# IMPLEMENTAÇÃO EM FPGA DE UM MULTIPLICADOR DE PONTO FLUTUANTE COM PIPELINE PROFUNDO

Sandro Silva, Alex Panato, Flávio Wagner, Ricardo Reis, Sergio Bampi
*Universidade Federal do Rio Grande do Sul - Instituto de Informática*
*Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre, RS, Brazil*
*e-mail: < svsilva, panato, flavio, reis, bampi>@inf.ufrgs.br*

**SUMMARY**

The utilization of floating-point multipliers is crucial in several digital designs. However, a critical point in this approach is the speed of these multipliers. The IEEE 754 standard leads to a high computation time to perform a single multiplication.

Pipelines are largely used to increase digital circuits performance since they implement parallelism in sequential operations. In FPGAs, the leader companies – Xilinx and Altera – present basic structures that are very useful for a deep pipeline approach. In their granules, called Logic Elements (LEs), these devices show two output options, the first one combinational and the second one registered. So, the use of pipelines at LE level becomes possible.

This work explores the development of a floating-point multiplier for FPGAs using deep pipeline, which follows the IEEE 754 standard. The floating-point multiplier is described using VHDL without dedicated components, generating, in theory, a universal code. However, due to the characteristics of the deep pipeline methodology, this portability is guaranteed only for Xilinx and Altera.

The design achieves high performance in data processing due to the use of a deep pipeline with reduced stage width. Its operation allows very high frequencies, very close to the structural limit of an FPGA device. However, this deepness leads to a high latency. This problem is minimized by modifications to the architecture of the integer multiplier. A frequency of 235MHz was achieved, which allows 235MFLOPS of operation when the pipeline is full.

**RESUMO**

A necessidade do emprego de multiplicadores de ponto-flutuante é um ponto crítico em diversos projetos digitais. Contudo, tem-se como limitação a velocidade de computação deste tipo de multiplicador. Devido à natureza do padrão IEEE 754 é necessário um longo tempo de computação para operar uma única multiplicação.

*Pipelines* são largamente utilizados para aumentar a performance de circuitos digitais por implementar paralelismo em operações seqüenciais. Em FPGAs, as empresas líderes de mercado – Xilinx e Altera – apresentam estruturas básicas propícias para o emprego de *pipelines* de alta profundidade. Nos seus grânulos, chamados de *Logic Elements* (LEs), estes dispositivos apresentam duas opções de saída, a primeira combinacional e a segunda registrada. Assim, o emprego de *pipelines* em nível de LEs torna-se possível.

Neste trabalho, explora-se o desenvolvimento de um multiplicador de ponto-flutuante padrão IEEE 754 para FPGAs, utilizando *pipeline* profundo. O multiplicador de ponto-flutuante é descrito em VHDL sem componentes dedicados, o que garante uma portabilidade, em tese, universal. Contudo, dadas as características da metodologia de pipeline profundo utilizada, esta portabilidade é garantida apenas para FPGAs da Altera e Xilinx.

O projeto busca alto desempenho em processamento de dados através do emprego de *pipeline* com tamanho do estágio reduzido. Sua operação permite o uso de freqüências altíssimas, muito próximas ao limite estrutural de funcionamento do FPGA. Contudo, tal profundidade provoca uma elevação na latência do dispositivo. Este problema é tratado por uma alteração arquitetural do multiplicador inteiro de modo a diminuir sua latência. A freqüência de operação obtida é de 235MHz, o que garante 235 MFLOPS de operação com o *pipeline* totalmente preenchido.

.

