CAPÍTULO V

IMPLEMENTAÇÃO DO ELEMENTO PROCESSADOR

V.1 - Arquitetura Final do Elemento Processador

O estudo de diversas arquiteturas e algoritmos para solução de operações aritméticas inteiras apresentado no Capítulo III definiu a estrutura básica do EP. A partir da análise das operações aritméticas reais discutida no Capítulo IV, esta arquitetura básica foi aprimorada, resultando no EP apresentado na Figura IV.2. Para completar o projeto do elemento processador, neste capítulo é definida a comunicação do EP com os outros EPs e com a Unidade de Controle.

V.1.1 - Comunicação entre EPs

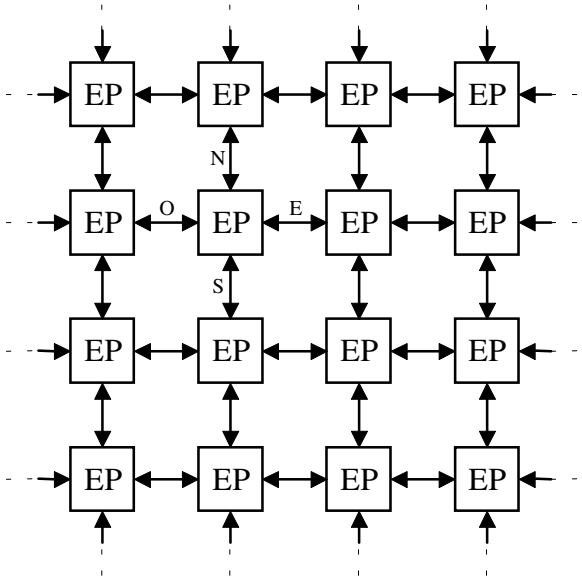

Figura V.1 - Comunicação entre EPs *bit* a *bit* com os quatro vizinhos mais próximos

Um ponto essencial a ser considerado na comunicação entre EPs numa estrutura do tipo matriz é que nem todos os processadores estarão no mesmo *chip* e nem todos os processadores estarão na mesma placa. Ou seja, é fundamental analisar o peso das ligações entre *chips* e entre placas, visto que estas ligações têm um custo mais elevado do que o das ligações internas. Um aumento das ligações entre *chips* implica num maior número de pinos e,

conseqüentemente, num aumento da área do *chip*. Um aumento das ligações entre placas implica num maior número de conectores e cabos, aumentando as possibilidades de interferências entre sinais e defeitos por mal contato e a probabilidade de falhas.

A estrutura de troca de dados entre EPs encontrada na maioria das máquinas SIMD é a da comunicação *bit* a *bit* com os quatro vizinhos mais próximos: norte (N), sul (S), este (E) e oeste (O); como mostra a Figura V.1. Para uma matriz n x n, esta configuração requer $2n^2 - 2n$ fios para as ligações internas e $4n$ fios para as ligações nas bordas da matriz. Ou seja, cerca de dois fios por EP. Por exemplo, se forem encapsulados 16 EPs (matriz 4 x 4) dentro de cada *chip*, serão necessários 16 pinos para as comunicações externas entre *chips* de EPs.

Como a arquitetura em questão é um processador de 4 *bits*, poder-se-ia adotar canais de comunicação de 4 *bits*. Porém, isto aumentaria em muito o custo das ligações. Seriam necessários $8n^2 - 8n$ fios para as ligações internas e $16n$ fios para as ligações nas bordas da matriz. Ou seja, cerca de oito fios por EP. Para o mesmo exemplo anterior (*chip* com 16 EPs), esta configuração demandaria 64 pinos por *chip* só para as comunicações externas entre *chips* de EPs. O ganho de desempenho na comunicação de dados entre EPs não justifica um aumento tão elevado do número de pinos, que praticamente inviabiliza o encapsulamento de vários EPs num único *chip*. Sendo assim, esta opção foi descartada.

