UNIVERSIDADE POSITIVO
NÚCLEO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DA COMPUTAÇÃO

AMBIENTE OPERACIONAL PARA UMA ARQUITETURA PARALELA RECONFIGURÁVEL

Curitiba
2008

UNIVERSIDADE POSITIVO
NÚCLEO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DA COMPUTAÇÃO

AMBIENTE OPERACIONAL PARA UMA ARQUITETURA PARALELA RECONFIGURÁVEL

Alexandro Cardoso

Monografia apresentada à disciplina de Trabalho de Conclusão de Curso, como requisito parcial à conclusão do Curso de Engenharia da Computação. Orientador: Prof. Edson Pedro Ferlin.

Curitiba
2008

TERMO DE APROVAÇÃO

Alexandro Cardoso

AMBIENTE OPERACIONAL PARA UMA ARQUITETURA PARALELA RECONFIGURÁVEL

Monografia aprovada como requisito parcial à conclusão do Curso de Engenharia da Computação da Universidade Positivo, pela seguinte banca examinadora:

Prof. Edson Pedro Ferlin (Orientador)
Prof. Amarildo Geraldo Reichel
Prof. Alessandro Zimmer

Curitiba
2008

AGRADECIMENTOS

Gostaria de agradecer os meus pais, e meu irmão, que sempre estiveram me apoiando, confiando e ajudando a superar todas as dificuldades durante esta jornada. Também aos meus amigos, professores, colegas de trabalho, todos que me apoiaram e fizeram parte desse período acadêmico da minha vida.

RESUMO

Com o aumento na demanda por capacidade de processamento em diversas áreas da computação, é cada vez mais freqüente as pesquisas na área de processamento. O processamento paralelo é geralmente aplicado em arquiteturas que utilizam vários processadores para obter ganho de desempenho em relação a computadores que operam por meio de um único processador. Para controlar todo o processamento dessas informações, há a necessidade de interfaces. O objetivo desse projeto é desenvolvimento de uma interface para análise de expressões aritméticas. A entrada de dados faz-se por meio de um computador, o qual possui um *software* para gerar as operações a serem processadas, desenvolvidos em C e C++, projeto de um *software* de comunicação para enviar as operações do computador para uma máquina paralela, que utiliza uma interface baseada no processador embarcado NIOS II da Altera desenvolvida em 2007, e um projeto de uma Arquitetura Paralela, desenvolvida em 2006. O sistema foi desenvolvido na ferramenta Quartus II da Altera, e que serão gravados em um kit para lógica programável, utilizando-se FPGA (*Field Programmable Gate Array*), que é um dos tipos de dispositivos lógico programáveis, os quais permitem que o projeto seja implementado sem a necessidade de construção física dos componentes, aumentando a produtividade e a flexibilidade no desenvolvimento do projeto. O projeto visa automatizar a parte de criação de *templates* e validação de expressões matemáticas, que serão processadas pela arquitetura paralela. Após a finalização das operações, os resultados serão enviados de volta para o computador.

Palavras-chave:
Ambiente Operacional, Análise de Expressões, Processamento Paralelo.

OPERATIONAL ENVIRONMENT FOR A PARALLEL RECONFIGURABLE ARCHITECTURE

ABSTRACT

With increasing demand for processing capacity in several areas of computing, is even more frequent researchs in processing area. The paralel processing is aplied in architetures that use multiple processors to achieve performance gains for computers operating through a single processor. To control all the processing of the information, there is a need for interfaces. The objective of this project is to develop an interface to analyze arithmetic expressions. The input of data is made by a computer, that has a software to generate the transactions to be processed. This software was developed in C and C++ programming languages. The comunication software, which was developed in 2007, sends the information from the computer to the paralel machine, which uses an architeture based on Altera's embedded processor NIOS II. The Paralel Architeture, which was developed in 2006, was compiled in the Altera's Quartus II program. These projects which will be recorded in a kit for programmable logic. Using the FPGA (Field Programmable Gate Array), which is one type of programmable logic device, permits the project be implemented without the need for physical construction of components, increasing productivity and flexibility. The project aims to automate the creation of templates and validation of mathematical expressions, which will be processed by paralel architeture. After the completion of operations, the results will be sent back to computer.

Keywords:

Operational Environment, Analysis of Expressions, Paralel Processing

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ASIC – Application-Specific Integrated Circuits.

CPLD – Complex Programmable Logic Device.

EEPROM – Electrically Erasable Programmable Read-Only Memory.

EP – Elemento Processador.

EPROM – Erasable Programmable Read-Only Memory.

FPGA – Field Programmable Gate Arrays.

IDE – Integrated Development Environment.

JTAG – Joint Test Action Group.

PLD – Programmable Logic Device.

PROM – Programmable Read-Only Memory.

RAM – Read Only Memory.

VHDL – VHSIC Hardware Description Language.

USB – Universal Serial Bus.

INTRODUÇÃO

Devido a grande evolução da tecnologia, o interesse por problemas cada vez mais complexos tem aumentado a necessidade de computadores cada vez mais potentes para resolvê-los. Porém, limitações físicas e econômicas tem restringido o aumento dos computadores seqüenciais, ou seja, que executam instruções uma após a outra pela CPU. Entretanto, problemas computacionais podem ser divididos em partes, e também solucionados ao mesmo tempo, ou processados em paralelo, suprindo assim uma grande demanda computacional por meio do uso simultâneo de recursos computacionais como processadores, para a solução de um problema. Esse conceito é definido como Computação Paralela que é caracterizada pelo uso de várias unidades de processamento ou processadores para executar uma computação de forma mais rápida.

Sistemas baseados em computação reconfigurável (sistemas de *hardware* reconfigurável) apresentam características adequadas para uso nesta classe de projetos, dentre outras vantagens e características como o baixo consumo, alta velocidade de operação, capacidade de intregração, flexibilidade, facilidade de programação, etc. Este projeto tem a finalidade de resolver e validar expressões e aritméticas por meio de um software que tem a função de transformar essas expressões em pacotes chamados de *templates*, que serão enviadas a uma máquina paralela baseada no modelo de fluxo de dados, ao qual será encarregada de resolver a expressão de forma fracionada, e apresentar os resultados no monitor.

CAPITULO 1
1. INDRODUÇÃO AO PROJETO

1.1. Motivação

Nos anos de 2006 e 2007 foram desenvolvidos dois projetos, sendo o primeiro uma arquitetura paralela reconfigurável básica, e o segundo, uma interface de controle reconfigurável para essa máquina paralela. Contudo, existe a necessidade de um ambiente operacional que automatize a parte de análise de expressões matemáticas, que serão enviadas a arquiteturas paralela, sendo esse processamento baseado no grafo de fluxo de dados. Esse ambiente possibilitará a geração de *templates* que serão processados por essa máquina, e os resultados apresentados no computador.

1.2. Contextualização

Análises e cálculos de expressões matemáticas utilizando compiladores exigem um bom planejamento do código, o que facilita o desenvolvimento de implementações de algoritimos desse gênero. O principal processo de um compilador para a análise de expressões aritméticas acaba sendo a Análise Sintática (também conhecido pelo termo em inglês *parking*), processo esse que analisa uma seqüência de entrada (lida de um arquivo do computador ou diretamente do teclado) que avalia uma estrutura gramatical baseado em uma determinada gramática definida no compilador, e é totalmente dependente de uma análise que precede ela, a primeira etapa de um compilador, as Análises Léxicas, que obrem um grupo de totens para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura. As manipulações dessas estruturas são geralmente desenvolvidas utilizando algoritmos de listas duplamente encadeadas, que facilitam o tratamento dessas expressões.

1.3. Principais Funcionalidades

O projeto será composto de um software montador de *templates*, do processador NIOS II desenvolvido no ano de 2007, e da arquitetura paralela desenvolvida no ano de 2006. O *software* é capaz de montar automaticamente os *templates* por meio de expressões digitadas na interface com o usuário, fornecendo informações que serão processadas pela máquina paralela. Entre o *software* e a arquitetura paralela, existe um componente de *hardware* embarcado em FPGA (*Field Programmable Gate Array*), responsável pelo gerenciamento do fluxo de informações vindas do

computador. Essa integração é feita pelo processador NIOS II, que utiliza um barramento de alta velocidade de grande capacidade de processamento. A arquitetura paralela é possui uma unidade de controle que faz a distribuição de processamento de vários EPs (*Elementos Processadores*) responsáveis pelo processamento.

1.4. Tecnologia

A interface do software foi implementada utilizando o software C++ Builder. Para as demais configurações e adaptações foram utilizadas as ferramentas Quartus II e NIOS II da Altera e o kit DE2 da Altera, contendo os dispositivos programáveis.