# IMPLEMENTAÇÃO EM FPGA DE UM MULTIPLICADOR DE PONTO FLUTUANTE COM PIPELINE PROFUNDO

Sandro Silva, Alex Panato, Flávio Wagner, Ricardo Reis, Sergio Bampi
*Universidade Federal do Rio Grande do Sul - Instituto de Informática*
*Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre, RS, Brazil*
*e-mail: <svsilva,panato,flavio,reis,bampi>@inf.ufrgs.br*

**RESUMO**

Este trabalho apresenta um multiplicador de ponto-flutuante padrão IEEE 754 de precisão simples para FPGAs. O projeto busca alto desempenho em processamento de dados através do emprego de *pipeline* com tamanho de estágio mínimo, isto é, um *pipeline* profundo. Sua operação permite o uso de freqüências altíssimas, muito próximas ao limite estrutural de funcionamento do FPGA. Contudo, tal profundidade provoca uma elevação na latência do dispositivo. Este problema é tratado por uma alteração arquitetural do multiplicador inteiro de modo a diminuir sua latência. O projeto é descrito em código VHDL portável para diversas arquiteturas de FPGA, como Altera e Xilinx. A freqüência de operação obtida é de 235MHz, o que garante 235 MFLOPS de operação com o pipeline totalmente preenchido.

## 1. INTRODUÇÃO

A necessidade do emprego de multiplicadores de ponto-flutuante é um ponto crítico em diversos projetos digitais. Contudo, ela esbarra na velocidade de computação destes multiplicadores. Devido à natureza do padrão IEEE 754 [1] é necessário um longo tempo de computação para operar uma única multiplicação.

*Pipelines* são largamente utilizados para aumentar a performance de circuitos digitais por implementar paralelismo em operações seqüenciais. Genericamente, o aumento do número de estágios inseridos torna menor o atraso de cada estágio. Isto deixa de ser verdade quando os elementos de armazenamento passam a acrescentar atrasos superiores à funcionalidade do estágio.

Em FPGAs, as empresas líderes de mercado – Xilinx e Altera – apresentam estruturas básicas propícias para o emprego de *pipelines* de alta profundidade. Nos seus grânulos, chamados de *Logic Elements* (LEs), estes dispositivos apresentam duas opções de saída, a primeira combinacional e a segunda registrada (Figura 1). Assim, o emprego de *pipelines* em nível de LE é possível dado que:

1. O atraso de propagação do LE já prevê o uso do registrador;
2. O atraso das interconexões é preponderante frente ao dos LEs, permitindo que um único estágio não precise computar diversas conexões.
3. O projeto pode ser feito em VHDL parametrizável, com total flexibilidade e migração de subcircuitos.

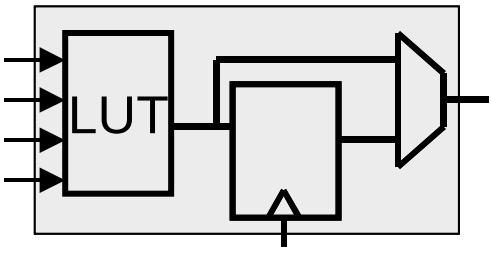


**Figura 1:** Estrutura básica de um LE

Em [2] é apresentada uma metodologia para desenvolvimento de circuitos com *pipeline* profundo, utilizando um único LE como tamanho de estágio de *pipeline*. A idéia é descrever cada blocos básicos com um máximo de quatro entradas e uma saída sempre registrada, de modo a ocupar um único LE. A descrição deve ser estrutural, usando-se sempre estes blocos como ponto de partida. A Altera apresenta uma sugestão de implementação de multiplicadores com *pipeline* profundo [5], porém não apresenta nenhum resultado referente à implementação.

Arquiteturas implementadas com a metodologia de *pipeline* profundo normalmente operam em freqüências altíssimas, muito próximas ao limite estrutural de funcionamento do FPGA. Contudo, tal profundidade provoca uma elevação na latência do circuito implementado.

Neste trabalho, explora-se o desenvolvimento de um multiplicador de ponto-flutuante para o padrão IEEE 754 utilizando esta técnica, porém com alterações arquiteturais que proporcionam uma latência menor que a do multiplicador proposto em [2].

O multiplicador é descrito em VHDL sem componentes dedicados, o que garante uma portabilidade, em tese, universal. Contudo, dadas as características da metodologia de *pipeline* profundo utilizada, esta

portabilidade é garantida apenas para FPGAs da Altera e Xilinx.