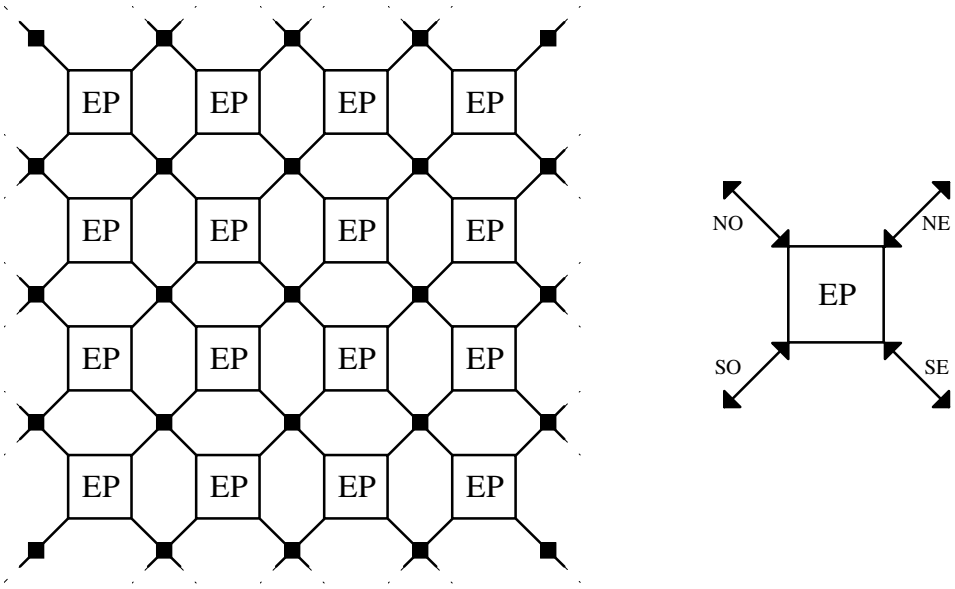

Figura V.2 - Configuração da rede de interconexões entre EPs

Uma possibilidade de melhorar o desempenho da comunicação a um custo bem mais baixo é utilizar um esquema de comunicação com os oito vizinhos mais próximos: norte (N), nordeste (NE), este (E), sudeste (SE), sul (S), sudoeste (SO), oeste (O) e noroeste (NO). Inicialmente pode parecer que isto dobraria o número de canais de comunicação, mas se for adotado o mesmo esquema utilizado no BLITZEN [Blev], não haverá aumento de custo nas

ligações. Este esquema, descrito no Capítulo II é repetido na Figura V.2 por conveniência. A cada EP são associados quatro fios direcionados aos vizinhos nas diagonais (nordeste, sudeste, sudoeste e noroeste). Na interseção entre duas linhas que se cruzam é acrescentada uma conexão física, unindo-as.

A idéia é obter a direção da comunicação de dados através da combinação das direções de envio e recebimento de dados. Por exemplo, para rotear dados na direção oeste, basta que cada EP envie dados para a direção noroeste e receba dados da direção nordeste. A Tabela V.1 apresenta as diversas combinações de envio e recebimento com os respectivos roteamentos obtidos.

O único custo adicional imposto por este modelo é um aumento no controle do EP. Para uma comunicação com quatro vizinhos, são necessários apenas dois *bits* para definir a direção da comunicação. Já numa comunicação com oito vizinhos, são necessários pelo menos três *bits* para definir a direção da comunicação. Como será apresentado mais adiante, para este processador em especial, isto não resultou num aumento da palavra de controle do EP, pois a direção do roteamento é dada por um conjunto de *bits* também utilizado para outros fins.

Tabela V.1 - Combinações de direções e roteamentos obtidos

Envio	Recebimento	Roteamento
NE	NE	Inválido
NE	SE	N
NE	SO	NE
NE	NO	E
SE	NE	S
SE	SE	Inválido
SE	SO	E
SE	NO	SE
SO	NE	SO
SO	SE	O
SO	SO	Inválido
SO	NO	S
NO	NE	O
NO	SE	NO
NO	SO	N
NO	NO	Inválido

Escolhidas a largura dos canais de comunicação e a topologia dos mesmos, falta definir a origem e o destino dos dados transmitidos. Para facilitar a comunicação de dados de quatro *bits*, foi escolhida como destino dos dados a entrada serial esquerda do registrador de deslocamento SH e como origem o conteúdo do barramento principal de um *bit*. Sendo assim, para transmitir palavras de quatro *bits*, basta carregá-las no registrador SH e executar quatro passos de transmissão, colocando como entrada do barramento principal de um *bit* a saída

serial direita de SH. Cabe ressaltar que, em cada operação de transmissão, o conteúdo do registrador SH é deslocado uma posição para a direita. Ao final dos quatro passos, a palavra transmitida se encontrará no registrador SH.

