

PROCESSADORES RECONFIGURÁVEIS: ESTADO DA ARTE

Leandro H. Möller, Fernando G. Moraes, Ney L. V. Calazans

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Faculdade de Informática (FACIN)
Av. Ipiranga, 6681 - Prédio 30 / Bloco 4 - CEP 90619-900 - Porto Alegre - RS - BRASIL
Fone: +55 51 3320-3611 - Fax: +55 51 3320-3621

{moller, moraes, calazans}@inf.pucrs.br

ABSTRACT

Processadores reconfiguráveis estendem a capacidade de um processador otimizando seu conjunto de instruções para uma tarefa específica. Com isto é obtido um aumento de desempenho e flexibilidade, ao mesmo tempo que a área do núcleo do processador permanece inalterada. Este artigo apresenta o estado da arte em processadores reconfiguráveis. Uma proposta de processador reconfigurável fracamente acoplado é apresentada. O objetivo deste processador é permitir a reconfiguração parcial e dinâmica utilizando dispositivos FPGAs atuais. Uma implementação prova de conceito é apresentada, mostrando o ganho de desempenho obtido.

1. INTRODUÇÃO

Aplicações com computações intensivas passam em média 90% do tempo em apenas 10% do código [1]. Isto permite otimizar o processador utilizado para executar os trechos de código responsáveis por este maior consumo de tempo de processamento. A implementação de um hardware específico é uma alternativa possível para alcançar um ganho de desempenho significativo.

Processadores com um conjunto de instruções específico para a aplicação (*Application-Specific Instruction-Set Processor - ASIP*) provêm flexibilidade e desempenho a um custo de área adicional para cada nova função implementada em hardware. Processadores com um conjunto de instruções reconfiguráveis (*Reconfigurable Instruction-Set Processor - RISP*) não apresentam um custo em hardware para cada nova instrução em hardware, mas sim um custo pré-definido de área onde a lógica reconfigurável pode ser programada em tempo de execução.

Exemplos de diferentes processadores reconfiguráveis são estudados na Seção 2. A Seção 3 classifica os processadores reconfiguráveis apresentados na Seção 2. A Seção 4 apresenta um estudo de caso de um processador reconfigurável denominado R8R. A Seção 5 apresenta as conclusões deste trabalho.

2. ESTADO DA ARTE EM PROCESSADORES RECONFIGURÁVEIS

2.1 PRISM-I

O PRISM-I (*Processor Reconfiguration through Instruction-Set Metamorphosis*) [2] teve sua primeira publicação por Peter Athanas et al. da Universidade do Estado da Virginia. Ele foi desenvolvido como prova de conceito para mostrar a viabilidade de incorporar um conjunto de instruções configurável a um processador de propósito geral. O PRISM-I consiste na interconexão de um nodo de processamento Armstrong em uma plataforma com hardware reconfigurável por meio de um barramento, conforme ilustrado na Figura 1.

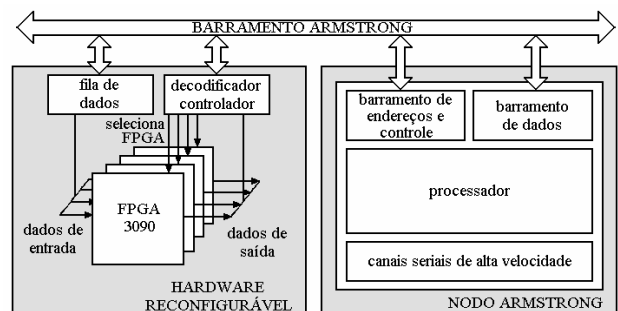


Figura 1 – Arquitetura do PRISM-I.

O nodo de processamento Armstrong é baseado em um processador M68010 da Motorola, que roda a uma frequência de 10MHz. A plataforma com hardware reconfigurável possui 4 FPGAs 3090 da Xilinx [3], cada um com seis mil portas lógicas equivalentes e 144 pinos de entrada e saída. A interconexão das duas plataformas foi feita pela interface de coprocessador de 16 bits do M68010.