CAPITULO 2

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo apresenta-se o estudo teórico, conceitos e tecnologias que foram utilizadas no decorrer desse projeto. Os conceitos abordados são: Compiladores, Análise de Expressões, Processamento Paralelo, Modelo de Fluxo de Dados, Computação Reconfigurável, Dispositivos Lógicos Programáveis e Linguagem para Descrição de H*ardware*.

2.1. Teoria de *Software*

2.1.1 Compiladores

Inicialmente descreve-se alguns conceitos referentes a linguagem de programação e compiladores, mostrando aspectos importantes na construção de compiladores, atendendo assim, a necessidade do desenvolvimento de um compilador simples, destacando as funções internas sem a necessidade de se fazer otimização e geração de código.

2.1.2 Funcionamento básico de um compilador

A análise de um compilador é dividida em outras três análises: análise léxica, análise sintática.

2.1.3 Analisador Léxico

A análise léxica é a primeira das três fases que compõe a análise do programa-fonte. Um Analizador léxico, também chamado *parser*, possui várias tarefas, não somente relacionadas à análise léxica. A principal função deste analisador é fragmentar o programa fonte de entrada em trechos elementares completos, bem fragmentados e com identidade própria. Estes componentes básicos são chamados *tokens*. O analisador léxico atua como uma interface entre o programa fonte e o analisador sintático, tendo a função de varrer o programa fonte da esquerda para a direita, agrupando os símbolos de cada item léxico e determinando sua classe. Esta estrutura nos obriga a construir uma máquina abstrata, conhecida como autômato finito.

Algumas funções internas do analisador léxico são descritas a seguir:

- **Extração e classificação de átomos:** Esta é a função mais clara e imediata do analisador léxico, ou seja, mapear o programa fonte em outro texto, formado por átomos representados pelos símbolos que compõem o programa fonte. As classes de átomos mais encontradas são: palavras reservadas, números inteiros sem sinal, números reais, cadeias de caracteres, sinais de pontuação, de operação, caracteres especiais, símbolos compostos e comentários, entre outros.

- **Tratamento de identificadores:** O tratamento é semelhante ao dado para números, visto que os identificadores também são cadeias de tamanho variável. Sua representação é feita através de uma tabela de símbolos que, por simplicidade reserva para cada símbolo um comprimento constante.

- **Identificação de palavras reservadas:** Esta função encarrega o analisador de encontrar um identificador no programa - fonte. Neste caso, é preciso verificar se tal identificador pertence a um conjunto de identificadores especiais com significado pré-definido pela linguagem, que são as palavras-chaves, ou palavras reservadas. Como, do ponto de vista formal, não existe nenhuma distinção entre identificadores e palavras reservadas, o compilador separa os identificadores em duas etapas: reconhece o átomo como identificador e depois verifica se o identificador é ou não palavra reservada. Em geral, a técnica utilizada é percorrer uma tabela, uma lista, ou uma árvore que representa o conjunto de palavras reservadas pela linguagem.

- **Recuperação de erros:** Quanto ocorre caracteres não identificados, ou seqüências que não obedecem a lei de formação da linguagem na leitura do programa fonte pelo analisador, diz - se que existem erros léxicos neste texto. Para que o compilador possa prosseguir na análise, apesar da perda de sincronismo no autômato que implementa o analisador léxico, são usados mecanismos de resincronização denominados *mecanismos de recuperação de erros*. Uma outra opção é separar os átomos não reconhecidos e enviá-los para serem tratados pelo analisador sintático.

- **Controle de listagens:** Existem, entre os comandos de controle do compilador, aqueles em que o programador tem acesso à rotinas que efetuam a geração de listagens. Entre estas rotinas estão as que permitem ao programador ligar e desligar

opções de listagem, mapeamentos de memória, coleta de símbolos em tabelas de referências cruzadas e formatação das saídas impressas.

- **Interação com sistema de arquivos :** Ao analisador léxico compete acessar o arquivo de leitura do programa fonte, via sistema operacional. No caso geral este procedimento é simples, porém pode existir a possibilidade do programador especificar, por meio de comando no compilador, o chaveamento de vários arquivos. O programa fonte é então construído pela justa posição destes arquivos. Neste caso, torna-se necessário construir uma interface mais elaborada com Sistema Operacional.

2.1.4 Analisador Sintático

A principal função do analisador sintático é de promover a análise da seqüência com que os elementos chamados *tokens* do programa fonte se apresentam, com base na estrutura dessa linguagem. O analisador sintático processará a sequência dos *tokens* proviniente do programa fonte, extraídos do analisador léxico. Por meio da análise léxica é obtido um grupo de *tokens*, para que o analisador sintático use um conjunto de regras, e efetue a verificação da ordem de apresentação na sequência, de acordo com a gramática na qual se baseia o compilador.

A seguir serão relacionadas algumas das principais funções do analisador sintático.

- **Identificação de sentenças:** O Analisador sintático pode ser visto como um aceitador de cadeias, em que o conjunto dessas cadeias forma a linguagem que é reconhecida pelo analizador.

- **Detectação de erros de sintaxe:** Nesse caso é detectado uma sentença que não pertence a linguagem, o analisador deve identificar sua ocorrência, acusando erros de sintaxe, ao qual o programador é informado sobre o ponto.

- **Recuperação de Erros:** Na análise sintática de muitos compiladores são incorporados meios de ressincronizão do reconhecedor, sempre que forem encontradas construções não pertencentes à linguagem. Isto é importante para que a análise não pare de ler o programa fonte, e, na continuação, possam ser encontrados todos os erros possíveis de serem detectados pelo compilador.

- **Comando de ativação do analizador léxico:** Em muitas implementações o analisador sintático detecta a necessidade de novos átomos a serem reconhecidos. Desta forma, é comandada a análise léxica do programa fonte em função da sintaxe.

2.1.5 Análise de Expressões

A avaliação automática de expressões aritméticas semelhantes aquelas encontradas na matemática foi uma das principais metas das primeiras linguagens de programação de alto nível, tendo suas principais características herdadas de convenções que se desenvolveram na matemática. Nas linguagens de programação, as expressões matemáticas consistem em operadores, operandos, parênteses e chamadas a função, tendo os operadores definidos entre unários (que possuem apenas um operando), ou binários (que possuem 2 operandos).

A análise de expressões é baseada perante as principais regras encontradas nas expressões aritméticas, como a de precedência de operadores, associatividade de operadores, ordem de avaliação de operandos, etc. As regras de precedência de operadores para avaliação de expressões definem a ordem em que os operadores de diferentes níveis de precedência são avaliados. Já quando uma expressão contém duas ocorrências de operadores com o mesmo nível de precedência, a questão sobre qual deles é avaliado primeiro, corresponde a associatividade de operadores. A ordem de avaliação de operandos é tratada quando há a presença de constantes, entre as variáveis de determinada expressão. Portanto, as expressões consistem em constantes, variáveis, parênteses, chamadas a função, operadores e operandos.

A avaliação de expressões matemáticas ou aritméticas, envolve também alguns conceitos de linguagens formais e compiladores, como análise sintática e semântica, além de conceitos sobre recursividade em linguagens de programação.

2.2. Teoria de *Hardware*

2.1.1 Processamento Paralelo

Embora os computadores tenham apresentado um grande avanço na velocidade de processamento nas últimas décadas, e apesar de existir uma grande potência computacional nos mais diferentes ramos de atividade, para muitos casos ela não é suficiente. A demanda por um poder computacional maior tem aumentado, e as exigências sobre eles cresceram mais rapidamente que a sua capacidade de processamento. A velocidade de operação dos processadores está sempre crescendo, mas não podem continuar assim indefinidamente, pois existem limitações físicas que em algum momento impedirão a evolução dos processadores como é feito atualmente. Em razão disso os projetistas de computadores estão se voltando para os computadores paralelos em busca de arquiteturas que utilizem vários processadores para obter ganho de desempenho em relação a computadores que operam por meio de um único processador. Em razão disso, é cada vez mais intenso o desenvolvimento de computadores paralelos. Esse projeto está relacionado ao desenvolvimento de uma interface entre o computador o qual estão as informações a serem processadas e a máquina paralela realizará o processamento, apresentando todo o resultado na tela do computador.