V.1.2 - Comunicação entre os EPs e a Unidade de Controle

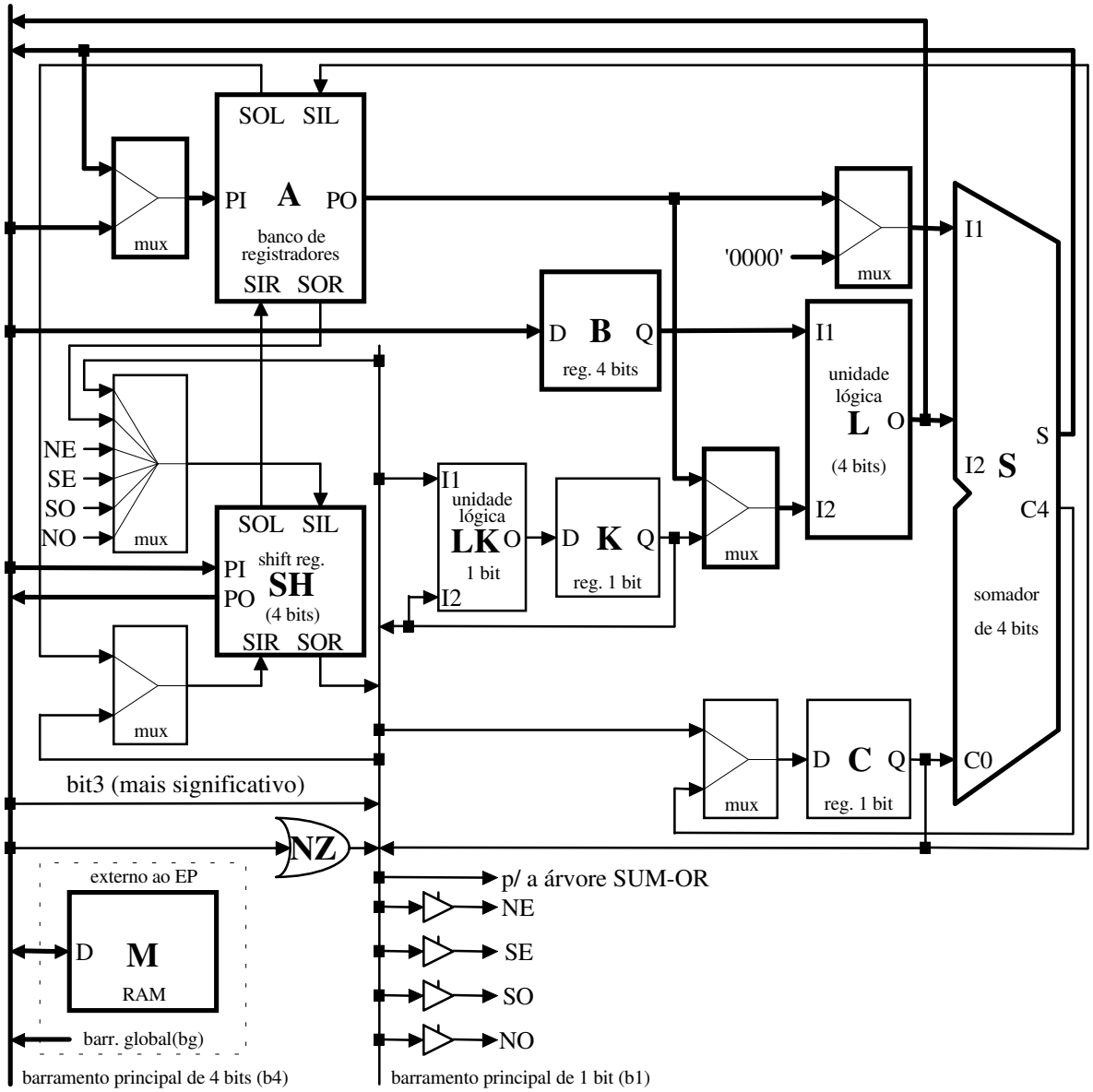

Figura V.3 - EP de 4 bits com banco de registradores

Além da comunicação entre processadores, também é importante definir um mecanismo que permita à Unidade de Controle (UC) tomar decisões baseada em resultados obtidos por um ou mais elementos processadores. Uma importante aplicação deste tipo de mecanismo é o tratamento de erros ou exceções durante o processamento. Por exemplo, numa operação de divisão pode-se identificar facilmente se houve uma exceção do tipo divisão por zero em algum dos processadores. Uma outra aplicação é utilizar os resultados de uma operação na definição