Para avaliar o sistema, o processador fez chamadas de funções implementadas em hardware reconfigurável. As operações testadas consumiram de 48 a 72 ciclos de relógio para completar a operação e entregar a resposta de volta ao processador.

Os testes executados, embora tenham sido iniciais e prejudicados pelo baixo desempenho do meio de comunicação, mostraram que é possível obter ganhos com a implementação de instruções em hardware reconfigurável. Dentre as diversas operações executadas, a que obteve melhor ganho foi a função de logaritmo na base 2, executando 54 vezes mais rápido que em software.

2.2 PRISM-II

O PRISM-II [4] foi a versão seguinte ao PRISM-I. Esta versão teve como principais objetivos estender o número de instruções suportadas pelo compilador e reduzir o tempo de comunicação entre o processador e o hardware reconfigurável. Enquanto o PRISM-I leva em média 100ns para devolver a resposta ao processador, o tempo gasto na nova versão é de 30ns. Outra modificação feita foi no tamanho da palavra de dados, que no PRISM-I era fixa em 32 bits, e que no PRISM-II é de 64 bits para entrada e 32 bits para saída. O PRISM-I também possuía a limitação que uma instrução tinha que caber em um único FPGA. A arquitetura do PRISM-II permite que uma única operação seja dividida em até três FPGAs.

Para prover um melhor balanceamento entre hardware e software o processador M68010 foi substituído pelo AMD Am29050, que roda a 33MHz e possui um desempenho aproximado de 28 MIPS. Os FPGAs 3090 foram substituídos pelos 4010 com aproximadamente 20 mil portas lógicas equivalentes e 160 pinos de entrada e saída. Todas essas modificações resultaram em um fator de aceleração de aproximadamente 86 vezes em relação a software para uma função de reversão de bits em uma palavra de 32 bits. Isso mostra que essa versão obteve um ganho significativo de desempenho em relação ao PRISM-I, pois quando a mesma função foi executada na primeira versão ela obteve um fator de aceleração de apenas 26 vezes em relação a software.

2.3 Nano Processor

O Nano Processor [5] foi proposto por Michael Wirthlin et al. da Universidade de Brigham Young. O seu principal objetivo é poder ser implementado em FPGAs da época, por isso ele dispõe de uma arquitetura simples, ocupando pouca área. Ele possui seis instruções nativas e permite um total de 32 instruções. O processador é organizado hierarquicamente em três níveis, conforme apresentado na Figura 2.

O nível mais interno do processador é o núcleo do Nano Processor, que possui as seis instruções nativas e não foi desenvolvido para ser modificado. Ele possui um registrador de instruções (*Instruction Register - IR*) de 5 bits, um registrador de página de endereços (*Page Address Register - PAR*) de 3 bits, um contador de programa (*Program Counter - PC*) de 8 bits, um registrador de endereços (*Address Register - AR*) de 8 bits e um acumulador de 8 bits com 1 bit de carry. Nesta arquitetura

apenas instruções aritméticas de adição e subtração foram implementadas. O núcleo do Nano Processor ocupa 40 CLBs da família 3000 de FPGAs da Xilinx [3].

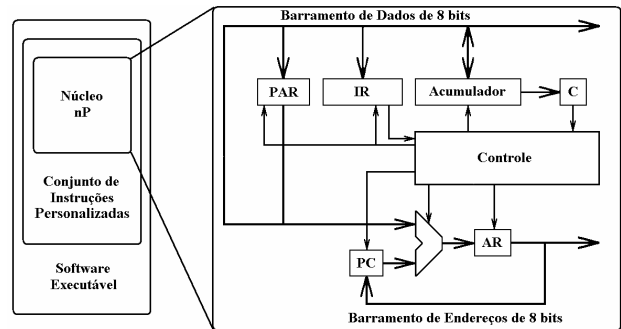


Figura 2 – Arquitetura do Nano Processor.