2.1.2 Modelo de Fluxo de Dados

No final da década de 70 surgiam as arquiteturas Fluxo de Dados (*Data Flow*), com o objetivo de explorar o paralelismo entre as instruções de um programa. Os sistemas Fluxo de Dados não possuem variáveis, pois os valores são representados por pacotes que são transmitidos entre os processadores. Essas máquinas possuem uma única memória para os dados e para as instruções e não possuem um apontador de instruções como no modelo Von Neumann. Cada processador executa alguma operação com sua entrada, e produz uma saída contendo o resultado, e nesse caso, cada operação depende exclusivamente de suas entradas. Cada processador inicia a execução das tarefas assim que seus dados de entrada forem disponibilizados. Dessa forma, não há variáveis globais ou qualquer informação externa. Para cada processador disponível é associado um *template* (pacote) que contém informações a respeito da operação a ser realizada, os dados de entrada, operação, sinais de controle e uma lista dos destinos de saída. Para cada ciclo de execução, é obtido e despachado todos os *templates* prontos, cujo resultados são armazenados nos destinos apropriados, e assim, disponibilizando *templates* que utilizam como entrada o resultado do processamento de outro *template*. Sob estas condições, se o processamento for iniciado com um único processador e posteriormente for adicionando mais processadores, o desempenho do computador crescerá até que todo paralelismo implícito tenha sido explorado, aproveitando a escalabilidade do sistema. O fluxo de controle sobre as operações é dado pela disponibilidade dos dados de entrada para a execução de uma instrução *dataflow*, num processo chamado *data driven*

(dirigido pelos dados). Um programa *dataflow* é organizado como um grafo, os nós representam as instruções e os arcos representam o fluxo de dados entre os nós. No modelo *data driven* no instante que um nó do grafo referente a uma instrução detectar que todos os seus arcos de entrada estão habilitados, ele executa a instrução e fornece um resultado de saída, que pode vir a habilitar outros nós do programa *dataflow*. Dessa forma, o paralelismo entre as instruções acontece de forma natural, na medida em que a disponibilidade dos dados para um nó esteja satisfeita. Este modelo pode explorar o paralelismo existente nas operações envolvidas no cálculo numérico de equações diferenciais como $y'' + 2xy' + 2y = 0$, em que várias operações podem ser executadas simultaneamente usando o modelo do grafo de fluxo de dados do problema. Na Figura 1 pode-se constatar que esta equação possui 16 operações elementares: 6 (multiplicação), 2 (adição), 2 (subtração), 3 (duplicação "d"), 1 (condicional "se"), 1 (condicional "<") e 1 (parada "P"). Como mostrado na figura 2.1, inicialmente tem-se 5 nós independentes que podem ser executados simultaneamente, depois outros 5 e assim sucessivamente, seguindo o grafo de fluxo de dados.

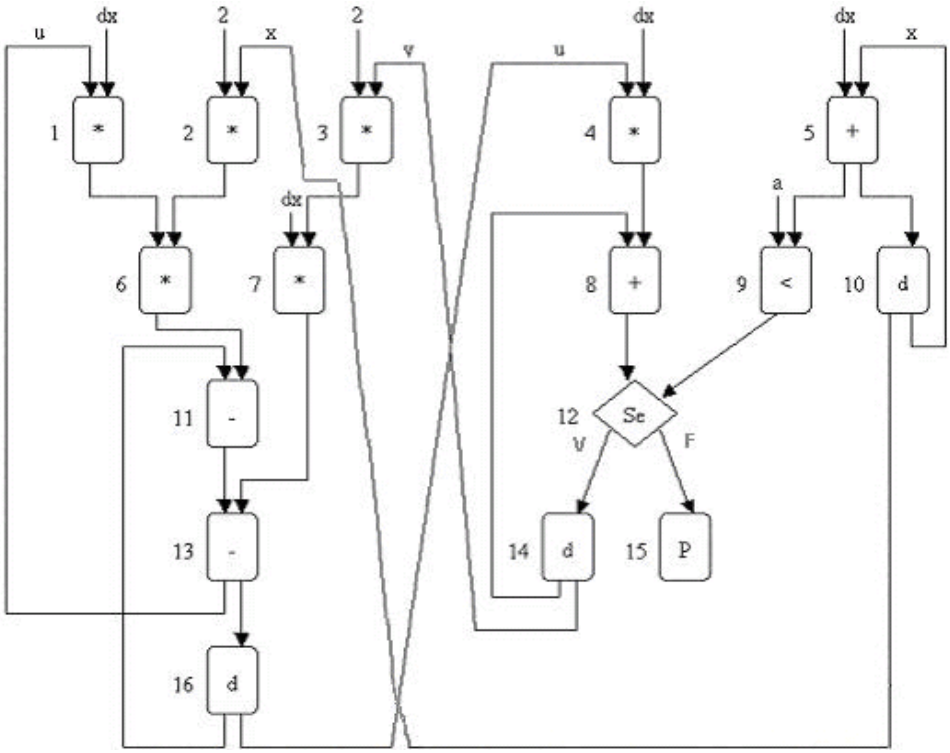


Figura 1: Grafo de Fluxo de Dados da Equação $y'' + 2xy' + 2y = 0$.
(Fonte: FERLIN, 2005)

2.1.3 Computação Reconfigurável

O principal objetivo da Computação Reconfigurável é a utilização de dispositivos lógicos programáveis (PLD – *Programmable Logic Devices)* para dispensar a utilização e construção de circuitos digitais específicos ao projeto, ou dispensar a utilização de processadores, geralmente

não otimizados para a aplicação desejada. A computação reconfigurável pode ser entendida como uma solução intermediária entre o ASIC (*Aplication-Specific Integrated Circuits)* e microprocessadores, combinando vantagens das duas áreas, como mostrado na figura 2:

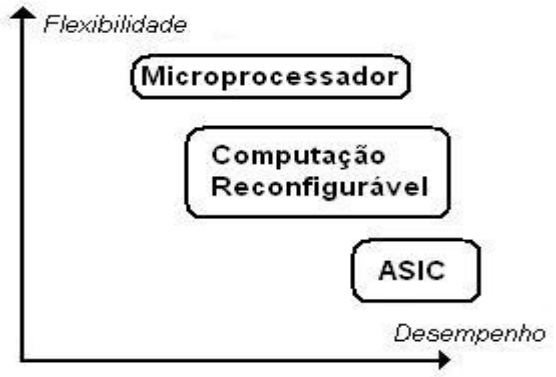


Figura 2 : Posicionamento da Computação Recofigurável
(Fonte: FERLIN, 2005)

As arquiteturas reconfiguráveis são aquelas implementadas por meio de técnicas de reconfiguração. Apresentam blocos lógicos e de interconexão que podem ser reconfigurados. Os blocos lógicos normalmente são as unidades funcionais de processamento, armazenamento, comunicação, entrada e saída de dados (FERLIN, 2005). A história da computação reconfigurável começou com as memórias PROM (*Programmable Read Only Memory*), e posteriormente com dispositivos PAL (*Programmable Array Logic*) e PLA (*Programmable Logic Array*). Na década de 80 foram lançados os primeiros PLDs, que deram origem às tecnologias CPLD (*Complex Programmable Array Logic*) e FPGA (*Field Programmable Gate Array*). Atualmente no mercado podemos encontrar três tipos de FPGAs quanto à sua tecnologia de construção :

- **RAM Estática**: PLD na qual suas conexões entre as portas são feitas entre blocos lógicos por meio de portas de transmissão ou multiplexadores controladas por células SRAM. Tem como vantagem a possibilidade de ser rapidamente configurada, porém exige *hardware* externo auxiliar que deve ser montado junto com os blocos lógicos.
- **Transistores de Passagem**: Essa é uma opção mais barata que a opção de RAM estática, composta por uma grande concentração de transistores que são configurados em modo de corte ou modo de condução.
- **EPROM/EEPROM**: Sua principal vantagem é permitir a reprogramação sem que se precise armazenar a configuração externa.

2.2.4 Dispositivos Lógicos Programáveis

Um FPGA (*Field Programmable Gate Arrays)* é um dispositivo lógico programável que é bastante utilizado para o processamento de informações digitais. Teve o seu lançamento no ano de 1985 como um dispositivo que poderia ser programado de acordo com as aplicações do usuário. FPGAs podem ser utilizados para a implementação de praticamente qualquer projeto de *hardware*. Um dos usos mais comuns, é de prototipação de componentes que virão no futuro a transformar-se em ASIC. A decisão para o uso de FPGAs em produtos finais, requer basicamente uma análise de custo. Possuem flexibilidade e custo baixo de prototipação.

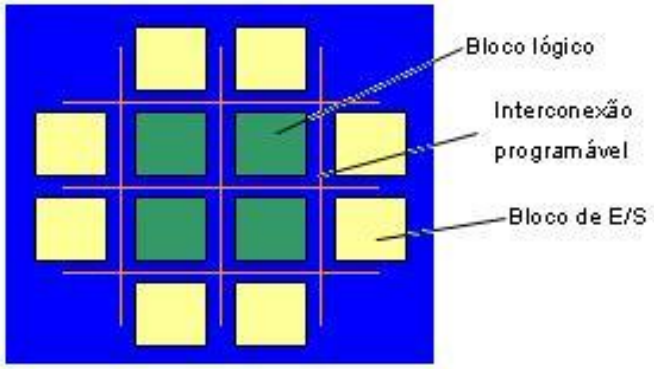


Figura 3: Estrutura interna de um FPGA
(Fonte: BARR, 1999)

O desenvolvimento um FPGA é fruto de uma evolução das PLD/CPLD, mas a sua arquitetura é um tanto distinta deles. Como mostrado na figura 3, a arquitetura de um FPGA é composta de 3 estruturas básicas: Blocos de entrada e saída (IOB), blocos lógicos configuráveis (CLB) e chaves de interconexão.