de um teste de parada de um laço. Ou seja, só encerrar a execução do laço quando todos os processadores tiverem atingido um determinado objetivo.

O mecanismo mais utilizado em arquiteturas SIMD para comunicação entre a matriz de processadores e a UC é uma "árvore OU". Por este mecanismo, cada EP envia um *bit* para a UC. Todos os *bits* enviados passam por uma árvore de portas OU-inclusivo, gerando um único *bit* de sinalização para a UC. Na verdade a "árvore OU" pode ser vista como uma grande porta OU de todos os *bits* enviados. O conteúdo do barramento principal de 1 *bit* foi escolhido para ser enviado à UC. Esta escolha visa aumentar a flexibilidade da comunicação. Através dela é possível enviar não só o conteúdo de elementos de 1 *bit*, como o registrador de *carry* (C) ou o registrador de máscara (K), como também informações sobre elementos de 4 *bits*, através do registrador de deslocamento SH, do operador NZ, ou do *bit* mais significativo do barramento principal de 4 *bits*.

A Figura V.3 apresenta a estrutura do EP após a inclusão dos mecanismos de comunicação aqui descritos.

V.1.3 - Palavra de Controle

Para completar a descrição do EP é necessário definir o formato do microcódigo que controlará o mesmo. Ao definir esta palavra de controle, evitou-se limitar as possibilidades de executar diversas operações simultaneamente, tornando a máquina mais versátil. É interessante que se possa, por exemplo, executar num mesmo ciclo uma soma de 4 *bits* e uma operação lógica de 1 *bit*. Este tipo de simultaneidade foi inclusive suposto possível no desenvolvimento dos algoritmos apresentados nos capítulos anteriores.

Os seguintes controles são necessários ao EP4BR:

- banco de registradores A - seleção do registrador (para escrita ou leitura), seleção de entrada (serial e paralela), escrita (condicional a K ou incondicional) e deslocamento (para a esquerda ou para a direita, condicional a K ou incondicional);
- registrador B - escrita;
- registrador K - escrita;
- registrador C - seleção de entrada e escrita;
- registrador de deslocamento SH - seleção das entradas seriais, escrita e deslocamento (para a esquerda ou para a direita);
- unidade lógica L - seleção de entrada e operação efetuada;
- unidade lógica LK - operação efetuada;
- somador S - seleção de entrada;
- comunicação com os vizinhos - seleção de direção e transmissão;
- memória - escrita, leitura e seleção de endereço;

- barramento principal de 4 *bits* - seleção de entrada;
- barramento principal de 1 *bit* - seleção de entrada.

Na definição da palavra de controle não foi incluído o endereço da memória citado acima. Como a UC pode oferecer várias formas de endereçamento, foi considerado que o endereço encontra-se em outra palavra.

Se cada um dos controles descritos acima fosse absolutamente independente, seriam necessários 45 *bits* para formar a palavra de controle, além do endereço da memória. Porém, existem operações cuja simultaneidade é impossível, como a escrita e o deslocamento no banco de registradores A. Em outras, a simultaneidade é inútil, como a seleção da entrada serial e a escrita do registrador de deslocamento SH. Além dessas reduções óbvias e de algumas um pouco mais elaboradas, criou-se uma única restrição: não é permitida uma operação lógica usando LK no mesmo ciclo de uma comunicação de dados entre EPs. Isto porque, para diminuir a largura da palavra de controle, os mesmos *bits* são utilizados para definir a direção da comunicação e a operação lógica efetuada por LK. Esta restrição não é grave, visto que, durante a comunicação, o barramento de 1 *bit* é utilizado como fonte de dados, o mesmo ocorrendo com uma operação lógica, o que já tornaria por si só esta simultaneidade de operações bastante restrita.

