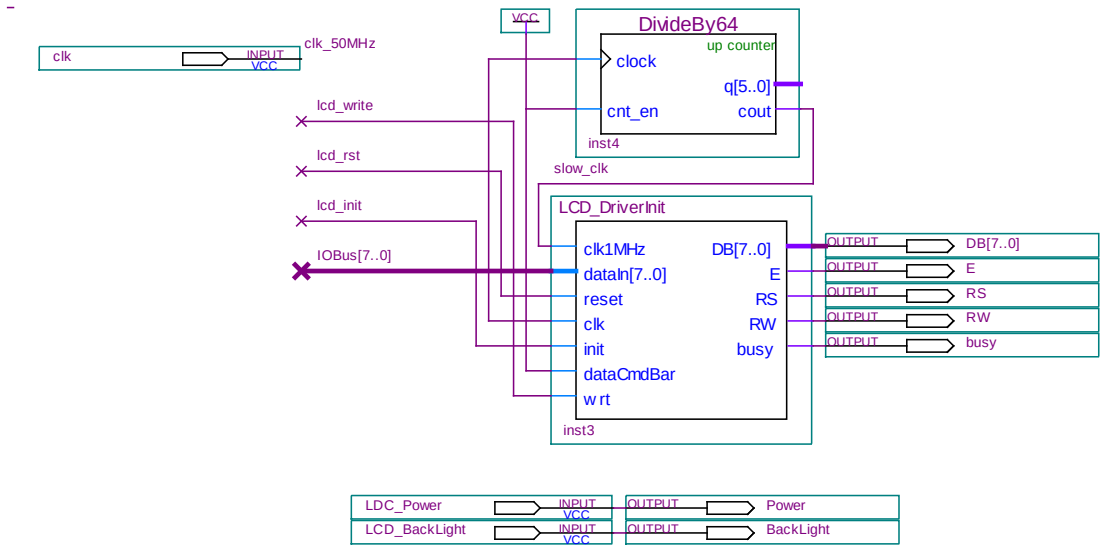


Figura 13 – Divisor de *Clock* acoplado ao Módulo LCD (NAVALBI, 2007 – Adaptado)

4.1.2 Módulo Teclado

Para que fosse possível a interação do usuário com o programa utilizado para o processador RISC um *driver* para o teclado, figura 14, desenvolvido por Navalbi (2007)

com algumas adaptações. Com o *driver* acoplado ao processador é possível utilizar qualquer botão de um teclado PS2 padrão conectado ao kit de desenvolvimento DE2 da Altera no programa de teste.

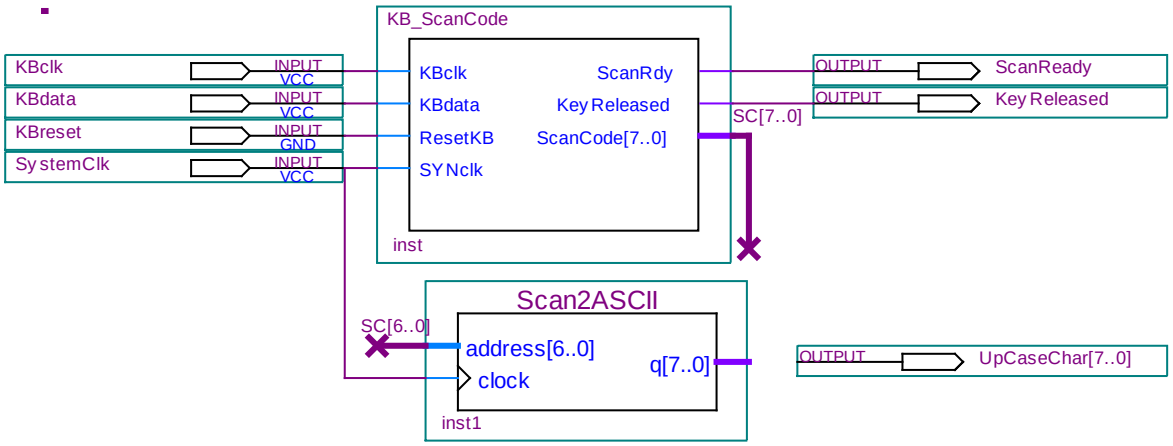


Figura 14 – Esquemático Driver do Teclado (NAVALBI, 2007)

4.1.3 Módulo Porta Paralela

Para simular a utilização de uma porta paralela no processador foram utilizadas as chaves e *LED's* presentes no kit de desenvolvimento DE2 da Altera. Para tornar essa ação possível foram adicionados alguns blocos no processador, Figura 15, o qual aciona as Chaves e *LED's* correspondentes no kit de desenvolvimento DE2 da Altera.

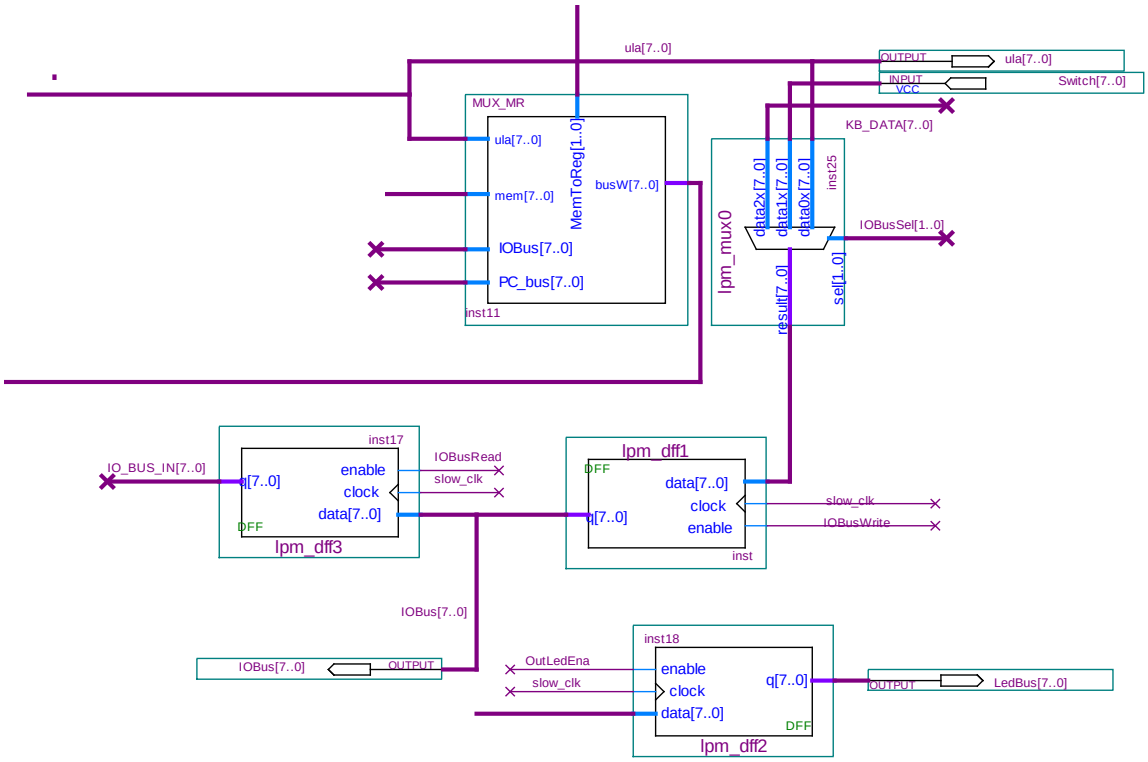


Figura 15 – Blocos de leitura das chaves e Led's para simulação da porta paralela

4.2 Memórias RAM e ROM

A memória RAM do processador foi configurada para conter uma capacidade de 256 palavras de 8 bits. Como mostra a Figura 16