2.2.5 VHDL

A tecnologia de circuitos *integrados* tem sido aperfeiçoada para permitir adicionar mais e mais componentes em um chip, sendo assim, os sistemas digitais continuam a aumentar a complexidade. O projeto detalhado dos sistemas no nível de portas e flip-flops se tornou monótono e consome muito tempo. Uma linguagem de descrição de hardware permite a um sistema digital ser desenvolvido e depurado em um nível mais alto antes da conversão para o nível de portas e flip-flop.

O seu desenvolvimento foi patrocinado pelo Departamento de Defesa dos Estados Unidos, e a primeira versão foi lançada em 1985 (YALAMANCHILI, 1998). Desde então, a indústria de

automatização de projetos expandiu o uso de VHDL de concepção inicial da documentação, para implementação e verificação funcional (PERRY, 2002).

Projetos em VHDL possuem duas seções principais: ENTITY, que define as portas de entrada e saída, e ARCHITECTURE, que define o funcionamento do projeto. Uma *architecture* pode ser desenvolvida em três modos: fluxo de dados, funcional e estrutural.

CAPITULO 3
3 ESPECIFICAÇÃO DO PROJETO

3.1 Descrição do Projeto

3.1.1 *Software* de Análise de Expressões Matemáticas

O projeto consiste em desenvolver um software para análise de expressões aritméticas, capaz de analisar as expressões e dividi-las por meio de *templates*. Esse processo é feito por meio de análise de expressões com uso de listas encadeadas e teoria de grafos, além de reconhecimento de linguagem através de um compilador que estará encarregado de fazer toda a análise léxica, sintática e semântica. A entrada e saída de dados é feita por um computador ao qual será implantado o *software*, que fará o tratamento dessas expressões matemáticas, e a partir disso, estará gerando alguns conjuntos de informações que são chamados de *templates*, que serão processados na máquina paralela, apresentando assim, os resultados novamente ao computador. A entrada será uma expressão numérica ou uma linguagem definida no compilador. As expressões matemáticas são formadas por números inteiros, operadores (+, -, /, *, ^), parênteses e variáveis (de A a Z). O *software* também faz o reconhecimento e validação de uma linguagem definida no compilador, como exemplo: *var x, z as int*. A entrada é avaliada primeiramente por meio de análises de sintaxe e semântica do compilador. Após feito o reconhecimento da linguagem e análise da expressão matemática, o *software* dividirá a expressão em *templates*, no formato da máquina paralela. A figura 4 mostra a primeira tela do software apresentando um exemplo com uma expressão do segundo grau:

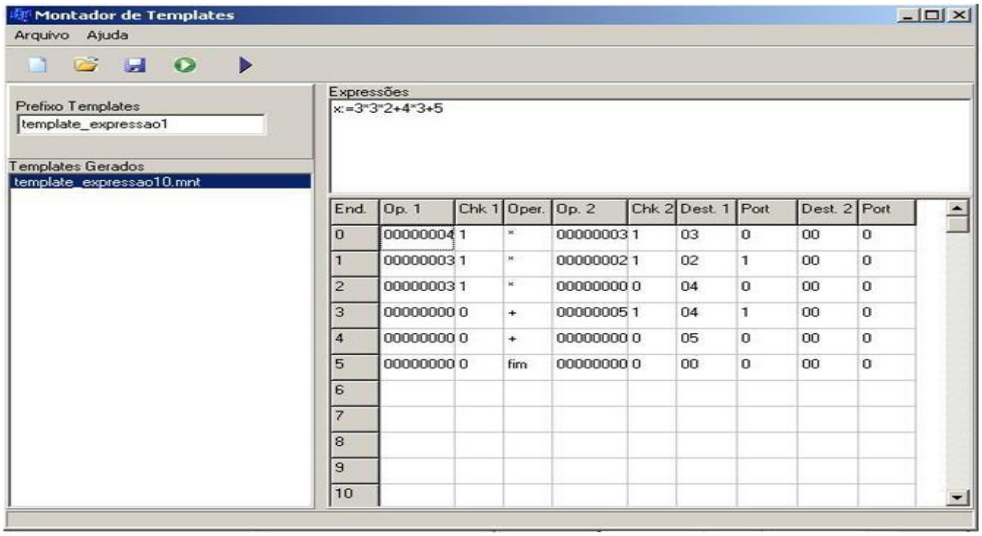


Figura 4. Primeira tela do software montador de templates.

A expressão usada como exemplo na figura 4, é uma equação do segundo grau representada por $2x^2+4x+5$ para x=3. A cada expressão executada, é gerado um arquivo no formato **mnt**. A expressão é decomposta no formato de *template* para envio à máquina paralela

Em seguida, a expressão é processada baseada no modelo de fluxo de dados, ao qual permite obter o máximo do paralelismo existente na aplicação. Os *templates* serão enviados à máquina paralela que fará todo o processamento, devolvendo todo resultado na tela do computador. A Figura 5 demonstra o grafo da expressão do segundo grau $2x^2+4x+5$ apresentada como exemplo, e a Figura 6 o resultado no software montador, em forma de uma tabela do *template* gerado:

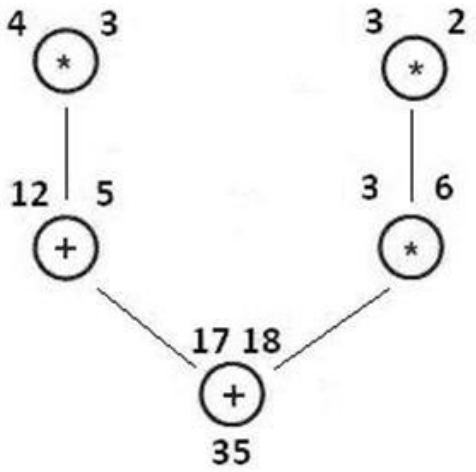


Figura 5: Grafo da equação do segundo grau.

Tabela de *Templates*

Endereço	Op1	Chk Op1	Op2	Chk Op2	Dest 1	Port	Dest 2	Port	Operação
0	3	☑	3	☑	1	0	0	0	*
1	-	☐	2	☑	3	0	0	0	*
2	4	☑	3	☑	3	1	0	0	*
3	-	☐	-	☐	4	0	0	0	+
4	-	☐	5	☑	5	0	0	0	+
5	-	☐	-	☐	-	-	-	-	fim

Figura 6: *Template* Gerado

O resultado do calculo dessa expressão, é demonstrado na Figura 7 abaixo, na linha 5 e coluna do Operando 1:

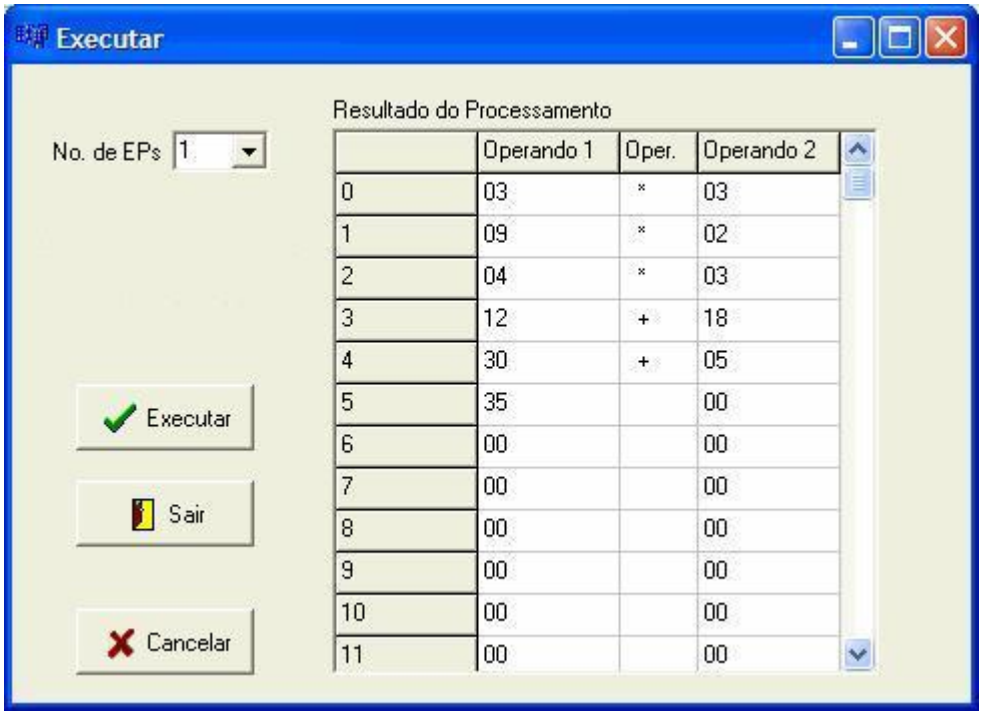


Figura 7: Resultado do processamento da equação do segundo grau.