Com as reduções feitas, chegou-se a uma palavra de controle de 31 *bits*, excetuando o endereço de memória. Como normalmente as palavras têm como largura uma potência de dois, pode-se pensar numa palavra de controle de 32 *bits*, com 1 *bit* de reserva, que pode ser usado para expansões futuras ou para distinguir uma palavra de controle do EP de uma palavra de controle da UC, por exemplo.

Os 31 *bits* definidos para a palavra são divididos pelos elementos que compõem o EP da seguinte forma:

- banco de registradores A - 7 *bits*;
- registrador B - 1 *bit*;
- registrador K - 1 *bit*;
- registrador C - 3 *bits*;
- registrador de deslocamento SH - 3 *bits*;
- unidade lógica L - 5 *bits*;
- unidade lógica LK e comunicação com os vizinhos - 4 *bits*;
- somador S - 1 *bit*;
- memória e barramento principal de 4 *bits* - 3 *bits*;
- barramento principal de 1 *bit* - 3 *bits*.

Uma descrição mais detalhada da palavra de controle é apresentada no Apêndice B.

V.2 - Simulação Lógica do EP

A fim de avaliar não só o funcionamento das diversas propostas de arquiteturas apresentadas, como também a validade dos algoritmos desenvolvidos, foi implementado um simulador lógico hierárquico denominado SLAP (Simulador Lógico de Arquiteturas de Processadores). Através deste simulador, foi possível identificar e corrigir, ainda na fase de desenvolvimento, erros de projeto nas arquiteturas, assim como nos algoritmos. A avaliação do desempenho dos algoritmos também pode ser conferida.

A escolha de uma simulação a nível lógico e não a nível elétrico se deve principalmente a lentidão dos processos de simulação elétrica. Além disto, a simulação elétrica não acrescenta informações significativas a esta etapa do projeto, principalmente levando-se em consideração que ela é extremamente dependente da implementação. Sendo assim, a simulação a nível elétrico foi deixada para a etapa de implementação do EP.

Para facilitar o trabalho do usuário, a descrição da arquitetura é feita com o auxílio de um editor gráfico hierárquico denominado EGHA, capaz de gerar um arquivo de descrição de arquitetura. Este editor permite descrever a arquitetura com alto nível de abstração. Para isto ele possui uma biblioteca de elementos contendo diversos tipos de registradores, decodificadores, multiplexadores, etc. Uma arquitetura é descrita então por: elementos, blocos, entradas, saídas e os barramentos e nós que ligam os mesmos. Um bloco é um outro arquivo de descrição de arquitetura. Através dos blocos é implementada a estrutura hierárquica do editor e do simulador.

A opção por um editor e um simulador hierárquicos teve por objetivo simplificar a utilização dos mesmos, assim como facilitar uma análise futura do funcionamento conjunto de vários processadores.

Muitas vezes vários barramentos concorrem a uma mesma entrada de um elemento. Para selecionar qual deles será responsável por enviar dados à entrada é usado um multiplexador. Como se pode observar na figura V.3, só no EP4BR esta situação ocorre seis vezes. Visando auxiliar o usuário, o editor permite que uma determinada entrada de um elemento seja repetida diversas vezes, cada vez ligada a um barramento diferente. Neste caso, o simulador interpreta que implicitamente existe um multiplexador selecionando qual barramento será responsável por enviar dados a esta entrada do elemento. O simulador se encarrega ainda de reservar *bits* no microcódigo para controlar este multiplexador. O mesmo mecanismo (consideração de um multiplexador implícito e criação dos devidos *bits* de controle) ocorre quando várias saídas de diversos elementos concorrem a um mesmo barramento.

Por último, como se trata de um simulador de arquiteturas de processadores, não é necessário ligar todos os controles dos elementos a uma entrada externa ou a uma saída de um outro elemento. Para cada controle não declarado no arquivo de descrição da arquitetura, o simulador cria os *bits* de controle necessários no microcódigo. Estas simplificações não só auxiliam o trabalho do usuário, como também deixam o desenho da arquitetura mais simples.

O editor gráfico e o simulador foram desenvolvidos em linguagem C, para um ambiente DOS e um microcomputador compatível com o IBM PC, utilizando como ferramenta o compilador Borland C++ 2.0. Os módulos que compõem o programa fonte do editor totalizam cerca de 8000 linhas e os do simulador totalizam cerca de 7700 linhas.