O nível intermediário do processador é chamado de Conjunto de Instruções Personalizadas. Nela são implementadas as instruções específicas ao problema a ser tratado pelo processador. Essas instruções podem ser selecionadas de uma biblioteca previamente desenvolvida ou criadas a partir de editores de esquemáticos ou ferramentas de síntese. O nível mais externo é o Software Executável. Nele é implementado a aplicação em código assembly onde são feitas as chamadas das instruções nativas e personalizadas.

O Nano Processor foi prototipado em uma plataforma da National Technologies. A plataforma inclui dois FPGAs 3090 da Xilinx [3], duas memórias SRAMs 32k x 8, uma memória DRAM de 1 Mb, um codec stereo de 16 bits e uma interface com o PC. O resultado obtido com esta plataforma foi 240 vezes mais rápido que o obtido com um processador 486 executando a 33 MHz para a aplicação *saturating mixer* [5].

2.4 DISC

O DISC (*Dynamic Instruction Set Computer*) [6] foi proposto por Michael Wirthlin et al. da Universidade de Brigham Young, um ano após a publicação do Nano Processor. Ele foi desenvolvido para suportar modificações no seu conjunto de instruções em tempo de execução. Para isso ele deve ser implementado em dispositivos que suportem reconfigurações dinâmicas parciais, como, por exemplo, os FPGAs fabricados pela Algotronix, Atmel, National Semiconductor e Xilinx. No DISC todas as instruções são implementadas como módulos de hardware e são substituídas quando não existe mais espaço disponível para a instrução seguinte. Essa abordagem possui a vantagem de não só utilizar uma área em hardware virtualmente maior que a existente fisicamente, mas também de permitir implementar funções complexas com melhor desempenho quando comparado a software. A arquitetura do DISC e exemplos de instruções são apresentados na Figura 3.

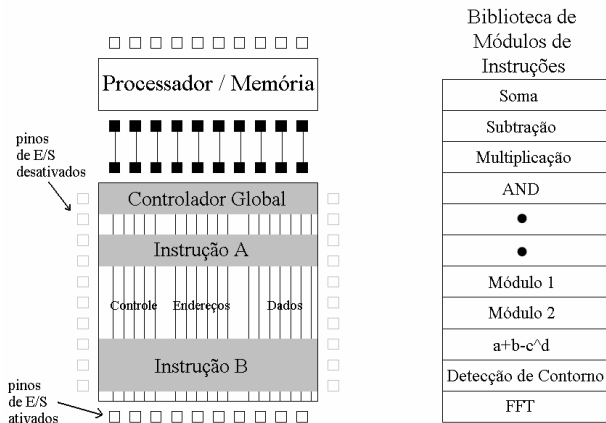


Figura 3 – Arquitetura do DISC e exemplo de biblioteca de módulos de instruções.

Antes de iniciar o DISC, um programa é carregado na memória, o módulo de controle global é carregado em hardware e o espaço de instruções dinâmicas é liberado. A seguir o DISC inicia a execução do programa. Quando for feita a chamada de uma instrução que não se encontra configurada, o processador é bloqueado e uma requisição de configuração de instrução é enviada ao computador hospedeiro pelos pinos de entrada e saída do sistema. O computador hospedeiro verifica o tamanho do módulo requisitado, seleciona onde o mesmo será posicionado, remove um ou mais módulos se não existir espaço suficiente no dispositivo e reposiciona o módulo para onde ele havia sido selecionado. A seguir, o computador hospedeiro envia a nova configuração para o sistema, o processador sai do estado de bloqueio e a instrução é executada. O principal módulo do DISC é o controlador global, que possui a tarefa de gerenciar os recursos de memória, os canais de comunicação e efetuar a reconfiguração dinâmica dos módulos de instrução.