A comunicação é feita por meio de adaptações com a linguagem da arquitetura paralela (VHDL), protocolo de comunicação USB que foi desenvolvido em C++, através de um *firmware*, gravado no kit da Altera, e o *sofware* montador de *templates*, implementado na linguagem C++. Na Figura 8, mais um forma de represntação do grafo e cálculo de uma equação matemática simples, dada pela seguinte função: $f(x) = 2x^2+4x-5$. A figura 2.1, mostra o grafo do cálculo desta equação:

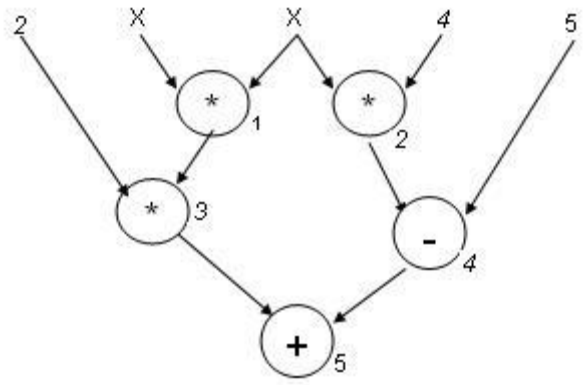


Figura 8: Grafo para cálculo da equação $f(x) = 2x^2+4x-5$.

Observa-se que, para o cálculo dessa equação, são necessárias cinco operações. O software analisará a expressão com o objetivo de dividir o cálculo em partes. No caso dessa equação especificamente, o cálculo seria em 3 passos, pois as instruções 1 e 2 podem ser executadas em paralelo. Posteriormente as operações 3 e 4 e, em seguida, a operação 5. A operação 3 é dependente da operação 1, pois um dos seus operandos é gerado por essa operação, assim como a 4 depende depende de 2 e a 5 depende de 3 e 4.

3.3 Descrição do *Hardware*

O hardware do projeto é composto por um computador, placa de desenvolvimento de lógica programável Altera DE2 e cabo de comunicação.

3.2.1 Computador (Host)

É utilizado no projeto um computador para interface com o usuário. Por meio deste é realizada entrada e saída de dados da máquina paralela, no FPGA. Deve ter como sistema operacional Windows 2000 ou XP e entrada USB para comunicação.

O computador possui o software de Montador de *templates*, que serve para enviar os dados para a máquina paralela, visualizar os *templates* e resultados do processamento, conforme a descrição do software no capítulo 4. Para visualização dos resultados são apresentados ao usuário as operações no formato de templates que é interpretado pela máquina.

A comunicação com a máquina é feita via porta USB, cujo protocolo de comunicação foi desenvolvido em C++ e compilado pelo software NIOSII.

3.2.2 Altera DE2

A placa DE2 da Altera para desenvolvimento de lógica programável, contém o FPGA que opera de acordo com o projeto da máquina paralela, um *firmware* responsável pelo gerenciamento de um núcleo de comunicação do processador NIOS II e do núcleo de controle da máquina paralela. Essas informações são gravadas no kit DE2 da Altera para funcionamento.

A placa DE2 possui diversos recursos para comunicação com periféricos externos, como se pode visualizar na Figura 4.1. Neste projeto foi utilizado apenas a Comunicação JTAG via USB-Blaster:

- Comunicação JTAG USB-Blaster: o JTAG (Join Test Action Group) é o protocolo utilizado entre o software Quartus II e a placa DE2 para a programação do FPGA. Para a transferência de dados é utilizado o cabo de comunicação USB-Blaster.

3.2.3 Cabo de Comunicação

O cabo de comunicação integra o host com a máquina paralela, permitindo a transferência de dados entre eles de acordo com o protocolo desenvolvido em C++ e em VHDL no FPGA, para que o usuário utilize o computador como interface de entrada e saída da máquina paralela, utilizando a porta USB.

3.2.4 Máquina Paralela

A máquina paralela, que foi desenvolvida por meio da computação reconfigurável, ficará encarregada de processar os *templates* enviados pelo *software* de Análise de Expressões. Esse processamento será feito por meio de vários *Elementos Processadores* (EPs) que operam de forma paralela, e que foi desenvolvido com a linguagem VHDL e implementados em um FPGA.

No exemplo da Figura 8 ficou claro a limitação imposta pelas dependências entre as operações. O maior desempenho teoricamente seria obtido com 2 EPs, visto que com mais EPs não haveria ganho, a não ser que todas as operações fossem independentes, aí sim, com 5 EPs seria possível executá-las em apenas 1 passo.

Cada uma das operações corresponde a um template na máquina paralela. Esse *template* será gerado por meio do *software* de análise de expressões, e enviado à máquina. O *template* conterá as seguintes informações: operação, operandos, endereços de destino e sinais de controle. Por exemplo, na equação da Figura 8, a instrução 2 possui como operandos X e 4, a operação multiplicação, e como destino o endereço de memória do *template* 4. A Tabela 1 mostra como seriam os *templates* referentes ao cálculo da Tabela 1.

Tabela 1. *Templates* para cálculo da equação $f(x) = 2x^2+4x+5$.

Template	**Operação**	**Operando1**	**Operando2**	**Destino**
1	Multiplicação	X	X	3
2	Multiplicação	X	4	4
3	Multiplicação	2	-	5
4	Subtração	-	5	5
5	Soma	-	-	-

O funcionamento da máquina paralela depende de outras ações além da execução dos *templates*, assim parte da operação é feita por outras duas unidades da arquitetura além dos EPs, que são chamados de unidades de despacho e unidade de armazenamento. Essas unidades são responsáveis pelo fluxo de dados e instruções da memória para os EPs, e dos EPs para a memória, que aqui chamamos de memória de templates, onde estão armazenados os templates. A Figura 9 apresenta uma visão geral da máquina paralela e do fluxo de dados entre as unidades:

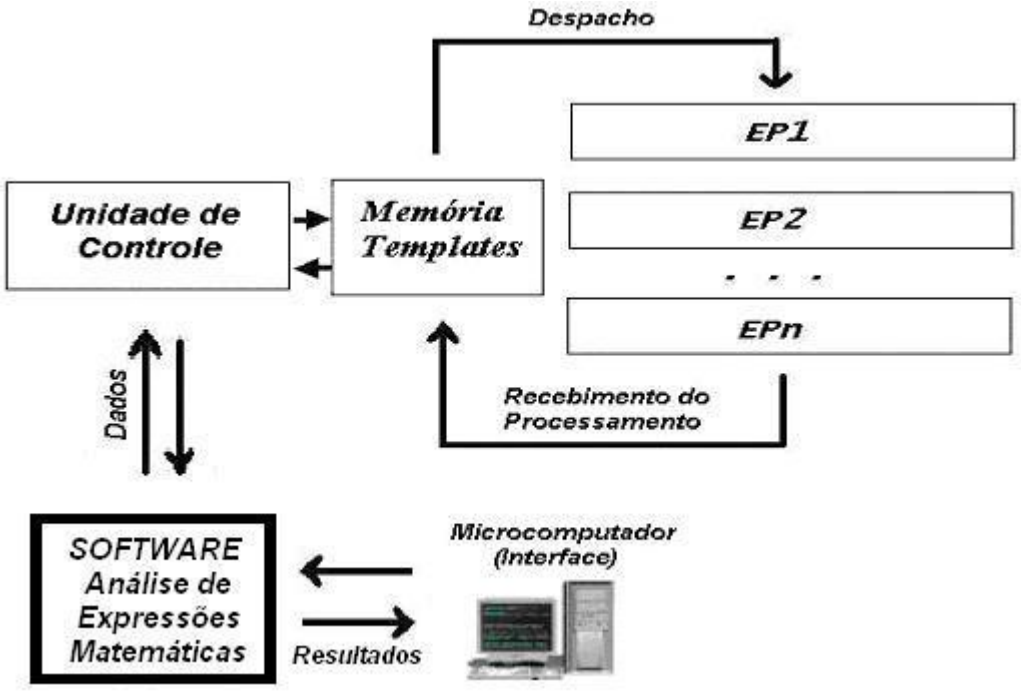


Figura 9: Máquina paralera e fluxo de dados entre as unidades.

Conforme definido anteriormente, a entrada e saída da máquina paralera são feitas por um computador (PC) que possui o *software* de análise de expressões e montagem de *templates*. O *software* faz a comunicação com a máquina paralela enviando os *templates* e recebendo as respostas do processamento diretamente no computador.