O programa fonte a ser simulado é um arquivo do tipo texto com um formato padrão (apresentado no Apêndice C) definido especialmente para o simulador. O SLAP possui um pequeno montador capaz de transformar um programa fonte armazenado em disco (no formato padrão) em um programa objeto armazenado na memória (vetor de instruções codificadas).

Carregado o programa na memória, o SLAP permite que se escolha as variáveis que serão vistas durante a simulação. As instruções podem ser simuladas passo a passo ou em rajada. Além do conteúdo das variáveis escolhidas a cada instrução, o simulador também exibe os comentários da última instrução simulada.

Abaixo é apresentado o algoritmo básico da simulação, com exceção dos comandos de controle do fluxo de instruções.

```
para todos os barramentos
    valor do barramento = estado indefinido
para todos as entradas
    valor da entrada = estado indefinido
para todos as saídas
    valor da saída = estado indefinido
para todos as variáveis internas dos elementos
    valor da variável interna = estado indefinido
para cada instrução do programa
    montar o microcódigo definido pela instrução
    calcular o valor de todas as entradas definidas pelo microcódigo
    de tempo = 0 até tempo = período, com acréscimos de delta_t
        calcular o valor de todas as entradas (a partir do valor atual dos barramentos)
        simular o funcionamento de todos os elementos, calculando as suas saídas
        calcular o novo valor dos barramentos, verificando possíveis conflitos
    mostrar o valor atual de todas as variáveis escolhidas
```

Utilizando o SLAP, foram testadas as versões mais completas das cinco arquiteturas propostas: EP1, EP1SH, EP1FIFO, EP4 e EP4BR. Para as quatro primeiras foram testados os algoritmos de soma, subtração, multiplicação e divisão de inteiros. Para a arquitetura escolhida (EP4BR) foram testados também os algoritmos de soma e multiplicação em ponto-flutuante.

Tabela V.2 - Relação de operações inteiras, operandos e resultados simulados

Operação	Operandos	Resultado esperado/obtido
$z = x + y$	$x = +54, y = +51$	$z = 105$
$z = x + y$	$x = -51, y = +54$	$z = 3$
$z = x + y$	$x = +54, y = -54$	$z = 0$
$z = x + y$	$x = -54, y = -51$	$z = -105$
$z = x - y$	$x = +54, y = +51$	$z = 3$
$z = x - y$	$x = -51, y = +54$	$z = -105$
$z = x - y$	$x = +54, y = -54$	$z = 105$
$z = x - y$	$x = -54, y = -51$	$z = -3$
$p = x \times y$	$x = +54, y = +51$	$z = 2754$
$p = x \times y$	$x = -51, y = +54$	$z = -2754$
$p = x \times y$	$x = +54, y = -54$	$z = -2916$
$p = x \times y$	$x = -54, y = -51$	$z = 2754$
$q, r = x / y$	$x = +54, y = +51$	$q = 1, r = 3$
$q, r = x / y$	$x = -51, y = +54$	$q = 0, r = -51$
$q, r = x / y$	$x = +54, y = -54$	$q = -1, r = 0$
$q, r = x / y$	$x = -54, y = -5$	$q = 10, r = -4$

Para cada operação aritmética foram utilizados quatro conjuntos de operandos, visando abranger o maior número de casos possível. Sendo assim, foram escolhidos conjuntos de operandos e respostas esperadas com diversas combinações de sinais. O Apêndice D apresenta um exemplo de cada programa simulado. A Tabela V.2 apresenta uma relação dos operandos testados para cada operação aritmética inteira e a Tabela V.3 apresenta uma relação dos operandos testados para cada operação aritmética real.

Tabela V.3 - Relação de operações reais, operandos e resultados simulados

Operação	Operandos	Resultado esperado/obtido
$z = x + y$	$x = +0, y = -0$	$z = +0$
$z = x + y$	$x = +0, y = +\infty$	$z = +\infty$
$z = x + y$	$x = +5.4, y = -5.1$	$z = +0.3$
$z = x + y$	$x = +\infty, y = -\infty$	$z = NaNs$
$p = x \times y$	$x = +0, y = -0$	$p = -0$
$p = x \times y$	$x = +0, y = +\infty$	$p = NaNs$
$p = x \times y$	$x = +5.4, y = -5.1$	$p = -27.54$
$p = x \times y$	$x = +\infty, y = -\infty$	$p = -\infty$

Os testes executados vieram comprovar a validade e o desempenho das arquiteturas e dos algoritmos desenvolvidos.