Embora não tenha ainda sido empregada para multiplicadores de ponto-flutuante, esta técnica já foi empregada em multiplicadores inteiros para ASICs. Em [3] apresenta-se um multiplicador inteiro de 8 bits, *full-custom*, usando estágios de *pipeline* de um meio-somador. Em [4] é usada uma estratégia semelhante para o desenvolvimento de um multiplicador de complemento de dois. Estes trabalhos apresentam uma elevadíssima freqüência de operação. Estas implementações, contudo, não foram estendidas para FPGAs.

Este artigo está organizado da seguinte forma. Na Seção 2 é feita uma breve introdução ao padrão de ponto-flutuante adotado. Na Seção 3 a arquitetura do multiplicador de ponto flutuante e seus blocos básicos é mostrada. Na Seção 4, a arquitetura proposta para diminuição da latência é introduzida. Resultados são apresentados na Seção 5 e as conclusões são discutidas na Seção 6.

## 2. PADRÃO IEEE 754

O padrão IEEE 754 [1] determina a formatação dos números de ponto-flutuante. Os dados possuem duas informações fundamentais: mantissa e expoente. A mantissa representa os dígitos significativos do número e neste padrão, ela é sempre um valor maior do que 1. O expoente indica a potência à qual a base deve ser elevada.

Os 32 bits da precisão simples são divididos da seguinte forma (Figura 2):

- Mantissa: Corresponde aos 23 bits menos significativos. A multiplicação deste número é feita por um multiplicador inteiro de 24 bits, acrescentando-se 1 no bit mais significativo.
- Expoente: Valor dos bits 23 a 31, com um *bias* de valor 127, isto é, valores inferiores a 127 indicam expoente negativo. Na multiplicação os expoentes são somados e posteriormente retira-se o *bias* de 127 referente a um dos expoentes.
- Sinal: Corresponde ao bit mais significativo. Representa o sinal da mantissa. O sinal final é obtido através de uma operação xor.

Para calcular o número de modo adequado, este deve passar por algumas etapas. São elas: multiplicação da mantissa, soma dos expoentes, geração do sinal, normalização e arredondamento, além da presença de um tratamento para exceções. A Figura 3 mostra estas etapas, com a exploração de paralelismo entre elas.

A única exceção tratada neste trabalho é a ocorrência de multiplicação por zero, pois o valor infinito nunca ocorre na multiplicação.

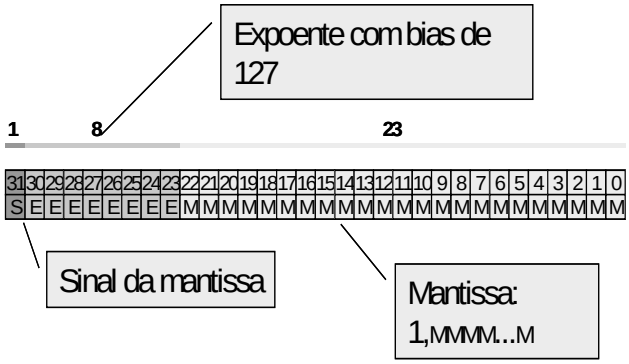


**Figura 2:** Organização dos bits no padrão IEEE 754

O arredondamento pode ser feito de diversas formas. Neste trabalho optou-se pelo truncamento como alternativa inicial. Um novo módulo para tratamento diferenciado pode ser adicionado sem alterações significativas nos resultados.