Para avaliar o desempenho da arquitetura foi desenvolvido um algoritmo de filtro de imagem tanto com instruções de propósito geral quanto com instruções específicas. O ganho de desempenho obtido pela versão com instruções específicas foi de 23,5 vezes sobre a implementação com instruções de propósito geral.

2.5 OneChip

O OneChip [7] teve a sua primeira publicação por Ralph Wittig et al. da Universidade de Toronto. Nesse projeto é defendida a idéia que o processador deve estar fortemente acoplado ao hardware reconfigurável, minimizando os bloqueios do processador com uma comunicação rápida e eficiente entre eles.

Os recursos reconfiguráveis foram integrados no estágio de execução do processador, sendo chamados de Unidades Funcionais Programáveis (*Programmable Functional Unit - PFU*). As PFUs são adicionadas em paralelo às Unidades Funcionais Básicas (*Basic Functional Unit -*

BFU) do processador. As PFUs implementam funções específicas a uma aplicação, podendo ser circuitos sequenciais ou combinacionais. As PFUs também podem ser utilizadas como lógica de cola para que o processamento de uma determinada função venha do mundo externo. As BFUs são responsáveis pelas operações lógicas e aritméticas elementares de qualquer aplicação.

O OneChip foi prototipado em um Transmogrifier-1 [8], que possui quatro FPGAs 4010 da Xilinx [3], dois chips de interconexão Aptix AX1024 e quatro memórias SRAMs 32k x 9. O processador DLX [1] baseado no MIPS foi particionado em três FPGAs, deixando um FPGA inteiro e a lógica restante dos outros dois para PFUs do usuário. O OneChip obteve um fator de aceleração de quase 50 vezes na aplicação de uma DCT (*Discrete Cosine Transform*) em relação a execução em uma Workstation SGI Indy rodando a 150MHz com processador MIPS R4400.

2.6 Garp

O Garp [9] teve sua primeira publicação por John Hauser da Universidade da Califórnia em Berkeley e foi projetado com o principal objetivo de acelerar laços de aplicações de propósito geral. O projeto implementa um coprocessador conectado ao MIPS, que consiste em uma malha bidimensional de CLBs interconectadas por recursos de roteamento programáveis (Figura 4). A célula lógica básica da matriz reconfigurável é uma unidade de 2 bits com quatro entradas de 2 bits cada. Quatro canais de 32 bits de dados e um canal de 32 bits de endereços comunicam o processador a matriz reconfigurável.

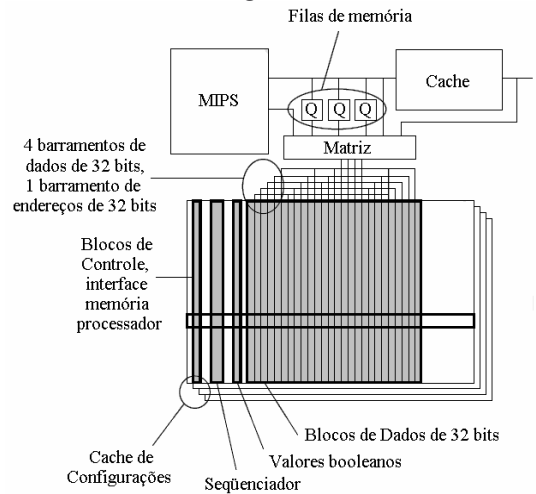


Figura 4 – Arquitetura do Garp.

A matriz reconfigurável possui pelo menos 24 blocos lógicos em uma coluna, dos quais 16 blocos a partir da primeira posição são conectados ao barramento compartilhado processador-memória. Na tentativa de minimizar os acessos ao barramento compartilhado, o projeto emprega a utilização de uma memória cache que pode armazenar

quatro configurações totais da matriz ou diversas configurações parciais. Uma reconfiguração total buscada da cache pode entrar em funcionamento em até cinco ciclos de relógio.

Para avaliar o desempenho do Garp foi utilizado um algoritmo de filtro de imagem que calcula a média dos pixels. A execução do Garp obteve um desempenho 43 vezes melhor que um Ultrasparc 170.