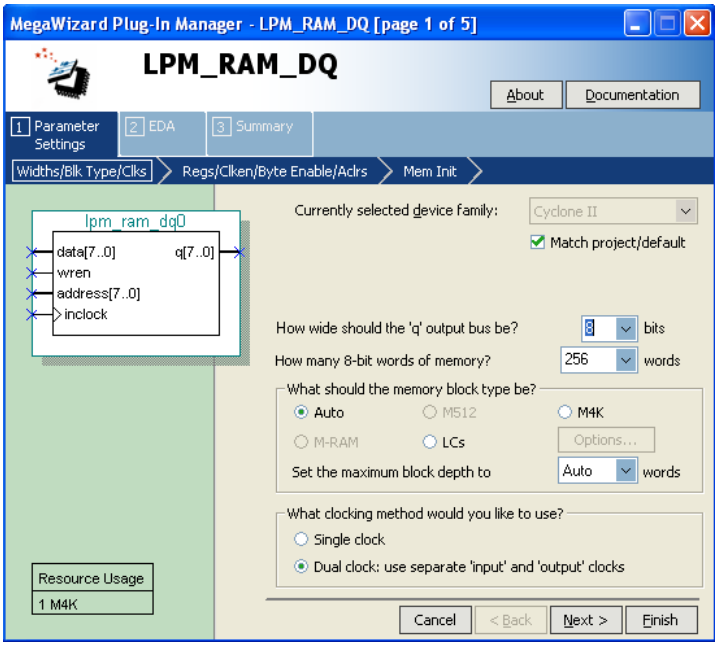


Figura 16 – configuração da memória RAM

Dentro da memória ROM é gerado o programa em Assembly a ser executado pelo processador RISC, o processador tem capacidade de gravar um programa de até 256 linhas de código. A Figura 17 mostra a configuração utilizada na memória ROM.

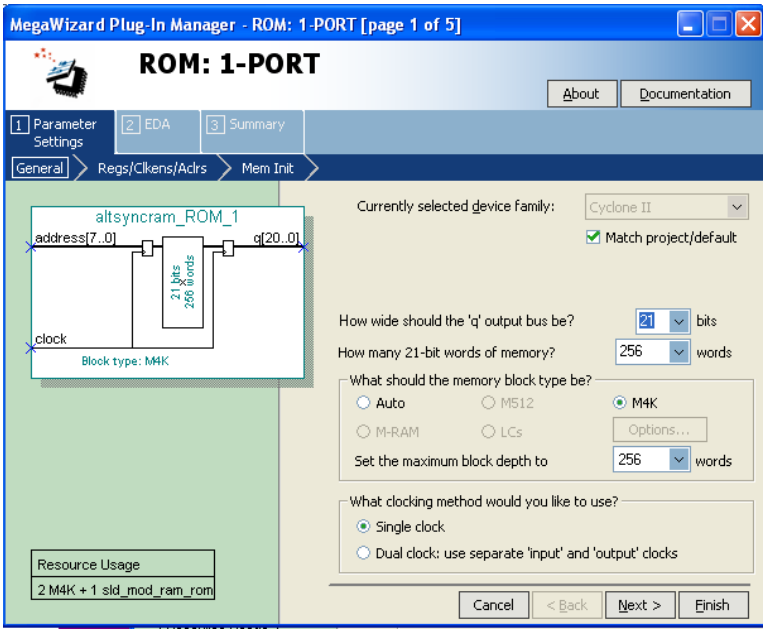


Figura 17 – configuração da memória ROM

4.3 Unidade de Controle

Se o processador é o coração do computador, pode-se dizer que a Unidade de Controle é o cérebro do processador. É ela quem organiza todas as ações que serão realizadas pelo processador. Também é nela que se tem a organização de todos os OpCode's que pode-se utilizar no desenvolvimento de um programa a ser executado pelo processador. O processador usado como base apresentava 4 bits de entrada na Unidade de Controle, dessa forma só era possível utilizar de 0000 a 1111 (em binário), ou seja, até 15 OpCode's. Foi alterado o barramento de entrada de dados para 5 bits de entrada na Unidade de Controle, assim podem ser utilizados até 31 Opcode's.

A Tabela 5 mostra quais os OpCode's temos disponíveis na Unidade de Controle.

Tabela 5 – OpCode's disponíveis na Unidade de Controle

OpCode	Função	Descrição
00000	ADD	Operação de "Adição"
00001	SUB	Operação de "Subtração"
00010	OR	Operação "Ou"
00011	AND	Operação "E"
00100	LOAD	Carrega algum dado
00101	STORE	Armazena algum dado
00110	BRANCH_EQ (IGUAL)	Efetua uma comparação buscando uma igualdade com zero
00111	BRANCH_MAIOR	Efetua uma comparação para ver se o valor é maior do que 10
01000	JUMP	Pula para o um endereço de memória
01001	LCD RESET	Reinicia o LCD
01010	LCD INIT	Inicia o LCD
01011	LCD WRITE	Escreve no LCD
01110	MOVE	Move algum dado
01111	RET	Retorno
10000	CALL	Chama alguma função
10001	BRANCH_KBOARD	Faz a comparação dos sinais das tecals do teclado
10010	IOBusRead	Faz a leitura do barramento de IO
10011	IOBusWriteReg	Escreve no barramento de IO
10100	IOBusWriteSwitch	Escreve no barramento das Chaves
10101	IOBusWriteKBoard	Escreve no barramento do Teclado
10110	LedWrite	Faz a leitura dos LED's

4.4 O Esquemático Final do Quartus II

Para a conclusão do processador no Quartus II foi necessário a criação de módulos auxiliares como visto no tópico 4.1. Depois de finalizado os testes os módulos foram adicionados ao escopo do processador.

A figura18 mostra como ficou o Esquemático Final do Projeto no Quartus II com a integração dos módulos apresentados.

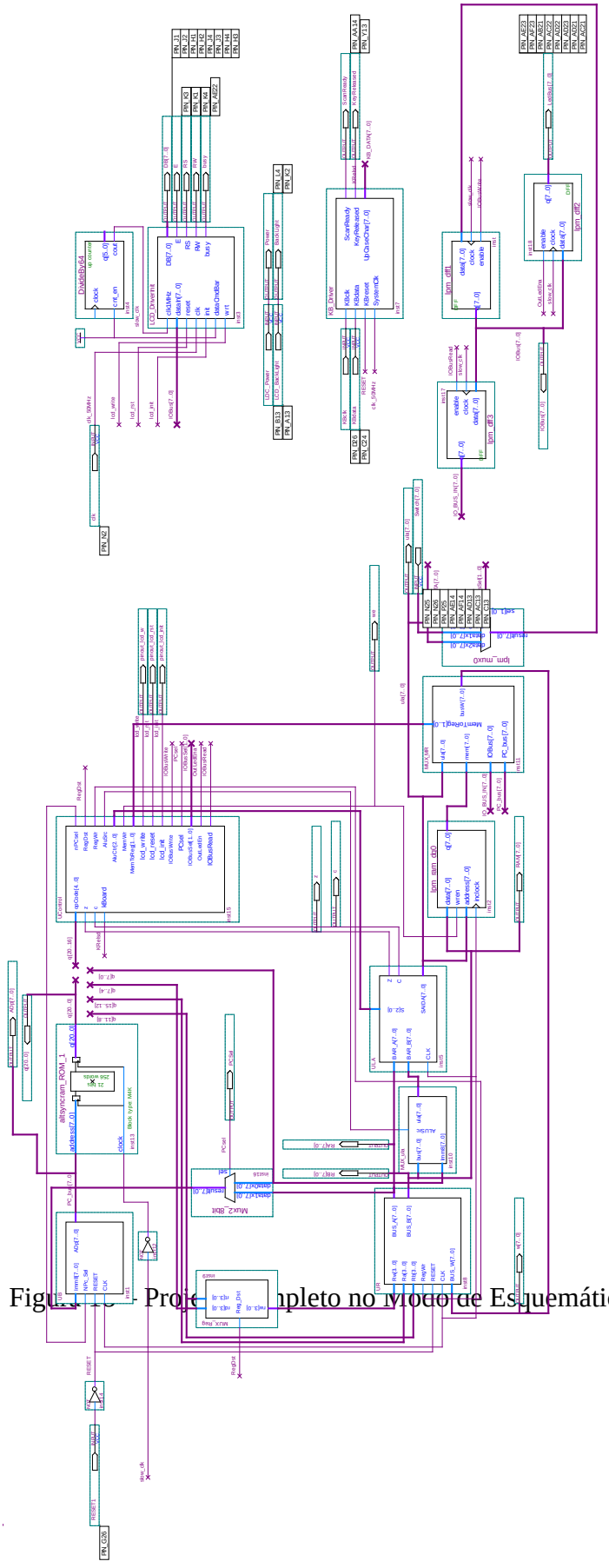


Figura 4.2 - Projeto completo no modo de Esquemático do Quartus II

CAPÍTULO 5 – VALIDAÇÃO DOS RESULTADOS

Os resultados finais obtidos no projeto foram totalmente satisfatórios, tendo em vista que todas as funcionalidades propostas originalmente foram alcançadas com sucesso. O processador foi testado de duas formas, primeiramente através de simulações, realizadas no Quartus II.