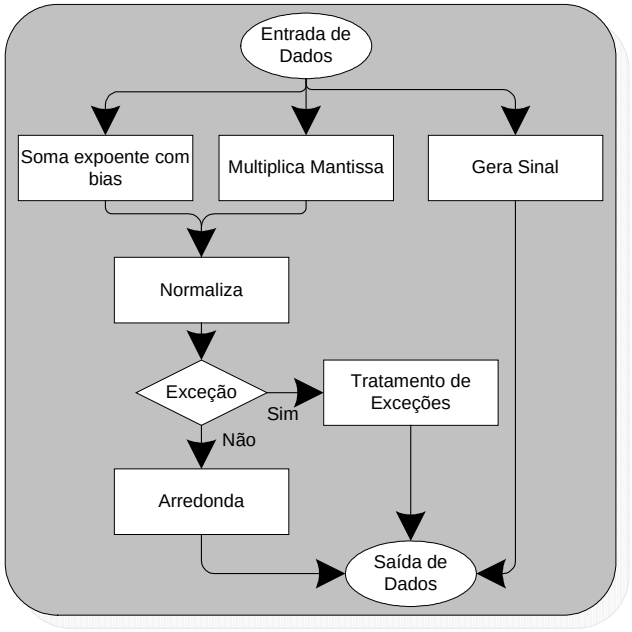


**Figura 3:** Operações realizadas na multiplicação.

## 3. ARQUITETURA DO MULTIPLICADOR DE PONTO-FLUTUANTE

Nesta seção apresenta-se a implementação do multiplicador de ponto-flutuante de precisão simples, baseado no padrão IEEE 754 [1]. A arquitetura desenvolvida para tanto é mostrada na Figura 4.

O sinal dos operandos é determinado por uma porta lógica xor (topo, à direita). O resultado é propagado por

38 registradores de modo a haver sincronismo com os demais sinais. Na coluna logo à esquerda pode-se observar o identificador de zero, que identifica a existência de zero em algum dos operandos (bloco chamado de '0?'). Seu resultado é propagado por 19 registradores. O sinal é responsável por escrever zero na saída.

O expoente é calculado pelos blocos 'adder' e 'sub', que tratam-se de blocos somador de 8 bits e subtrator de 9 bits, respectivamente, usando a técnica de *pipeline* profundo. O somador soma os expoentes dos operandos, enquanto o subtrator retira o *bias* excedente introduzido na operação anterior. Ele indica também a ocorrência de um *overflow* ou *underflow*. Como o bloco 'adder' soma dois valores de 8 bits, para representar seu resultado são necessários 9 bits. Logo todos os registradores existentes após o bloco 'adder' necessitam possuir 9 bits. O bloco 'sub', que retira o *bias* do expoente final, precisa ter 9 bits em cada entrada, gerando um valor de 8 bits.

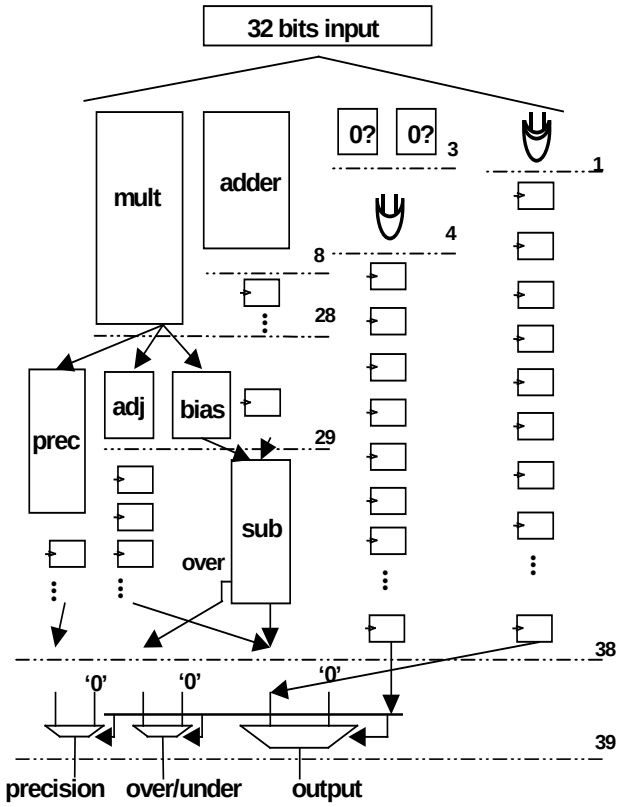


**Figura 4:** Arquitetura do multiplicador de ponto-flutuante

A multiplicação utiliza um multiplicador de inteiros de 24 bits, usando a técnica de *pipeline* profundo. Este bloco é o responsável pela alta latência do circuito e é mostrado em maiores detalhes na seção seguinte, onde se tratará também as modificações que visam diminuir a latência.