2.7 Chimaera

O Chimaera [10] foi proposto por Scott Hauck et al. da Universidade de Washington. Ele possui Unidades Funcionais Reconfiguráveis (*Reconfigurable Functional Unit* - RFU) pequenas o suficiente para serem integradas no próprio processador, buscando minimizar o gargalo de comunicação entre processador e lógica reconfigurável. Esta abordagem permite utilizar a lógica reconfigurável para acelerar as computações críticas das aplicações, enquanto aproveita a flexibilidade dos processadores para a criação de software. A Figura 5 apresenta a arquitetura do Chimaera.

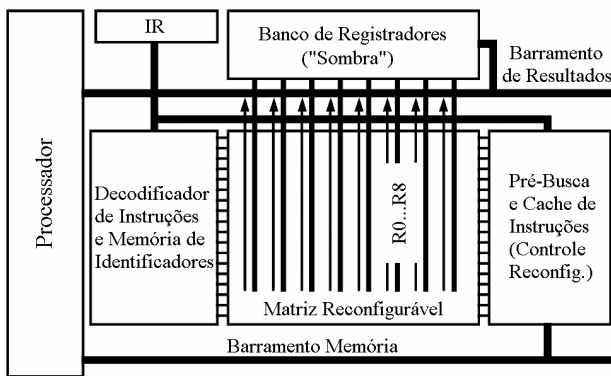


Figura 5 – Arquitetura do Chimaera.

O principal componente do sistema é a matriz reconfigurável, que possui uma arquitetura própria baseada no Triptych [11], na família FLEX 8000 da Altera [12] e no PRISC [13]. A matriz recebe entradas do banco de registradores chamado “sombra”, que possui cópia de um subconjunto de valores do banco de registradores do processador. Cada modificação feita no banco de registradores do processador é propagada pelo barramento, assim atualizando o “sombra”. As instruções que forem carregadas na matriz podem utilizar uma ou mais linhas de RFUs. Sempre que uma linha de RFUs for configurada, uma memória conectada à matriz armazena, em uma posição específica para a linha da RFU, o identificador da instrução que foi carregado na linha. No momento em que uma operação da matriz for executada, as linhas de saída da matriz que possuem o identificador da instrução são habilitadas a colocar o resultado no barramento.

Enquanto o processador executa instruções nativas, o módulo de pré-busca verifica a cada ciclo de relógio o

Registrador de Instruções (*Instruction Register* - IR), de forma “especulativa”. Isso permite iniciar a execução da operação na matriz antes mesmo da chamada ter sido feita pelo processador, assim reduzindo ou eliminando o bloqueio do mesmo à espera do resultado. Quando a chamada é de fato executada pelo processador, a matriz escreve o resultado no barramento de resultados caso a operação já tenha sido computada, caso contrário ela bloqueia o processador até a computação ser efetuada. Também pode ocorrer da operação não estar presente na matriz, assim causando um bloqueio no processador até que a operação seja buscada da cache de instruções ou da memória, parcialmente configurada na matriz e executada. Neste caso o tempo de reconfiguração pode ser significativo, logo o programador deve evitar ficar recarregando a RFU constantemente.

O Chimaera possui nove registradores que podem servir como origem de uma operação a ser executada nas RFUs. O programador deve saber quais destes registradores devem ser carregados antes de chamar a operação a ser executada na matriz, pois quando o processador chama uma operação implementada na RFU é passado como parâmetro somente o identificador da operação e o registrador de retorno.

Em cada linha da matriz reconfigurável existe uma célula por bit do processador. Por exemplo, um processador de 32 bits possui 32 células por linha, portanto uma célula N tem acesso ao enésimo bit dos registradores R0 a R8. Cada célula possui quatro entradas e quatro saídas que são utilizadas para comunicação com outras células.