No início da simulação, Figura 19, os valores devem aparecer zerados, com modificações dos sinais no decorrer da mesma. Para efetuarmos a avaliação da simulação devemos fixar valores para o programa, a assim verificar os valores obtidos.

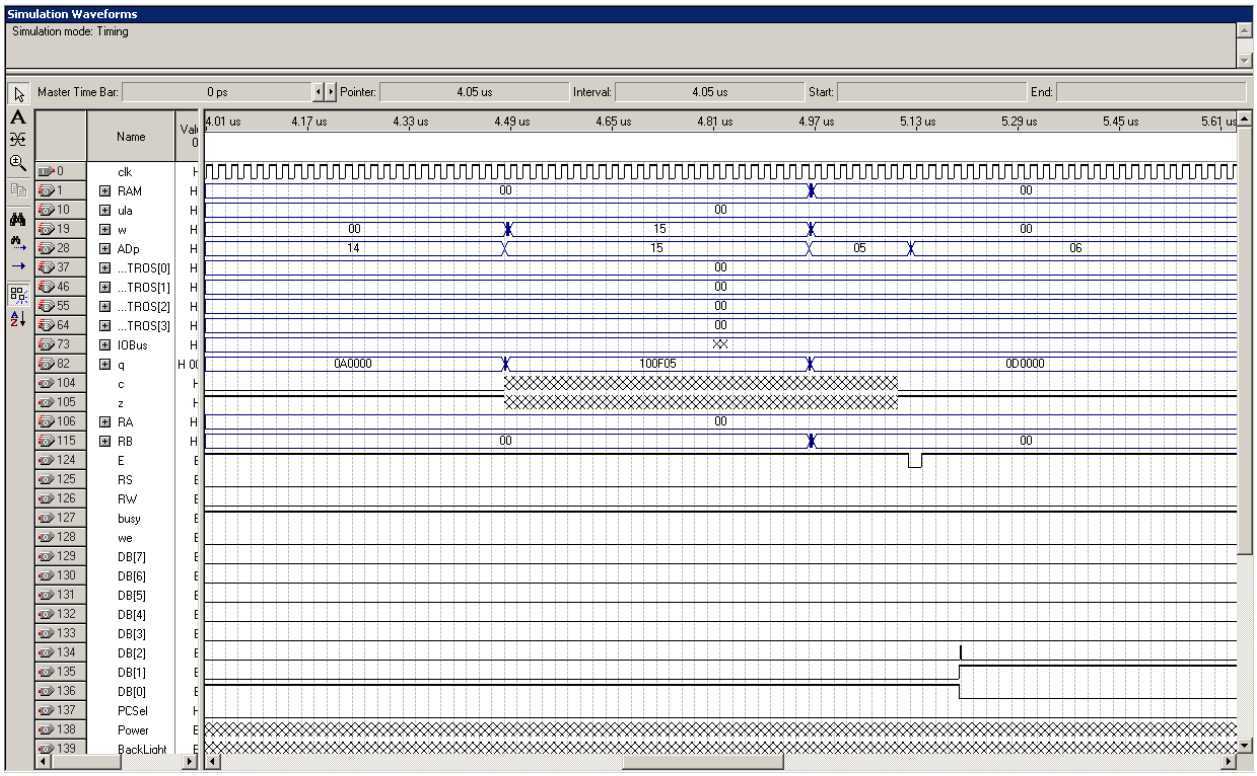


Figura 19 – tela de simulação no Quartus II

A Figura 20 mostra o projeto montado em seu estado final.