V.3 - Implementação em EPLDs

EPLDs (*Erasable Programable Logical Devices*) são excelentes dispositivos para desenvolvimento de *hardware*. Primeiramente porque são programáveis e apagáveis, o que facilita a implementação de modificações durante a fase de desenvolvimento do projeto. Além disto, eles são bastante versáteis, possuindo dispositivos com um bom número de *flip-flops* (até 256) e pinos de entrada e saída (até 164), porém uma lógica combinacional e uma matriz de interconexões limitada.

Mas o principal motivo que levou a escolha destes dispositivos na implementação do Elemento Processador foi a disponibilidade não só dos dispositivos em si como também das ferramentas de desenvolvimento e gravação dos mesmos. O Núcleo de Computação Eletrônica possui um conjunto de ferramentas capaz de trabalhar com a linha MAX5000 da Altera, denominado MAX+PLUS [Alte90a]. O *sofware* é composto por quatro editores (gráfico, simbólico, de textos e de formas de onda), um compilador, um simulador e um programador.

O MAX+PLUS permite que a arquitetura seja descrita através de desenhos esquemáticos (criados com o auxílio do editor gráfico), arquivos de texto na linguagem AHDL (*Altera Hardware Description Language*) [Alte90b] ou uma combinação dos dois. De uma maneira geral a descrição dos Elementos Processadores através da linguagem AHDL é mais vantajosa, visto que ela é hierárquica e suporta, além de equações booleanas, tabelas verdade e máquinas de estado, se aproximando muito de uma linguagem de programação científica de alto nível, como por exemplo o Pascal.

Descrita a arquitetura, passa-se a etapa de compilação, onde a lógica é sintetizada e alocada no *chip*. Ultrapassada a etapa de compilação, pode-se simular o funcionamento da arquitetura, corrigindo e/ou otimizando a implementação.

Cabe ressaltar que, por limitações do otimizador, as descrições por equações booleanas, tabelas verdade ou máquinas de estado geram resultados diferentes para uma mesma arquitetura, tanto em termos de total de recursos da EPLD utilizados quanto em termos de tempo de atraso. Sendo assim, para cada módulo da arquitetura (somador, unidade lógica, etc) testou-se as diversas formas de descrição, a fim de encontrar a de melhor desempenho. O resultado obtido, ao contrário do que se esperava, não permitiu eleger um determinado tipo de descrição como o de melhor resultado genérico, independente do módulo sintetizado. Cada módulo adaptou-se melhor a um determinado tipo de descrição. Este resultado tornou o trabalho de otimização da implementação uma tarefa muito mais complexa do que o esperado,

considerando-se o uso de um sistema integrado de desenvolvimento e implementação de circuitos lógicos como o MAX+PLUS.

A última etapa da implementação é a gravação da EPLD, executada através do programador, com o auxílio do MAX+PLUS.

A linha de dispositivos MAX5000 [Alte91] possui dispositivos de 16 até 192 macrocélulas. Cada macrocélula é composta por um *flip-flop*, e uma lógica combinacional, como mostra a Figura V.4. Para aumentar a capacidade das macrocélulas no que tange à lógica combinacional sem inserir atrasos de realimentação são usados os expansores, que são produtos lógicos invertidos (portas *NAND*).

Foi usada a maior EPLD disponível, que é a EPM5128. Este dispositivo se caracteriza por possuir 128 macrocélulas, 256 expansores, 8 entradas e 52 entradas/saídas.