As RFUs do Chimaera foram desenvolvidas para tratar com eficiência instruções típicas de um processador como AND, OR, XOR, XNOR, operações aritméticas simples, condições de salto e combinações destas. Essa arquitetura, embora simples, apresentou desempenhos até 160 vezes superiores do que a execução do processador sem lógica reconfigurável.

2.8 Morphosys

O Morphosys foi proposto por Hartej Singh et al. [14] da Universidade da Califórnia em Irvine e consiste em sete componentes principais (Figura 6): um processador baseado no MIPS (Tiny_RISC), uma malha 8x8 de células reconfiguráveis, uma memória de contextos, uma fila, uma cache de dados, um controlador de DMA e duas memórias principais (uma do processador e outra de contextos da malha reconfigurável).

Cada célula da malha reconfigurável pode ser individualmente reconfigurada. Ela é composta de uma ULA de 16 bits com módulo de multiplicação acoplado, uma unidade de deslocamento e dois multiplexadores que selecionam as entradas da célula. Cada célula também possui um registrador de saída e um banco de registradores.

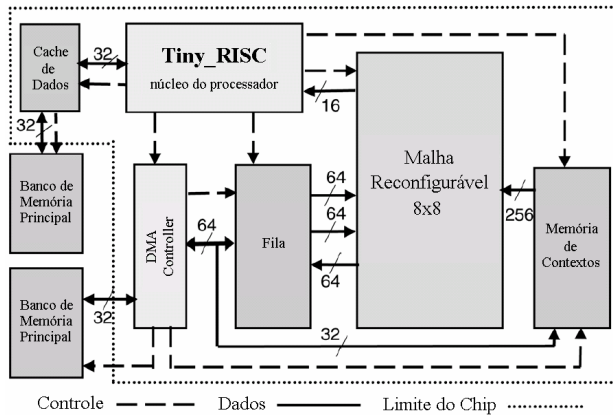


Figura 6 – Arquitetura do MorphoSys.

Enquanto o processador executa as tarefas sequenciais e de controle de transferência de dados entre o hardware reconfigurável e a memória, o hardware reconfigurável explora o paralelismo disponível do algoritmo da aplicação. O processador RISC possui duas classes de instruções específicas para controlar os componentes do MorphoSys. Uma dessas classes é responsável por carregar contextos da memória principal para a memória de contextos e a outra é responsável por iniciar transferências de dados entre a memória principal e a fila, essa classe é chamada de instruções de DMA.

A fila é organizada em dois conjuntos (0 e 1) de dois bancos (A e B) de 128 palavras de 16 bits. Um barramento de operandos de 128 bits conecta a fila com a matriz de células reconfiguráveis. Todas as células reconfiguráveis de uma mesma linha compartilham 16 bits do barramento de operandos. A transferência de dados entre a fila e a matriz de células reconfiguráveis pode executar apenas em modo entrelaçado, por exemplo, os dados podem ser acessados apenas alternando entre banco A e B. Enquanto este método aumenta a velocidade de acesso consecutivo à memória, ele restringe o acesso a dados aleatórios.

Uma característica importante da malha reconfigurável é a sua interconexão em três níveis. O primeiro nível acessa o vizinho mais próximo da mesma linha/coluna. O segundo nível provê conectividade completa com a linha/coluna do mesmo quadrante. O terceiro nível provê conectividade inter-quadrante, que consiste em barramentos que percorrem todas as linhas e colunas, cruzando as bordas dos quadrantes.

Um dos algoritmos utilizados para avaliar o desempenho do MorphoSys foi o de DCT, sendo executado em 37 ciclos de relógio. Este tempo foi mais de seis vezes menor que a execução do algoritmo em um computador com processador Pentium MMX com instruções otimizadas.