Figura 20 – projeto montado em sua fase final

Foi necessário criar um programa em Assembly para testar o funcionamento do processador. Esse programa foi gravado diretamente na memória ROM. Como o programa a ser executado não faz parte do escopo do projeto optou-se por um programa simples, apenas para mostrar o processador funcionando. Abaixo é mostrado o código comentado do programa em Assembly que consiste apenas em efetuar a soma de dois números entre 0(zero) e 9(nove), e simular a porta paralela com as chaves e *LED's* presentes no Kit de Desenvolvimento DE2 da Altera.

```
WIDTH=21;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
        000  :  040000;        --              load R0,RAM[0]
                                                guarda variável da ram0 em no reg R0
        001  :  040202;        --              load R2,RAM[2]
```

```
                                    guarda variável da ram2 em no reg R2
002 :  080010;      --              jump Main
                                    pula para MAIN
003 :  0D0000;      --              NOP
004 :  0D0000;      --              NOP
                    --              DELAY:
005 :  040101;      --              load R1,RAM[1]
                                    guarda variável da ram1 em no reg R1
006 :  06010D;      --              BEQ:  beq R0,R1, RET
                                    compara R0 com R1 se for igual pula pra
                                    RET
007 :  0D0000;      --              NOP
008 :  0D0000;      --              NOP
009 :  011210;      --              sub R1,R1,R2
                                    subtrai R2 de R1 e guarda em R1
00A :  080006;      --              jump BEQ
                                    pula para BEQ
00B :  0D0000;      --              NOP
00C :  0D0000;      --              NOP
00D :  0FF000;      --              RET
                                    retorna para quem chamou delay
00E :  0D0000;      --              NOP
00F :  0D0000;      --              NOP
                    --              MAIN:
010 :  140000;      --              IOBusWrite_Switch
                                    escreve no IObus o que tem nos switchs
011 :  160000;      --              Leds = IOBus
                                    escreve nos leds o que leu dos switchs
012 :  0A0000;      --              LCD init
                                    inicializa LCD
013 :  100F05;      --              call DELAY
014 :  0D0000;      --              NOP
015 :  0D0000;      --              NOP
```

```
016 :  090000;      --          LCD reset
                                reinicia LCD
017 :  100F05;      --          call DELAY
018 :  0D0000;      --          NOP
019 :  0D0000;      --          NOP
                                Agora vai escrever “SOMA:” no LCD

01A :  040403;      --          load R4,RAM[3]
                                guarda variável da ram3 em no reg R4 "S"
01B :  134000;      --          IOBus = R4
                                coloca em IOBUS
01C :  0B0000;      --          LCD write
                                escreve no LCD o que está em IOBUS
01D :  100F05;      --          call DELAY
01E :  0D0000;      --          NOP
01F :  0D0000;      --          NOP
020 :  040404;      --          load R4,RAM[4]
                                guarda variável da ram3 em no reg R4 "O"
021 :  134000;      --          IOBus = R4
                                coloca em IOBUS
022 :  0B0000;      --          LCD write
                                escreve no LCD o que está em IOBUS
023 :  100F05;      --          call DELAY
024 :  0D0000;      --          NOP
025 :  0D0000;      --          NOP
026 :  040405;      --          load R4,RAM[5]
                                guarda variável da ram3 em no reg R4 "M"
027 :  134000;      --          IOBus = R4
                                coloca em IOBUS
028 :  0B0000;      --          LCD write
                                escreve no LCD o que está em IOBUS
029 :  100F05;      --          call DELAY
02A :  0D0000;      --          NOP
02B :  0D0000;      --          NOP
```

```
02C : 040406;     --    load R4,RAM[6]
                        guarda variável da ram3 em no reg R4 "A"
02D : 134000;     --    IOBus = R4
                        coloca em IOBUS
02E : 0B0000;     --    LCD write
                        escreve no LCD o que está em IOBUS
02F : 100F05;     --    call DELAY
030 : 0D0000;     --    NOP
031 : 0D0000;     --    NOP
032 : 04040A;     --    load R4,RAM[10]
                        guarda variável da ram3 em no reg R4 ":"
033 : 134000;     --    IOBus = R4
                        coloca em IOBUS
034 : 0B0000;     --    LCD write
                        escreve no LCD o que está em IOBUS
035 : 100F05;     --    call DELAY
036 : 0D0000;     --    NOP
037 : 0D0000;     --    NOP

038 : 040407;     --    load R4,RAM[7]
                        carrega30ASCII carrega o valor 30 em ascii
                        para ser usado na conversão pega primeiro
                        valor
039 : 11003F;     --    Branch_KB Kboard, KB_RLS
                        aguarda o pressionamento de uma tecla no
                        teclado
03A : 0D0000;     --    NOP
03B : 0D0000;     --    NOP
03C : 080039;     --    jump Branch
03D : 0D0000;     --    NOP
03E : 0D0000;     --    NOP
03F : 11003F;     --    Branch_KB Kboard, KB_RLS
                        aguarda a liberação da tecla pressionada no
                        teclado
```

```
040 :  0D0000;      --          NOP
041 :  0D0000;      --          NOP
                                escreve no IOBUS o que leu do teclado
042 :  150000;      --          IOBusWrite_KBoard
                                salva IOBUS no reg R5
043 :  120500;      --          IOBusRead R5 = IOBus
                                escreve o valor no LCD
044 :  0B0000;      --          LCD write
                                aguarda tempo de operação do LCD
045 :  100F05;      --          call DELAY
046 :  0D0000;      --          NOP
047 :  0D0000;      --          NOP
                                coloca o sinal de + da RAM11 para reg R10
048 :  040A0B;      --          load R10,RAM[11] (R10 = `+`)
                                põe valor de reg r10 no IOBUS
049 :  13A000;      --          IOBus = R10
                                escreve no LCD
04A :  0B0000;      --          LCD write
                                aguarda tempo de operação do LCD
04B :  100F05;      --          call DELAY
04C :  0D0000;      --          NOP
04D :  0D0000;      --          NOP
                                tira trinta do valor lido do teclado e
                                armazenado em
                                r5 fazendo a conversao de ASCII para
                                decimal
04E :  015450;      --          R5 = R5 - R4 (R5=KBdata-0x30)
                                aguarda o pressionamento de uma tecla no
                                teclado
04F :  110055;      --          Branch_KB Kboard, KB_RLS
050 :  0D0000;      --          NOP
051 :  0D0000;      --          NOP
052 :  08004F;      --          jump Branch
053 :  0D0000;      --          NOP
```

```
054 :  0D0000;      --          NOP
                                aguarda a liberação da tecla pressionada no
                            teclado
055 :  110055;      --          Branch_KB Kboard, KB_RLS
056 :  0D0000;      --          NOP
057 :  0D0000;      --          NOP
                                escreve no IOBUS o que leu do teclado
058 :  150000;      --          IOBusWrite_KBoard
                                salva IOBUS no reg R6
059 :  120600;      --          R6=IOBus
                                escreve o valor no LCD
05A :  0B0000;      --          LCD write
                                aguarda tempo de operação do LCD
05B :  100F05;      --          call DELAY
05C :  0D0000;      --          NOP
05D :  0D0000;      --          NOP
                                coloca o sinal de = da RAM12 para reg R10
05E :  040A0C;      --          load R10,RAM[12] (R10=`=`)
                                põe valor de reg r10 no IOBUS
05F :  13A000;      --          IOBus = R10
                                escreve no LCD
060 :  0B0000;      --          LCD write
                                aguarda tempo de operação do LCD
061 :  100F05;      --          call DELAY
062 :  0D0000;      --          NOP
063 :  0D0000;      --          NOP

                                tira trinta do valor lido do teclado e
                                armazenado em r5 fazendo a conversao de
                                ASCII para decimal
064 :  016460;      --          R6 = R6 - R4 (R6=KBdata-0x30)
                                faz a soma dos valores digitados
065 :  005670;      --          R7 = R5 + R6
                                põe resultado no IOBUS
```

```
066 : 137000;   --   IOBus = R7
                     mostra valor de IOBUS nos leds
067 : 160000;   --   Leds = IOBus
                     carrega 0xA (10 decimal) no R8
068 : 040808;   --   load R8,RAM[8] (R8=0x0A)
                     compara o resultado da soma para saber se
é maior
                     que 10
069 : 077871;   --   Branch_Maior R7>R8?MAIOR:MENOR
                     se for maior...
                --   MAIOR:
                     carrega o numero 1 em R9
06A : 040909;   --   load R9,RAM[9]
                     põe no barramento IOBUS
06B : 139000;   --   IOBus = R9
                     ecreve 1 no LCD
06C : 0B0000;   --   LCD write
                     atraso do LCD
06D : 100F05;   --   call DELAY
06E : 0D0000;   --   NOP
06F : 0D0000;   --   NOP
                     tira 10 do resultado da soma
070 : 017870;   --   R7 = R7 - R8 (R7=R7-0x0A)
                     senão era menor...
                --   MENOR:
                     soma 30 fazendo a conversao Decimal para
ASCII
071 : 007470;   --   R7 = R7 + R4 (R7=R7+0x30)
                     põe no R7
072 : 137000;   --   IOBus = R7
                     escreve
073 : 0B0000;   --   LCD write
                     espera
074 : 100F05;   --   call DELAY
```

```
075 :  0D0000;      --          NOP
076 :  0D0000;      --          NOP
                                fica pra sempre preso no loop infinito
                                até que ocorra o proximo reset
077 :  080077;      --          while 1
078 :  0D0000;      --          NOP
079 :  0D0000;      --          NOP

[07A..0ff] :  0D0000;

END;
```

Ao inicializar o programa aparecerá a palavra “SOMA:” no display e os *LED’s* referentes as chaves acionadas estarão ligados. A figura 21 mostra o processador já com o programa inicializado.

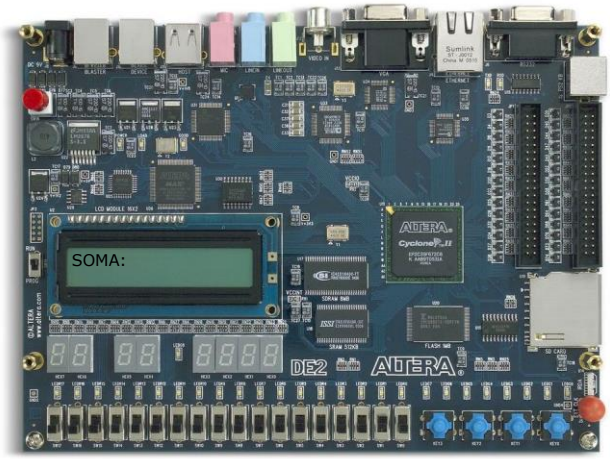


Figura 21 – programa inicializado (ALTERA, 2009 – Adaptado)

Com o programa inicializado basta selecionar no teclado um valor entre 0(zero) e 9(nove) para termos o primeiro número para a realização da soma. A figura 22 mostra a configuração do Kit após selecionado o número 2(dois).

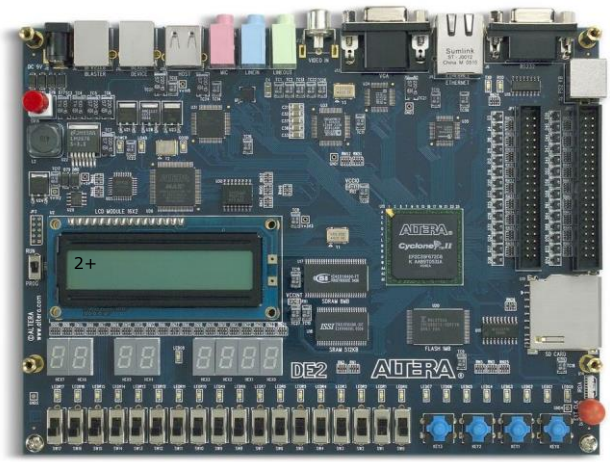


Figura 22 – programa após selecionado o primeiro número para a realização da adição (ALTERA, 2009 – Adaptado)

Após selecionado o primeiro número da adição a ser realizada, basta ao usuário selecionar com o teclado um segundo número entre 0(zero) e 9(nove) para que o processador mostre o resultado final da operação, caso o usuário queira realizar uma nova operação basta apertar o botão reset. A figura 23 mostra a operação concluída.