3.3 Descrição de *Firmware*

O *firmware* é responsável pelo controle do hardware diretamente, gerenciamento do núcleo de comunicação do processador Nios II e do núcleo de controle da máquina paralela. Possui acesso ao software Gerador de Templates por meio do núcleo de comunicação, utilizando o protocolo USB. Os acessos à máquina paralela são feitos através do núcleo de controle, que tem a responsabilidade de disponibilizar os sinais de controle e comunicação que interligam a máquina paralela e a interface de controle. O firmware foi desenvolvido em C++.

3.3.1 Descrição do Núcleo de Controle

O processador Nios II é um dos componentes de hardware configurado na FPGA. Também serão escritos diversos componentes em linguagem de descrição de hardware. O principal componente de hardware a ser construído em VHDL nesse projeto é chamado de Controlador do Núcleo AP e implementa a interface de comunicação entre o microprocessador Nios II e a máquina paralela de processamento numérico. Todos os componentes escritos linguagem de descrição de hardware serão configurados em uma única FPGA. Além do CORE capaz de realizar a interface, tem-se também um processador, interface de comunicação e diversos controladores de dispositivos presentes no kit de desenvolvimento.

3.3.2 NIOS II IDE

Software utilizado para o desenvolvimento do firmware responsável pelo gerenciamento processador Nios II e seus periféricos. Essa ferramenta disponibiliza um ambiente integrado de desenvolvimento, distribuição e depuração na qual é possível escrever os programas em liguagem C ou C++, compilar, distribuir para um hardware específico e depurar o programa implementado. Essa ferramenta também está integrada ao SOPC Builder, o que permite a utilização do sistema processado sem a necessidade de grande esforço adicional.

CAPITULO 4

4 IMPLEMENTAÇÃO E DESENVOLVIMENTO

Nesse capítulo faz-se o detalhamento do projeto montador de *templates* e os procedimentos adotados para o desenvolvimento.

4.1 Desenvolvimento da Interface do Software

Abaixo são enumerados os requisitos, componentes e as funcionalidades do *software* montador de templates.

4.2 Requisitos do *Software*

Os requisitos necessários para o desenvolvimento do software e funcionamento e integração do projeto com FPGA são listados abaixo:

- Sistema Operacional Windows 2000 ou XP.
- Alterações na Interface como botões, gravações de arquivos, demonstrações de *templates* e espaço para digitações de expressões.
- Implementar compilador básico para identificações de linguagens e validações de expressões aritméticas.
- Implementar funções para geração de *templates* de forma automática a partir de expressões digitadas pelo usuário.
- Limitar desenvolvimento e tratamento de expressões aritméticas aos operadores de soma, subtração, divisão e multiplicação.

4.3 Desenvolvimento do Software

Com o levantamento de requisitos torna-se possível a construção lógica, alterações e melhorias na interface do software montador de templates. A figura X mostra um diagrama simples do início do desenvolvimento da interface, que possue um espaço para entrada de expressões

digitadas pelo usuário, *templates* que são gerados por meio dessas espressões, gravações de *templates* em arquivos, e abertura desses arquivos, para envio à máquina paralela:

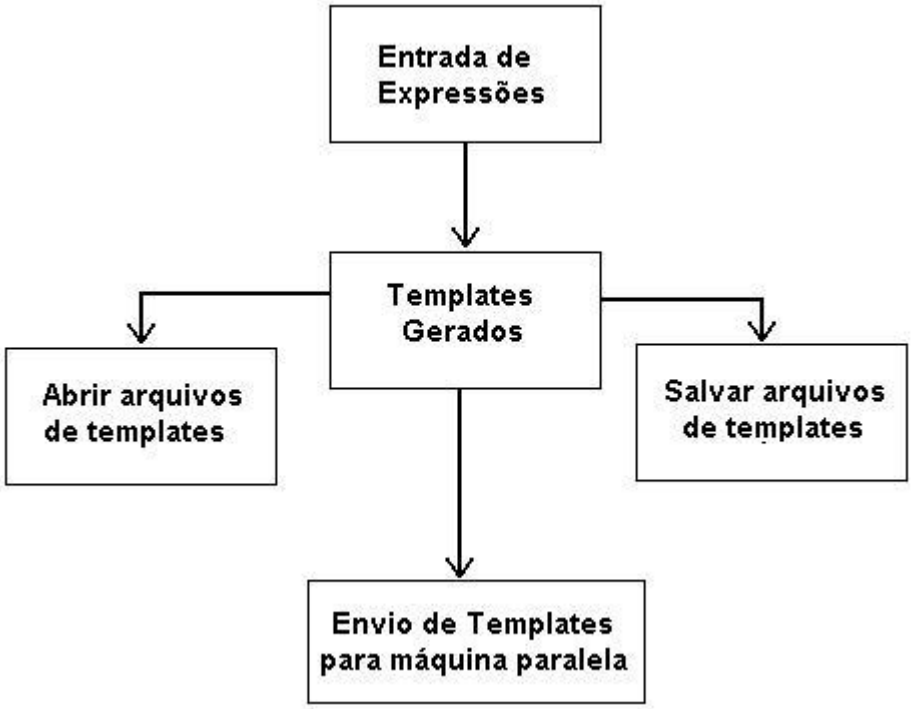


Figura 10. Diagrama básico da interface do software.

Os *templates* são demonstrados na interface de forma visual, conforme formato esperado pela máquina paralela. A expressão é quebrada e separada em campos e colunas através de algoritmo de grafos e funções utilizando listas encadeadas desenvolvidas com linguagem C e C++, que será demonstrado a seguir.

4.4 Desenvolvimento do Compilador

O compilador tem a função de interpretar linguagens simples pré-definidas, validações de expressões aritméticas e detecção de erros de sintaxe. A primeira análise feita pelo compilador é a *Análise Léxica*, através de identificação de *tokens* e palavras reservadas, e validações de linguagens simples. Após essa análise, partimos para a *Análise Sintática* que faz a interpretação das *cadeias de tokens.* A Figura 11 demonstra um diagrama para validação da linguagem *var x as integer* para *n variáveis* digitadas pelo usuário, feita pela *Análise Sintática*:

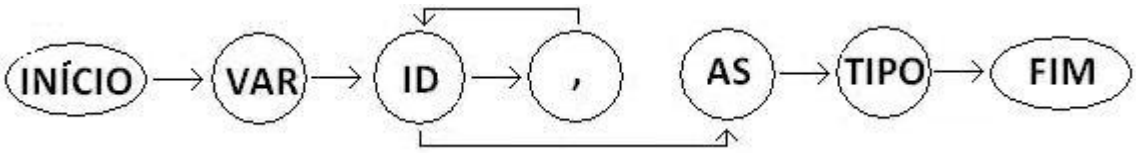


Figura 11. Diagrama para validação de linguagem simples.

4.5 Desenvolvimento do Analisador de Expressões Aritméticas

O desenvolvimento da parte de análise de expressões aritméticas também faz parte do *Analisador Sintático*, desenvolvido no compilador. As expressões são validadas conforme linguagem definida no compilador e regras de análise de expressões matemáticas como validações das quantidades de parênteses e sintaxe da expressão. A Figura 12 demonstra um diagrama que representa a validação das expressões aritméticas digitadas pelo usuário no *software* montador:

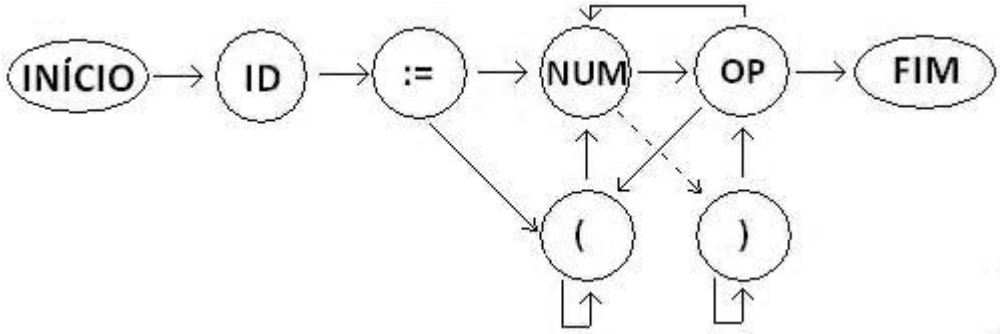


Figura 12: Diagrama para validação de expressões aritméticas

A linguagem e sintaxe do compilador são limitadas à apenas operadores de soma, subtração, divisão e multiplicação, validações de parênteses apenas e variáveis (de A a Z). As expressões são processadas uma por vez. Cada expressão gera um único arquivo de *template* para envio à máquina paralela. O espaço para digitações suporta até 99 expressões, com um máximo de 254 operações. A figura 13 demonstra a tela principal do software, usando como exemplo a expressão $2x^2+4x+5$ para x =3, que o software interpreta pela seguinte sintaxe: X:=3*3*2+4*3+5. Na tela é demonstrado também, logo abaixo do espaço para entrada de expressões matemáticas, o formato do template que

pertente à expressão citada anteriormente, e que foi criado a partir de algoritmos utilizando grafos e listas encadeadas:

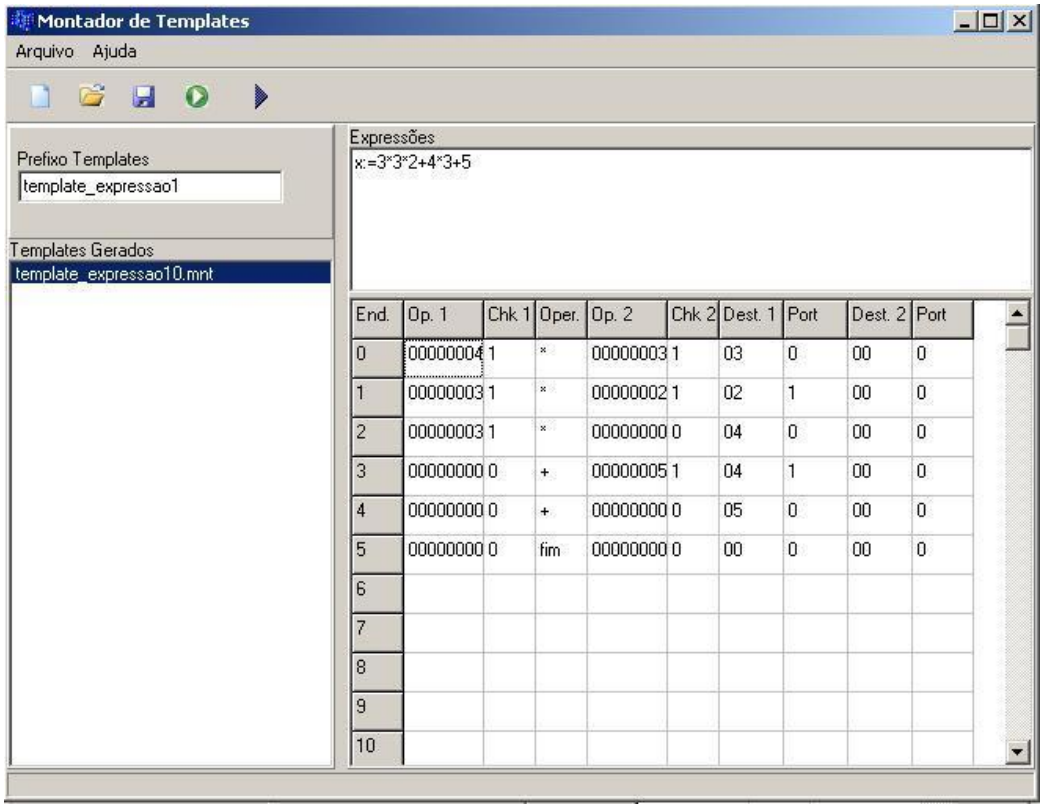


Figura 13. Tela Principal do software

O grafo dessa expressão é demonstrado na figura 14, que utiliza todo o paralelismo existente na equação:

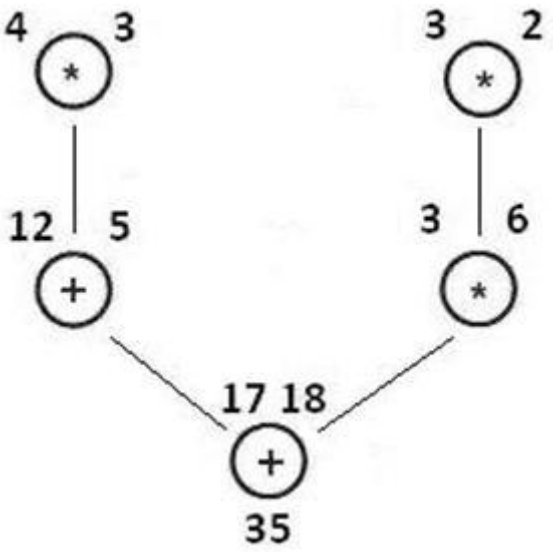
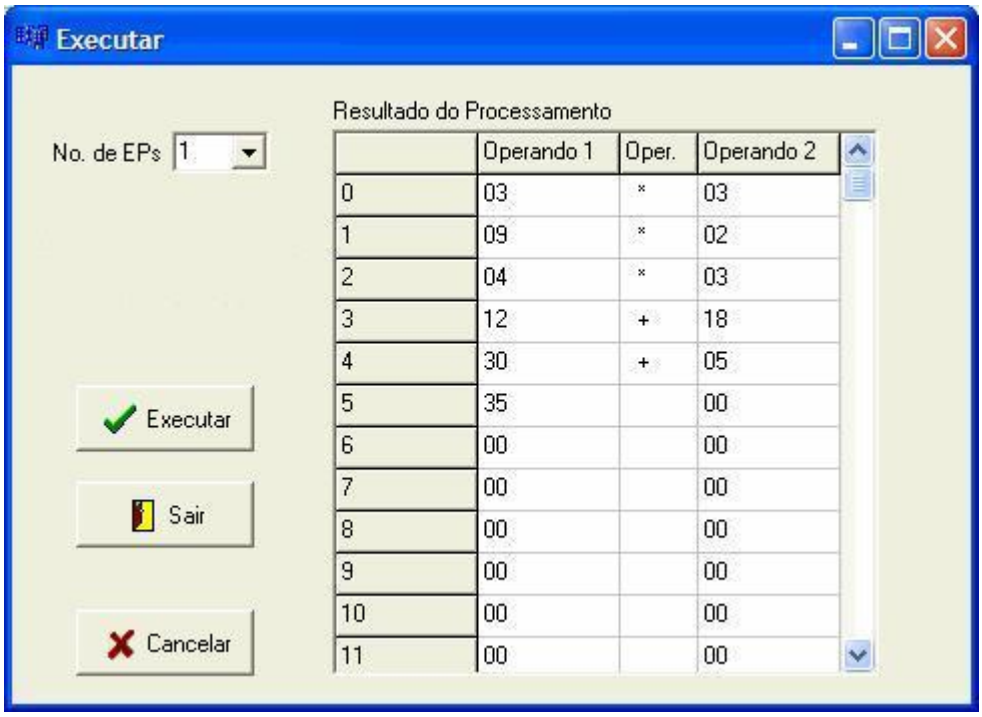


Figura 14. Grafo da expressão $2x^2+4x+5$.

O resultado dessa equação é demonstrado na figura 15. O resultado é demostrado na linha 5, coluna Operando1. As equações são resolvidas conforme o paralelismo existente na equação, e os resultados de cada operação são resolvidos conforme as dependências existentes entre uma operação e outra.



	Operando 1	Oper.	Operando 2
0	03	*	03
1	09	*	02
2	04	*	03
3	12	+	18
4	30	+	05
5	35		00
6	00		00
7	00		00
8	00		00
9	00		00
10	00		00
11	00		00

Figura 15. Tela com resultado da expressão $2x^2+4x+5$.

4.6 Desenvolvimento da comunicação e envio de dados

Para a comunicação e envio de informações para a máquina paralela, foi utilizado o mesmo código desenvolvido no projeto de 2007 (Ceccon, 2007), com pequenas adaptações com o novo software montador de *templates*.

4.7 Compilação do Projeto da Máquina Paralela

Para compilação e funcionamento da Máquina Paralela, foi necessário a compilação do projeto de 2006 (Brodzinski, 2006), utilizando a ferramenta Quartus II da Altera, e gravação das informações em FPGA no kit DE2 da Altera.

4.8 Compilação do Projeto do NIOS II

Para compilação e funcionamento do Processador NIOS II e interface USB desenvolvida no projeto de 2007 (Ceccon, 2007), foi utilizado a ferramenta NIOS II da Altera, e gravação das informações em FPGA no Kit DE2 da Altera.