Uma vez que o padrão define como 1 o bit mais significativo de ambos os operandos, o resultado terá ao menos um bit 1 nos dois bits mais significativos (1X ou 01). Assim, caso o bit mais significativo seja 1, é necessário realizar uma normalização da mantissa e do expoente. A normalização da mantissa é feita simplesmente através de um multiplexador que desloca as entradas de modo adequado (bloco 'adj'). A normalização do expoente é feita através do ajuste do *bias* que será subtraído. Se os operando necessitarem de normalização, o valor a ser subtraído será 128; caso contrário, será 127 (bloco 'bias').

Para indicar a precisão, uma estrutura de três ciclos de operação é inserida após a multiplicação. Ela compara se os bits da mantissa rejeitados pelo arredondamento são iguais a zero. Neste caso, o sinal de precisão é acionado. Por fim, um conjunto de multiplexadores é inserido para escrever o valor zero nas saídas caso ocorra a exceção zero.

## 4. ARQUITETURA DO MULTIPLICADOR INTEIRO

A arquitetura do multiplicador inteiro apresenta três macro-estruturas. Na primeira ocorre a geração de todos os produtos parciais organizados na forma de um vetor de n posições. Na segunda ocorre a transformação deste vetor de n valores para um vetor de 3 valores (*soma*, *carry1* e *carry2*). Na terceira estrutura ocorre a propagação do ajuste do *carry*. A Figura 5 mostra a estrutura geral do multiplicador *array* modificado.

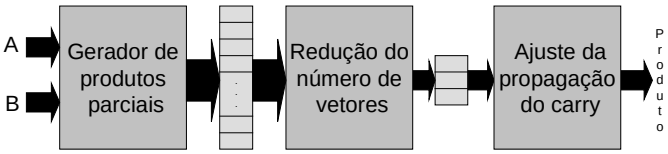


**Figura 5:** Multiplicador *Array* Modificado.

A primeira estrutura, chamada *mrow1*, gera um vetor de produtos parciais, a partir das duas entradas de dados A e B, que possuem n bits de tamanho de palavra. Esta estrutura utiliza n/2 blocos e é implementada utilizando exclusivamente o bloco básico *mcell2a*, que produz dois valores de saída, o primeiro correspondendo à soma e o segundo ao *carry*. Estes valores são armazenados em um vetor de produtos parciais (Figura 6).

Os valores do vetor de produtos parciais não possuem um ajuste de *bias*. Assim, deve ser feito um ajuste de *bias* de todos os valores deste vetor. A regra para este ajuste é a seguinte:

- deslocar para a esquerda os dados das posições pares em k posições, onde k é a posição no vetor;
- deslocar para a esquerda os dados das posições ímpares em k+1 posições.

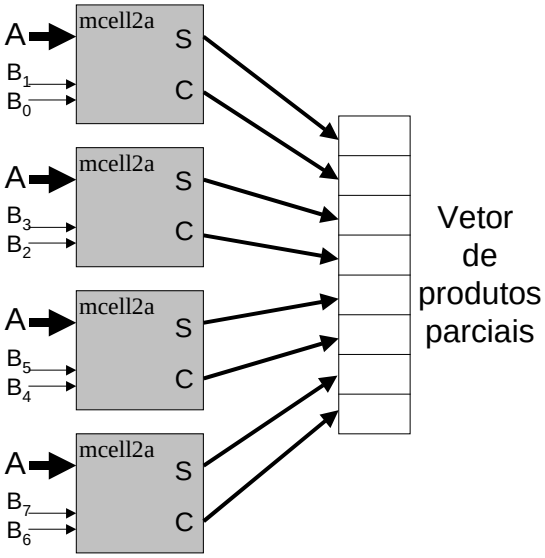


**Figura 6:** Geração do vetor de produtos parciais para um multiplicador de 8 bits.

A segunda estrutura, chamada *mrow2*, que realiza a redução do vetor de n valores para o vetor de 3 valores (*soma*, *carry1* e *carry2*), é implementada por m estágios somadores de 4 entradas, onde m é dependente do número de bits utilizados na implementação. Em cada estágio ocorre a soma em paralelo de cada quatro posições do vetor de entrada.