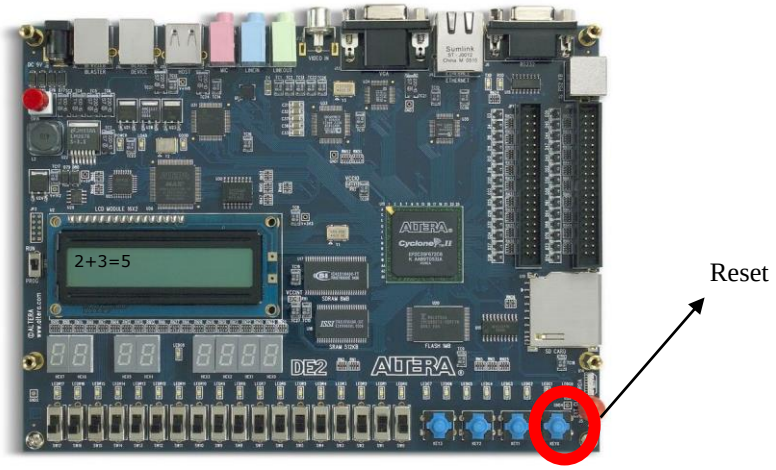


Figura 23 – operação de adicção concluída (ALTERA, 2009 – Adaptado)

Para a simulação da porta paralela optou-se por usar as chaves e *LED's* presentes no kit de desenvolvimento DE2 da Altera caso seja necessário mostrar outro valores na porta paralela basta acionar as chaves desejadas e apertar o

botão reset. A figura 24 mostra as chaves e *LED's* acionados para a simulação do uso de porta paralela.

Figura 24 – simulação do uso da porta paralela

CAPÍTULO 6 - CONCLUSÃO

O sistema mostrou-se bastante eficiente, com bom desempenho, de fácil utilização e muito intuitivo. Porém, devido à arquitetura utilizada, a programação fica restrita a utilização de apenas 256 linhas para a criação de um programa em Assembly e para o uso de 31 OpCode's para o desenvolvimento do mesmo, além de no teclado PS2, devido ao *driver* utilizado, não funcionar o teclado numérico.

Após transferido o programa do Quartus II para o Kit de Desenvolvimento DE2 da Altera, o programa torna-se muito intuitivo e pode ser utilizado por qualquer pessoa que saiba realizar uma operação matemática simples de adição.

Como melhorias futuras, poderia citar a expansão das memórias RAM e ROM e o aumento dos bits de entrada na Unidade de Controle, para que se possa elaborar um programa mais complexo, com mais linhas de código e mais OpCode's. Também pode-se alterar a estrutura do processador para que ele seja capaz de utilizar um mouse e uma tela de computador convencional, tendo em vista que no kit utilizado existem entradas USB e VGA que não são utilizadas no projeto atual. Além do uso de *pileline* para otimizar o processamento de dados do RISC.

CAPÍTULO 7 - REFERÊNCIAS BIBLIOGRÁFICAS

ALTERA CORPORATION, "**Altera: *The Programmable Solutions Company***". Disponível em: http://www.altera.com. Acesso em: junho de 2009.

ALECRIM, Emerson - Publicado em 08/07/2009 disponível em http://www.infowester.com/memoria.php, Acesso em Março de 2009.

SICA, Carlos **ESTUDO SOBRE A ARQUITETURA RISC**, 1999, Centro de Tecnologia, Departamento de informática) Universidade Estadual de Maringá UEM, disponível em www.din.uem.br/sica/material/4ano/risc.doc, acessado em julho de 2009

GIACOMINI, Renato Camargo, **Apostila Básica VHDL**, 2000, Professor Doutor da Universidade São Judas Tadeu, disponível em http://ipsulinha.net78.net/ acessado em Julho de 2009.

MENDONÇA, Alexandre; ZELENOVISKY, Ricardo **PROJETOS COM FPGA: FAMÍLIAS MODERNAS**, 2008, disponível em http://www.mzeditora.com.br/artigos/fpga_fam.htm acessado em setembro de 2009.

JÚNIOR, Valfredo Pilla; **Material didático da disciplina de MICROPROCESSADORES**, Curso de Engenharia da Computação da Universidade Positivo, 2006 disponível em www.up.edu.br dentro de área restrita a alunos do curso, acessado em outubro de 2009.

MATOS, Nádia Pádua - SESA; RASKIN, Sara Fichman - GPT; **Aspectos da Arquitetura de Processadores RISC**, disponível em http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1259, acessado em Março de 2009.

NAVALBI, Z. **Embedded Core Design with FPGAs**. McGraw-Hill. New York, 2007.

ALTERA CORPARATION. **DE2 Development and Education Board User Manual**. Disponível em:

http://www.altera.com/education/univ/materials/boards/DE2_UserManual.pdf. Acessado em Maio de 2009.

BRISOLARA, Lisane Brisolara de; PILLA, Maurício Lima, **Descrição VHDL do MIPS *Pipeline*,** 2000, Universidade Federal do Rio Grande do Sul, disponível em http://www.inf.ufrgs.br/~lisane/relatMIPS.pdf acessado em Agosto de 2009.

JUNQUEIRA, Roberto S. **PDA BASEADO NO MICROPROCESSADOR NOIS II COM SISTEMA OPERACIONAL UCLINUX EMBARCADO EM FPGA**, 2007, Universidade Positivo, Disponível em www.engcomp.up.edu.br, Acessado em Maio de 2009

ANEXO A <ARTIGO>

Processador RISC Didático.

Raphael D'Oliveira Fogaça

Universidade Positivo
Prof. Pedro Viritato Parigot de Souza,
5300, Campo Comprido, Curitiba- PR

Resumo: *Este documento consiste no projeto de desenvolvimento de um processador RISC didático aonde se possibilita uma melhor visualização e testes no processador. O projeto consiste num processador embarcado em um kit de FPGA onde o aluno tem uma interação com o processador, tornando visual a funcionamento do mesmo, facilitando sua compreensão.*

Palavras-Chave: Processador, RISC, FPGA.

1 – INTRODUÇÃO

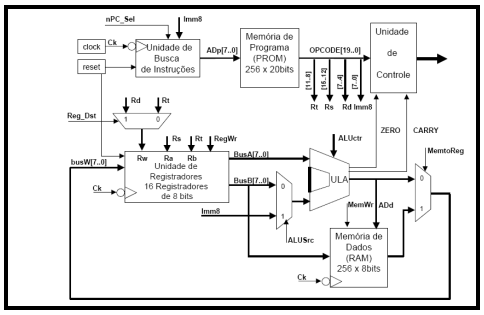
O presente trabalho mostra um projeto realizado durante o ano letivo atual na Unidade Positivo, cujo objetivo principal é desenvolver uma ferramenta educacional e de aprendizado acadêmico. Nesse projeto descreve-se os métodos utilizados para a elaboração de um processador RISC (*reduced instruction set computer*) didático utilizando um kit de desenvolvimento FPGA (*field-programmable gate array*) DE2 da Altera.

Diagrama do Projeto

2 - METODOLOGIAS E METODOS UTILIZADOS

No escopo do projeto está a elaboração de dispositivos de entrada e saída (teclado e porta paralela) bem como a utilização de um display LCD (já presente no kit DE2) para a visualização dos resultados. Para a elaboração do projeto foi utilizado como base um processador RISC desenvolvido durante a disciplina de Microprocessadores, com o professor

Valfredo Pilla Jr. no decorrer do curso de Engenharia da Computação. Foram acrescentados ao processador base alguns blocos para a utilização do teclado, display LCD e simulação da porta paralela, bem como alterada a arquitetura do mesmo para que fosse possível criar programas mais complexos do que os utilizados originalmente.



Processador RISC base

Para o display LCD e teclado foram criados *drivers* em VHDLcom as funções necessárias para a utilização dos mesmos, já para a porta paralela foi realizado uma simulação da utilização da mesma com chaves e *LED's* presentes no kit.

A Unidade de Controle original foi alterada para ampliar a sua capacidade de gerenciamento de OpCode's no programa Assembly a ser desenvolvido, e as memórias RAM e ROM também tiveram suas capacidades ampliadas a fim de tornar possível um programa com mais linhas de código. Com o projeto completo é possível um estudante do curso, ou alguém que tenha noção de programação Assembly e Quartus II, Criar um programa, com os OpCode's existentes e efetuar testes no processador.