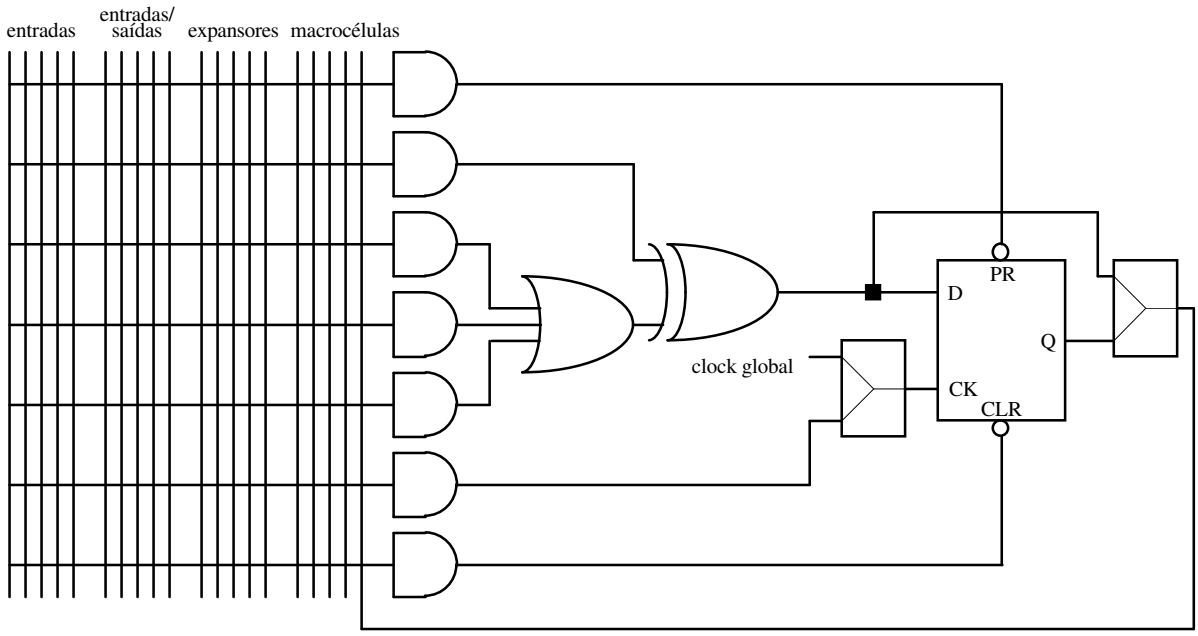

Figura V.4 - Arquitetura de uma macrocélula da linha MAX5000

Para implementar o EP na EPLD, tentou-se minimizar o número de realimentações de uma macrocélula para outra, a fim de minimizar o ciclo de relógio do processador. O ideal seria que cada macrocélula correspondesse a um registrador ou a uma entrada/saída. Desta forma seriam necessárias apenas 47 macrocélulas para implementar um EP, o que permitiria o encapsulamento de dois EPs em uma única EPLD. Porém, a prática mostrou que isto não é possível. A lógica combinacional que antecede alguns registradores é mais complexa do que uma única macrocélula pode sintetizar, mesmo com o auxílio dos expansores. Sendo assim, foi necessário utilizar macrocélulas para sintetizar lógicas combinacionais intermediárias.

Só foi conseguido implementar um único EP em cada EPM5128. Além das limitações inerentes ao dispositivo, outro fator que contribuiu para esta fraca integração foram as limitações do algoritmo de otimização e alocação do próprio MAX+PLUS, que muitas vezes optou por soluções claramente piores do que as "manuais". Quanto ao dispositivo, o ponto mais fraco encontrado em relação à implementação do EP foi a utilização de uma soma de apenas três produtos na geração do sinal de entrada *D* do *flip-flop*, o que restringiu bastante a lógica combinacional.

Um resultado bem melhor é esperado com a utilização de EPLDs da linha MAX7000 [Alte93], porém o projeto ainda não dispõe destes dispositivos nem do MAX+PLUS II, que é a ferramenta capaz de programá-los. O MAX+PLUS II conta com um otimizador mais eficiente. Porém, a grande expectativa em relação a linha MAX7000 se deve à sua arquitetura, que permite uma soma de mais produtos na geração do sinal de entrada *D* do *flip-flop*, podendo usar para tal os produtos anteriormente exclusivos dos sinais de *CK*, *PR* e *CLR* do *flip-flop*, além dos produtos de macrocélulas vizinhas. Neste último caso, ao se "roubar" um produto de uma macrocélula vizinha, perde-se esta macrocélula, porém não se acrescenta o atraso de uma realimentação, ou seja, não se aumenta o ciclo de relógio do processador.