**Tabela 1:** Número de estágios da estrutura *m_row2*.

| ***Tamanho da palavra*** | ***Número de estágios (m)*** |
|---|---|
| 4 | 1 |
| 8 | 4 |
| 16 | 6 |
| 24 | 8 |
| 32 | 9 |
| 54 | 11 |
| 64 | 11 |

Conforme a Tabela 1, para reduzir um vetor cujo tamanho da palavra tenha 8 bits, por exemplo, são necessários 4 estágios (Figura 7). No primeiro estágio existem 2 blocos somadores, para reduzir o vetor de entrada (vetor de produtos parciais ajustado), de 8 valores para 6 valores. O segundo estágio reduz o vetor de 6 valores (vindo do primeiro estágio) para 5 valores. O terceiro estágio reduz o vetor de 5 valores para 4 valores. Por fim o quarto estágio reduz o vetor de 4 valores para 3 valores. Este 3 valores são a entrada para a última macroestrutura.

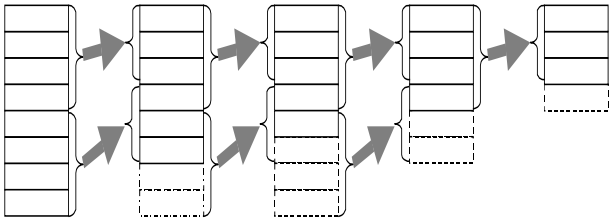

**Figura 7:** Redução realizada na segunda estrutura para tamanho de palavra de 8 bits.

Todos os blocos somadores desta estrutura são constituídos por blocos básicos *soma_4in*, que recebem 4 valores de entrada e geram 3 valores de saída.

Como regra podemos definir que nesta estrutura:

- 4 entradas válidas geram 3 valores;
- 3 entradas válidas geram 2 valores;
- 2 entradas válidas geram 2 valores; e
- 1 entrada válida gera 1 valor.

Na terceira estrutura, chamada *mrow_last*, ocorre a propagação do ajuste do *carry*, necessitando de uma latência proporcional ao tamanho da palavra. Nesta proposta, foi utilizado o bloco básico *soma_2x2*. As entradas deste bloco são dois bits de duas palavras diferentes (totalizando 4 bits de entrada), produzindo dois bits de soma e um bit de *carry*, de acordo com a Figura 8.

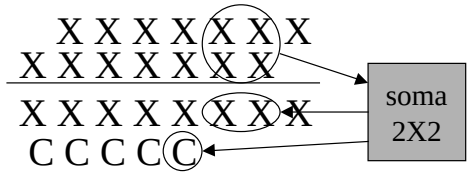


**Figura 8:** Representação funcional do bloco soma_2x2.

O número de estágios de *pipeline* necessários nesta estrutura é apresentado na Tabela 2.

Cada bloco desta estrutura segue o seguinte modelo:

- (n+k) blocos *mcell3*;
- 1 bloco *soma_2x2*;
- (tam_palavra - n+k+2) blocos *soma_4in*;

**Tabela 2:** Número de estágios da estrutura *mrow_last*.

| ***Tamanho da palavra*** | ***Número de estágios (m)*** |
|---|---|
| 4 | 2 |
| 8 | 5 |
| 16 | 12 |
| 24 | 19 |
| 32 | 27 |
| 54 | 48 |
| 64 | 58 |

## 5. RESULTADOS

Nesta seção são apresentados os resultados de área, latência e performance obtidos na implementação do multiplicador de ponto-flutuante, que segue o padrão IEEE 754 apresentado na Seção 2, onde utilizamos um multiplicador inteiro de 24 bits, um somador de 8 bits e um subtrator de 9 bits. Os resultados são comparados com o trabalho anterior apresentado em [2] e com o multiplicador de ponto-flutuante da Altera disponível na biblioteca Mega Wizard do software Quartus II, versão 2.2 da Altera [6]. É importante lembrar que a arquitetura da Altera não permite comparações estruturais, uma vez que ela não é aberta ao usuário.

A implementação do multiplicador em ponto flutuante utilizando *pipeline* profundo proporciona um grande incremento na velocidade de processamento, como pode ser visto na Tabela 3. Apesar de apresentarem grande latência, as duas versões de multiplicador de ponto flutuante que utilizam *pipeline* profundo são aproximadamente 3,73 vezes mais rápidas e com área apenas 2,7 vezes maior do que a versão utilizando as arquiteturas pré-definidas na biblioteca Mega Wizard da Altera.