3 – CONCLUSÃO

O sistema mostrou-se bastante eficiente, com bom desempenho, de fácil utilização e muito intuitivo. Porém, devido à arquitetura utilizada, a programação fica restrita a utilização de apenas 256 linhas para a criação de um programa em Assembly e para o uso de 31 OpCode's para o desenvolvimento do mesmo, além de no teclado PS2, devido ao *driver* utilizado, não funcionar o teclado numérico.

Após transferido o programa do Quartus II para o Kit de Desenvolvimento DE2 da Altera, o programa torna-se muito intuitivo e pode ser utilizado por qualquer pessoa que saiba realizar uma operação matemática simples de adição.

Como melhorias futuras, poderia citar a expansão das memórias RAM e ROM e o aumento dos bits de entrada na Unidade de Controle, para que se possa elaborar um programa mais complexo, com mais linhas de código e mais OpCode's. Também pode-se alterar a estrutura do processador para

que ele seja capaz de utilizar um mouse e uma tela de computador convencional, tendo em vista que no kit utilizado existem entradas USB e VGA que não são utilizadas no projeto atual. Além do uso de *pileline* para otimizar o processamento de dados do RISC.

APÊNDICE <MANUAL>

Para se utilizar o sistema é preciso ter um computador com o software Quartus II da altera instalado, o Kit de Desenvolvimento DE2 da Altera e um teclado PS2 convencional. Além dos arquivos do projeto.

O sistema tem as seguintes limitações de uso:

- Programa Assembly com no máximo 256 linhas de código;
- Código Assembly utilizando OpCode’s disponíveis no processador, conforme Tabela 1, ou criar novos OpCode’s dentro da Unidade de Controle do processador;
- Teclado PS2 convencional sem funcionamento da parte do teclado numérico.

Tabela 1 – OpCode’s disponíveis no processador.

OpCode	Função	Descrição
00000	ADD	Operação de “Adição”
00001	SUB	Operação de “Subtração”
00010	OR	Operação “Ou”
00011	AND	Operação “E”
00100	LOAD	Carrega algum dado
00101	STORE	Armazena algum dado
00110	BRANCH_EQ (IGUAL)	Efetua uma comparação buscando uma igualdade com zero
00111	BRANCH_MAIOR	Efetua uma comparação para ver se o valor é maior do que 10
01000	JUMP	Pula para o um endereço de memória
01001	LCD RESET	Reinicia o LCD
01010	LCD INIT	Inicia o LCD
01011	LCD WRITE	Escreve no LCD
01110	MOVE	Move algum dado
01111	RET	Retorno
10000	CALL	Chama alguma função
10001	BRANCH_KBOARD	Faz a comparação dos sinais das tecals do teclado
10010	IOBusRead	Faz a leitura do barramento de IO
10011	IOBusWriteReg	Escreve no barramento de IO
10100	IOBusWriteSwitch	Escreve no barramento das Chaves
10101	IOBusWriteKBoard	Escreve no barramento do Teclado
10110	LedWrite	Faz a leitura dos LED’s

Programa Teste

Primeiramente deve-se energizar o Kit de Desenvolvimento DE2 da Altera com a fonte que o acompanha, conectar o cabo USB no Computador com o Quartus II instalado e o teclado na entrada PS2 presente no Kit. Como mostra o diagrama da Figura 1.

Figura 1 – Diagrama de montagem do projeto

Depois de montar o projeto deve-se inicializar o Quartus II e abrir o projeto LCD_Driver_Tester, como mostra a Figura 2.

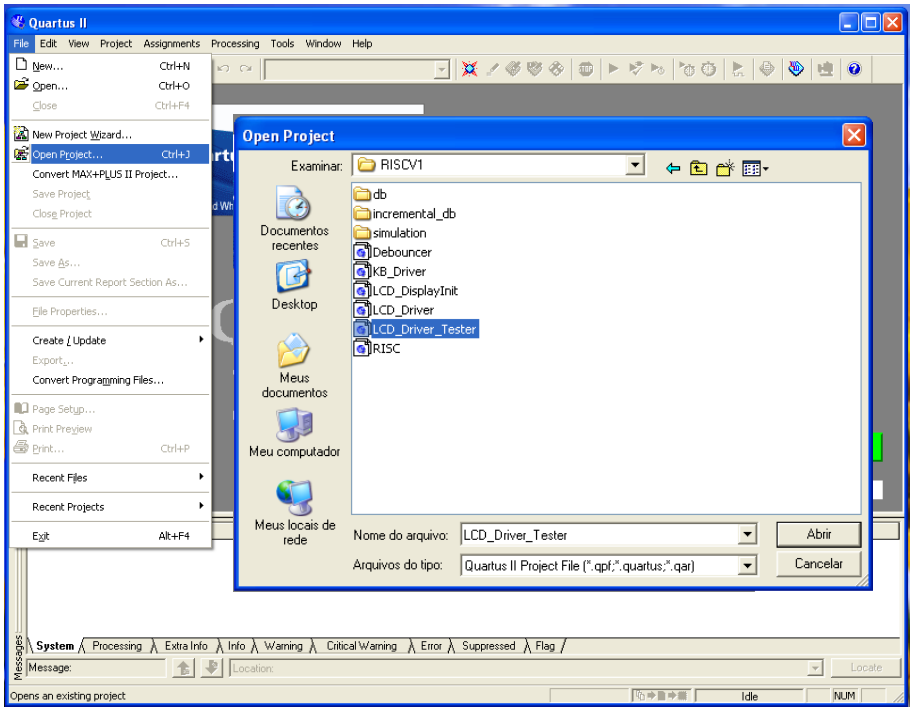


Figura 2 – Tela de abertura do projeto

Com o projeto aberto o próximo passo e compilá-lo, para isso basta clicar no botão “Start Compilation” e aguardar a compilação completa.

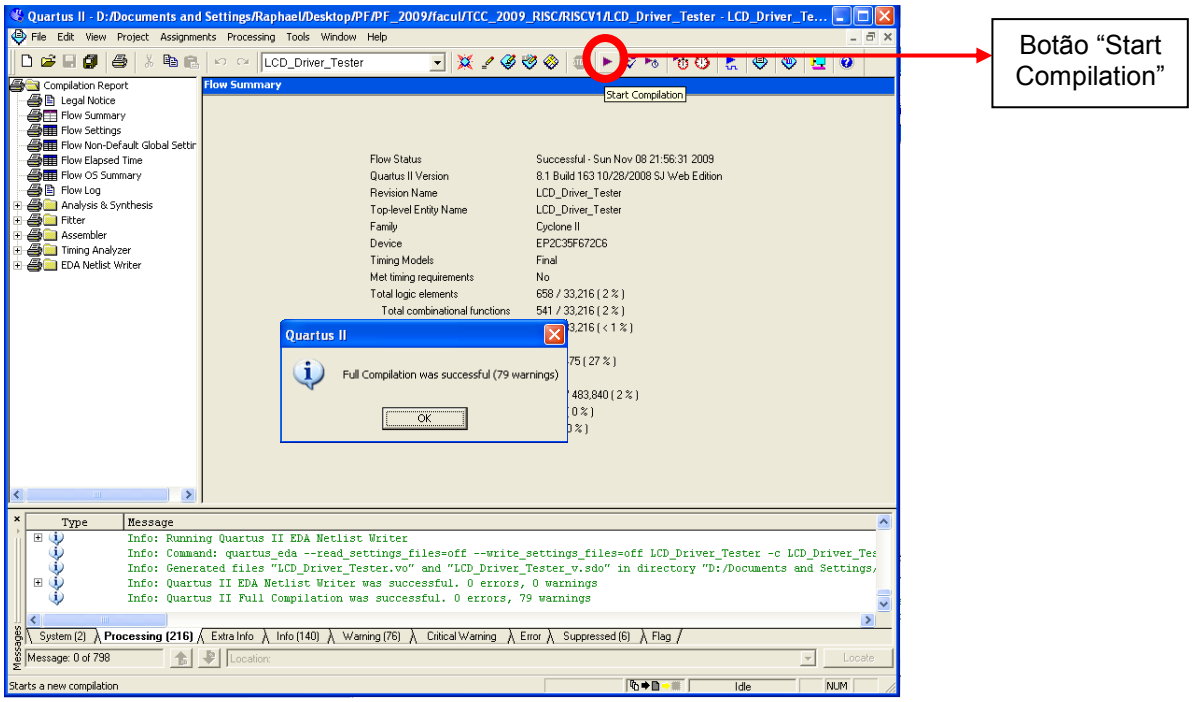


Figura 3 – Tela da compilação do programa

Após o projeto compilado deve-se gravar o programa na FPGA do Kit de desenvolvimento DE2 da Altera. Para isso basta clicar no botão “Programmer” na tela seguinte clique em “Autodetect” para que o Quartus detecte o Kit DE2 e por fim clique em Start e aguardar o projeto ser gravado na FPGA.