4.9 Gravações em FPGA

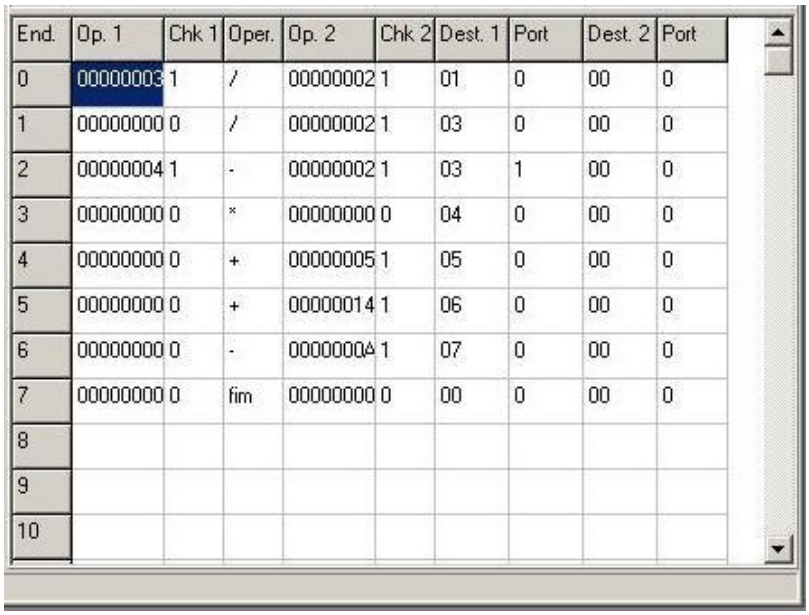
Para o funcionamento completo do projeto, foi necessário a gravação dos códigos já compilados da máquina paralela e do processador Nios II. As gravações foram feitas através de comunicação USB e envio de informações para o Kit DE2 da Altera.

CAPITULO 5

5 Resultados Obtidos.

5.1 Demonstração dos resultados

Como resultado do projeto obtém-se a montagem e a transferência de *templates* para o Núcelo AP, onde o processamento é realizado. Na figura 16 é mostrado no software um *template* pronto para ser enviado à máquina Paralela.



End.	Op. 1	Chk 1	Oper.	Op. 2	Chk 2	Dest. 1	Port	Dest. 2	Port
0	00000003	1	/	00000002	1	01	0	00	0
1	00000000	0	/	00000002	1	03	0	00	0
2	00000004	1	-	00000002	1	03	1	00	0
3	00000000	0	*	00000000	0	04	0	00	0
4	00000000	0	+	00000005	1	05	0	00	0
5	00000000	0	+	00000014	1	06	0	00	0
6	00000000	0	-	0000000A	1	07	0	00	0
7	00000000	0	fim	00000000	0	00	0	00	0
8									
9									
10									

Figura 16: Demonstração de *templat*e no software.

Na Figura 4.2 pode ser visualizado o cálculo realizado por meio da tela do *software* montador. O resultado é apresentado na linha 5 e coluna Operador 1.

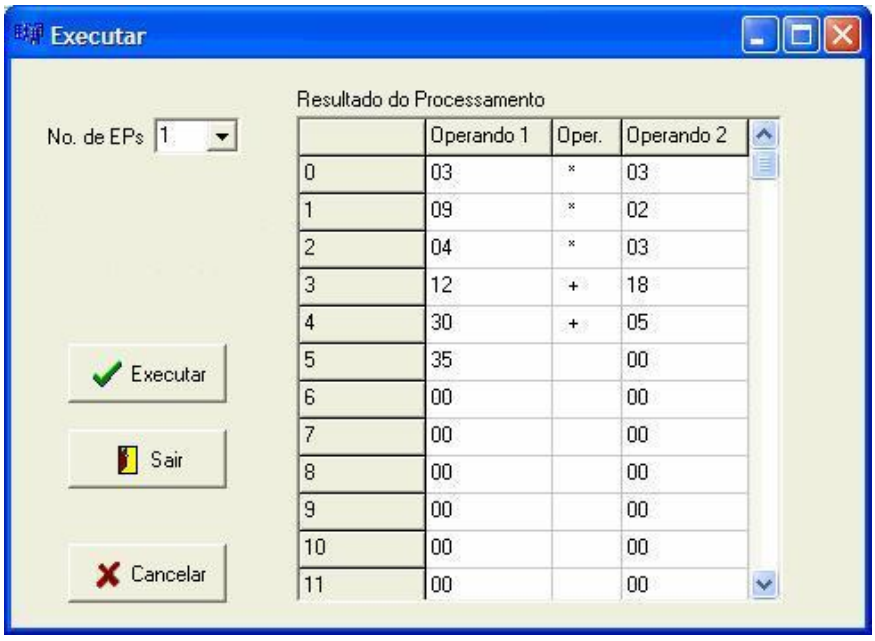


Figura 17: Exemplo de resultado após processamento.

5.2 Problemas Encontrados

Enumeram os principais problemas encontrados durante o desenvolvimento e implementação dos componentes do projeto

- Durante o desenvolvimento do software montador de *templates*, que foi feito utilizando as linguagens C e C++, foi encontrado muitos problemas com a utilização de ponteiros e erros de acesso a memória, devido à utilização de listas encadeadas na estrutura do programa, do desenvolvimento do compilador e no tratamento de expressões.
- Durante o processo de compilação do projeto de 2007, foi encontrado diversos problemas em versões dos softwares do NIOS II disponíveis no site da Altera, ao qual houve uma grande dificuldade na reconfiguração e compilação do código fonte, para gravação no Kit da Altera.
- Durante o processo de compilação do projeto de 2006, houve uma grande dificuldade para gravações do código compilado no kit da altera, devido à versão atualizada do software Quartus II não ser totalmente compatível com o código desenvolvido naquele ano.

- Durante a gravação dos dois projetos no Kit Altera, foi encontrado problemas de comunicação com a porta USB, ao qual foi verificado a necessidade da configuração do driver USB-Blaster, no software Quartus II.

CAPÍTULO - 6
5. CONCLUSÃO

Baseado nos resultados obtidos nesse projeto, pode-se se concluir que a utilização de compiladores no desenvolvimento de análise de expressões matemáticas, se torna muito produtiva e eficiente, utilizando toda teoria que envolve também alguns conceitos de linguagens formais, como análise sintática e semântica, além de conceitos sobre recursividade em linguagens de programação, principalmente na organização do código e validações de variáveis que possam a ser utilizadas nessas expressões. Podemos também observar todo o paralelismo existente em equações aritméticas, afim de se obter o maior desempenho possível através da arquiteturas de processamento paralelo, pois uma grande quantidade de EPs utilizados pela máquina, nem sempre aumenta a velocidade em processamentos envolvendo expressões matemáticas.

6.1 TRABALHOS FUTUROS

Nesse capítulo apresenta-se todas as melhorias que podem dar continuidade no desenvolvimento do projeto:

- Aumentar quantidade de expressões processadas ao mesmo tempo, pois o software trata apenas uma expressão por vez.
- Desenvolvimento na parte de otimização das expressões, melhorando e aumentando ainda mais o paralelismo existente em cada expressão.
- Utilização de variáveis nas expressões, que pode ser desenvolvida utilizando consultas nas tabelas de símbolos já desenvolvida no compilador.
- Melhoria no código do compilador, para reconhecimento de loops e condicionais, o que possibilitaria o envio de uma grande quantidade de expressões e processamentos.
- Desenvolvimento de código no software montador de templates para tratamento de números reais, assim que a máquina esteja reconhecendo dados desse tipo.

REFERÊNCIAS BIBLIOGRÁFICAS

- ANTÔNIO EDSON CECCON. **Interface Reconfigurável Para Arquitetura Paralela Baseada em Processador NIOS II.** TCC Engenharia da Computação. Universidade Positivo. Curitiba, 2007.
- MAURICIO VITOR BRODZINSKI. **Máquina Paralela Reconfigurável Baseada em Fluxo de Dados a Cálculo Numérico.** TCC Engenharia da Computação. Universidade Positivo. Curitiba, 2006.
- ANTONIO FERNANDO TRAINA. **Proposta de Construção de um Compilador PASCAL para Arquitetura RISC-LIE.** São Paulo,1993.
- TANENBAUM, A. S. **Organização Estruturada de Computadores.** Rio de Janeiro, LTC, 2001.
- AHO, A. V. – SETHI, R. – ULLMAN, J.D. **Compiladores Princípios, Técnicas e Ferramentas.** Rio de Janeiro, 1995.
- SCHILDT, H.. **C Completo e Total.** 3ª Ed. São Paulo, LTC, 1997.
- ROTH, C. H. **Digital Systems Design Using VHDL.** Boston. PWS Publishing, 1998.
- YALAMANCHILI, S. **VHDL starter's guide.** Upper Saddle River, New Jersey, Prentice Hall, 1998.
- ALTERA CORPORATION, **Quartus II Version 6.1 Handbook - Volume 4 SOPC Builder** disponível em <http://www.altera.com/literature/manual/qts_qii5v4.pdf>. Acesso em Agosto de 2008.
- ALTERA CORPORATION, **Introduction to the Altera SOPC Builder** disponível na em <http://www.altera.com/literature/hb/qts/tut_sopc_introduction.pdf>. Acesso em Agosto de 2008.
- ALTERA CORPORATION, **Nios II Processor Reference Handbook** disponível na Internet em <http://www.altera.com/literature/hb/qts/n2cpu_nii5v1.pdf>. Acesso em Setembro de 2008.
- ALTERA CORPORATION, **DE2 Development and Education Board - User Manual** disponível em <http://www.terasic.com/literature/DE2_UserManual_1.4_final.pdf>. Acesso em Agosto de 2008.