**Tabela 3:** Resultados de área e performance.

| ***multiplicador*** | ***Área (LEs)*** | ***Performance (MHz)*** |
|---|---|---|
| Versão anterior [2] | 4434 | 235 |
| Versão atual | 4361 | 235 |
| Versão Altera | 1655 | 62,9 |

Conforme apresentado na Tabela 3, as diferenças de área e desempenho entre a versão anterior e a versão atual do multiplicador com *pipeline* profundo não são significativas. Porém, na versão introduzida neste trabalho, que utiliza a técnica descrita na Seção 4, foi possível reduzir sensivelmente o número de estágios de pipeline de 58 para 39, proporcionando uma redução na latência de aproximadamente 33%. Os resultados desta redução sobre o multiplicador de ponto-flutuante são resumidos na Tabela 4.

**Tabela 4:** Latência do multiplicador de ponto flutuante.

| ***multiplicador*** | ***Pipeline*** |
|---|---|
| Versão anterior [2] | 58 |
| Versão atual | 39 |
| Versão Altera | 5 |

A comparação entre o multiplicador de ponto-flutuante com *pipeline* profundo introduzido neste trabalho e o multiplicador disponível na biblioteca Mega Wizard da Altera mostra que existe um interssante espaço de projeto a ser explorado na seleção do multiplicador mais adequado a uma determinada aplicação. Se área for o fator mais crítico, a solução da Altera é obviamente preferível. No entanto, se o desempenho é crítico, a solução proposta neste trabalho traz uma grande vantagem, desde que os custos associados de área e latência sejam aceitáveis. O custo de latência torna-se tanto menos significativo quanto maior for o fluxo contínuo de dados na entrada do multiplicador, o que permite que o *pipeline* opere totalmente preenchido por intervalos maiores de tempo.

## 6. CONCLUSÕES

Este trabalho investigou a implementação de um multiplicador de *pipeline* profundo em VHDL. Os resultados indicam que este tipo de multiplicador apresenta uma grande performance quando operando com seu *pipeline* totalmente preenchido.

O objetivo de diminuição do número de estágios de *pipeline*, que implica tanto em uma diminuição da latência quanto em uma maior facilidade de preenchimento completo do *pipeline*, foi plenamente atingido. Além de obter-se uma redução no número de estágios do *pipeline*, o que é diretamente proporcional à latência do circuito, a nova arquitetura obteve uma ligeira redução de área total. As taxas de desempenho obtidas são bastante significativas, atingindo-se 235 MFLOPS.

Neste trabalho, ao contrário da versão anterior do multiplicador introduzida em [2], não foi utilizada nenhuma técnica de otimização de posicionamento, procedimento que poderia melhorar ainda mais o desempenho final do circuito.

## REFERÊNCIAS


[1] IEEE Standard for Binary Floating-Point Arithmetic in ANSI/IEEE Std 754-1985. New York, USA, 1985.

[2] PANATO, Alex, SILVA, Sandro, WAGNER, Flávio, JOHANN, Marcelo, REIS, Ricardo & BAMPI, Sérgio. Design of Very Deep Pipelined Multipliers for FPGAs. Designer's Forum of DATE 2004.

[3] NOLL, Tobias G., SCHMITT-LANDSIEDEL, Doris, KLAR, Heinrich & ENDERS, Gerhard. A Pipelined 330-MHz Multiplier. IEEE J. Solid-State Circuits, vol sc-21, no 3, June 1986.

[4] SOMASEKHAR, Dinesh & VISVANATHAN, V. A 230-MHz Half-Bit Level Pipelined Multiplier Using True Single-Phase Clocking. IEEE Trans on VLSI Systems, vol 1, no 4, December 1993.

[5] ALTERA. Pipelined Multipliers in FLEX 8000 Devices. Application Brief 134. May 1994, version 1.

[6] ALTERA. NCO Compiler. MegaCore Function User Guide. Version 2.0.2. November 2002.