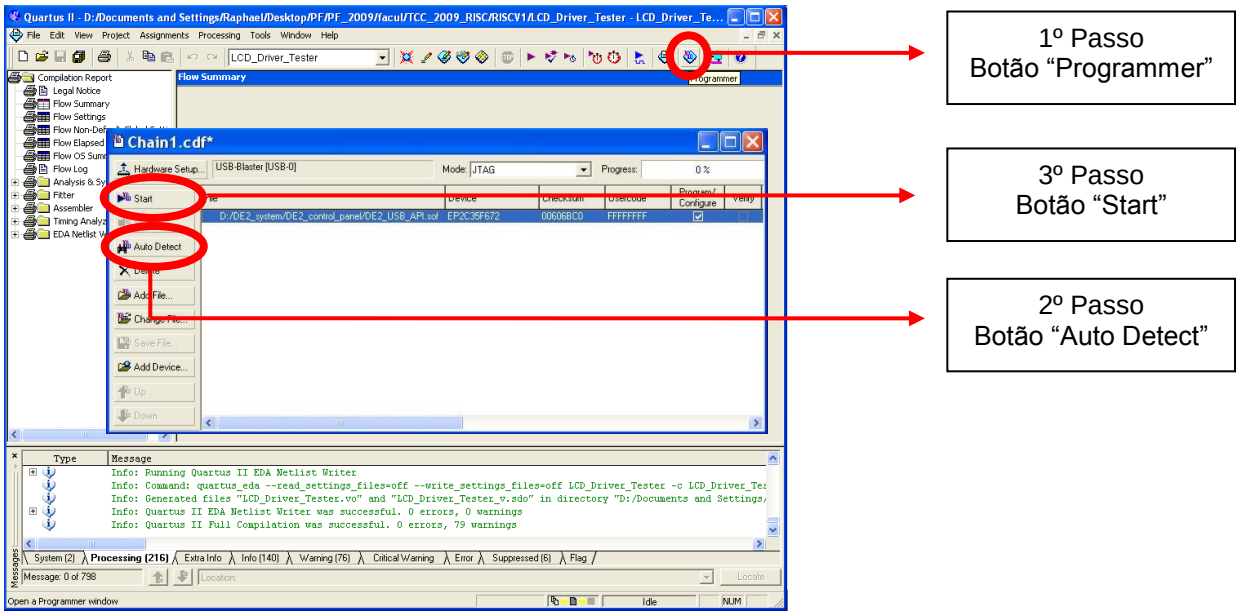


Figura 4 – Tela de gravação do projeto na FPGA

Já com o projeto gravado na FPGA do Kit o programa ira indicar automaticamente e aparecerá a palavra “SOMA:” no display e os *LED’s* referentes as

chaves acionadas estarão ligados. A figura 5 mostra o processador já com o programa inicializado.

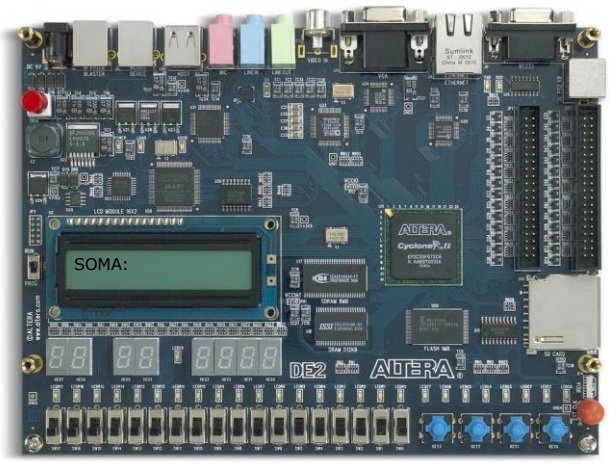


Figura 5 – programa inicializado

Com o programa inicializado basta selecionar no teclado um valor entre 0 (zero) e 9 (nove) para termos o primeiro número para a realização da soma. A figura 6 mostra a configuração do Kit após selecionado o número 2 (dois).

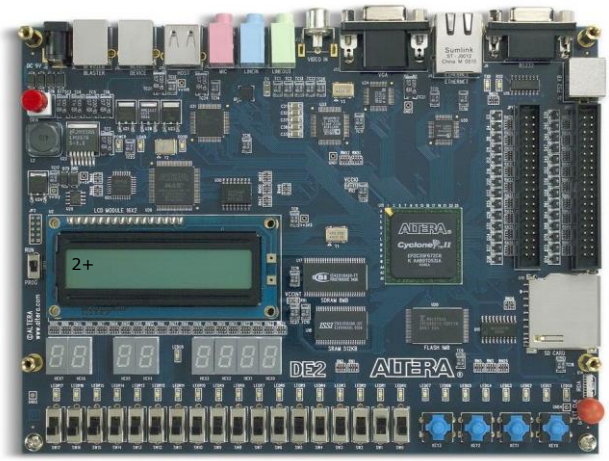


Figura 6 – programa após selecionado o primeiro número para a realização da adição

Após selecionado o primeiro número da adição a ser realizada basta ao usuário selecionar com o teclado um segundo número entre 0(zero) e 9(nove) para que o processador mostre o resultado final da operação, caso o usuário queira realizar uma nova operação basta apertar o botão reset. A figura 7 mostra a operação concluída.

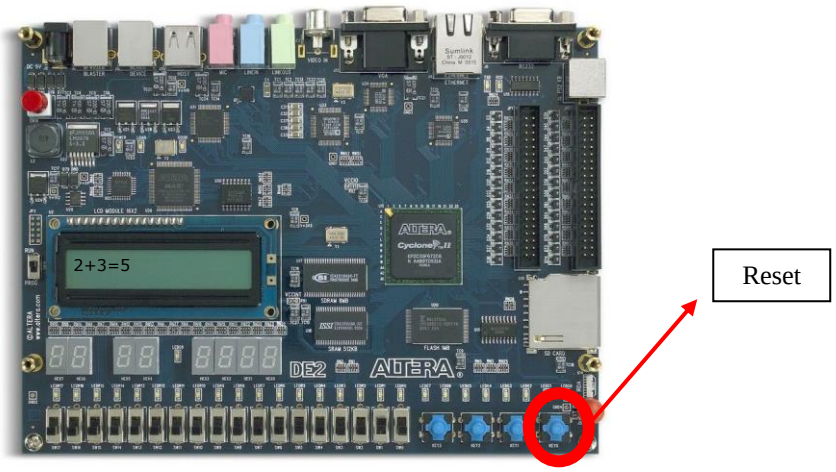


Figura 7 – operação de adição concluída

Para a simulação da porta paralela optou-se por usar as chaves e *LED's* presentes no kit de desenvolvimento DE2 da Altera caso seja necessário mostrar outro valores na porta paralela basta acionar as chaves desejadas e apertar o botão reset. A figura 8 mostra as chaves e *LED's* acionados para a simulação do uso de porta paralela.

Figura 8 – simulação do uso da porta paralela

Criar OpCode’s na Unidade de Controle

A Unidade de Controle apresenta um barramento de entrada de 5 bits, de 00000 a 11111 (em binário), assim é possível criar e executar até 31 OpCode’s. Para isso, basta dar um duplo-clique na Unidade de Controle, Figura 9, presente no projeto e programar a função desejada conforme a estrutura do processador/Kit.

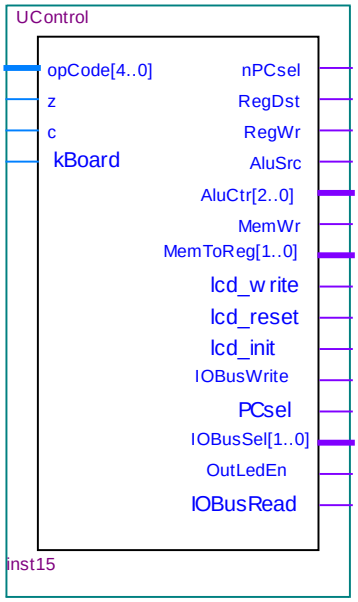


Figura 9 – Unidade de Controle

Em seguida é mostrado um exemplo de programação da função “ADD” no processador.

```
WHEN B"00000" => -- ADD
                 nPCsel = GND;
                 PCsel = GND;
                 RegDst = VCC;
                 RegWr = VCC;
                 MemWr = GND;
                 MemToReg = B"00";
                 lcd_reset = GND;
                 lcd_write = GND;
                 lcd_init = GND;
                 AluSrc = GND;
                 AluCtr[] = B"101";
                 IOBusWrite = GND;
                 IOBusRead = GND;
                 IOBusSel = B"00";
                 OutLedEn = GND;
```

Criar Programa Assembly

Para criar um programa Assembly novo, basta alterar o arquivo ROM.mif , com o bloco de notas, presente nos arquivos do processador com o novo código Assembly desenvolvido.

Com o programa alterado, torna-se necessário carregá-lo novamente na memória ROM, como mostra o passo a passo abaixo:

1. Duplo clique na memória ROM presente no Projeto, para abrir a tela de configuração da mesma.

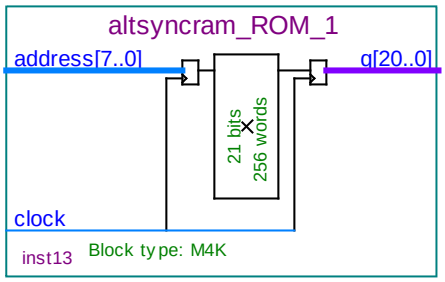


Figura 10 – Memória ROM

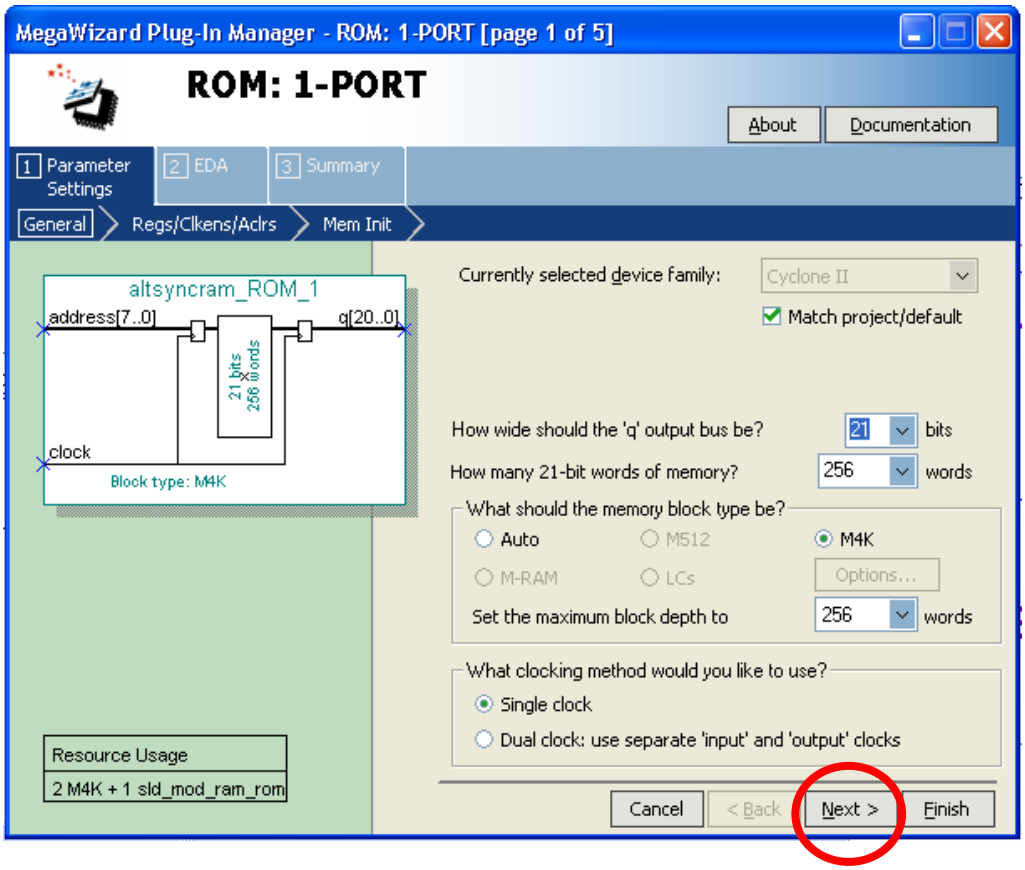


Figura 11 – Tela de configuração da memória ROM

2. Com a tela de configuração da memória ROM aberta clique em “Next” até aparecer a opção de procurar o arquivo ROM.mif, conforme Figura 12, localize o arquivo.mif referente ao novo código Assembly e finalize a operação ao clicar no botão “Finish”.

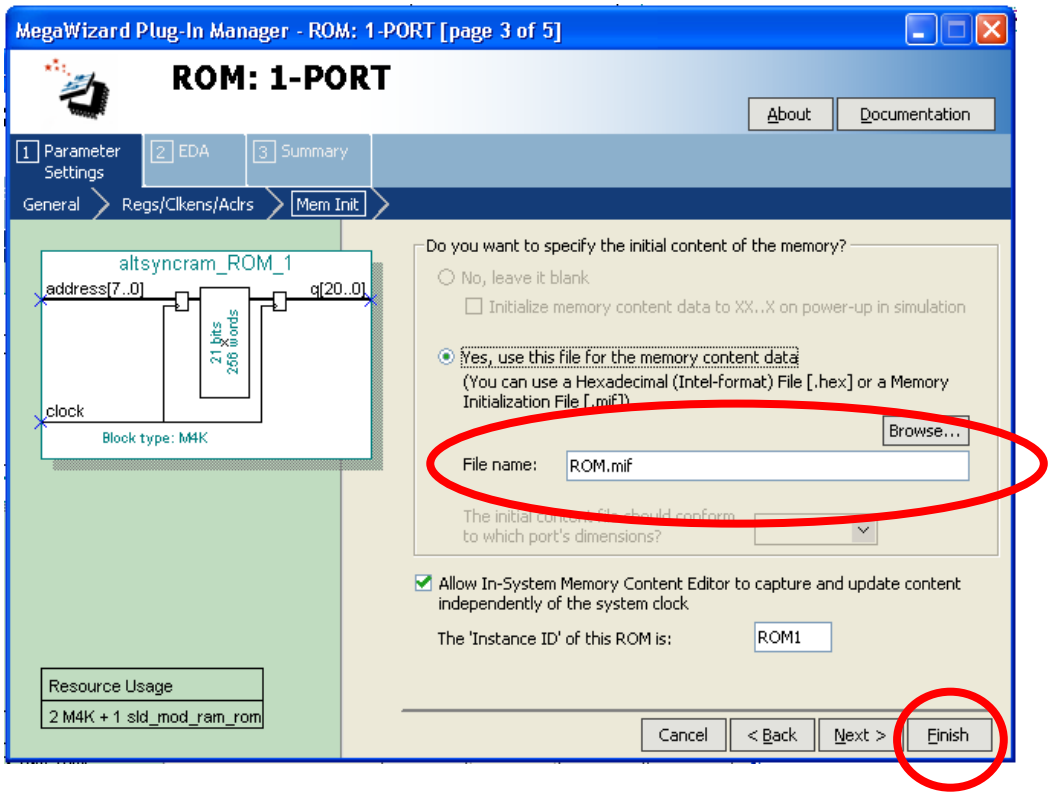


Figura 12 – Tela de busca de arquivos da memória ROM

Com essas alterações efetuadas é possível testar o seu programa no processador RISC Didático

É importante lembrar que esse código deve atender as limitações de 256 linhas de código e aos OpCode’s presentes no projeto ou criados previamente.