2.9 NAPA

O NAPA (*National Adaptive Processing Architecture*) [15] foi proposto por Charlé Rupp et al. da National Se-

miconductor Corporation. Ele é composto de uma rede denominada ToggleBus que interconecta um ou mais processadores, conforme apresentado na Figura 7. Um dos processadores da arquitetura é responsável por comunicar o sistema com o computador hospedeiro e gerar o vetor de controle (C) que define o tipo de fluxo de dados e parâmetros para cada ciclo do ToggleBus. Blocos de memória externa (M) se comunicam diretamente com os processadores da arquitetura por linhas de endereços (E) e dados (D). Cada processador possui acesso à rede de comunicação ToggleBus pelas portas P e Q. Além disso, cada processador possui 128 pinos de entrada e saída para atuar diretamente com sensores externos.

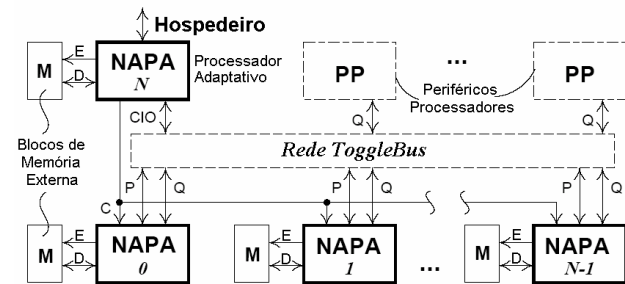


Figura 7 – Arquitetura do NAPA.

O NAPA integra lógica reconfigurável (*Adaptive Logic Processor - ALP*) com um processador escalar fixo (*Fixed Instruction Processor - FIP*) de maneira fortemente acoplada. Na primeira implementação do NAPA, as ALPs possuíam 6144 células reconfiguráveis (64 colunas por 96 linhas) e 1608 linhas de roteamento global. O processador utilizado foi o CompactRISC que é um pequeno processador RISC de 32 bits que roda a 50 MHz. Não foram apresentados dados de desempenho do sistema no artigo [15].

2.10 OneChip-98

O OneChip-98 [16] é uma versão posterior do OneChip, publicada por Jeffrey Jacob e Paul Chow ambos da Universidade de Toronto. A primeira versão do OneChip com processador, memória e lógica reconfigurável integrados obteve quase 50 vezes superior desempenho que uma estação de trabalho da época. A partir dele, detectou-se que o gargalo não era mais a interface entre o processador e a lógica reconfigurável, mas sim a interface destes com a memória. Por este motivo, o OneChip-98 (Figura 8) busca uma maior taxa de transferência de dados com a memória, permitindo o acesso concorrente à mesma pela lógica reconfigurável e pelo processador.

Para aumentar a taxa de transferência com a memória principal, caches foram adicionadas ao sistema, implementando uma hierarquia de memória semelhante à empregada em computadores pessoais. Conseqüentemente, um mesmo dado pode ser trazido tanto para a cache do processador quanto para a cache do FPGA, causando um

possível problema de coerência de caches. O tratamento deste problema é analisado no artigo [16].

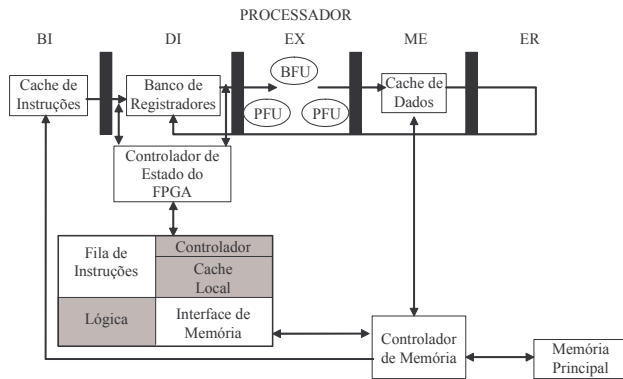


Figura 8 – Arquitetura do OneChip-98.

O OneChip-98 foi prototipado em um Transmogrifier-2 [8] com dois PLDs Altera Flex10K50. A aplicação de um filtro FIR 64-tap obteve uma aceleração de 32 vezes sobre um Ultrasparc 2 rodando a 300MHz. Os autores apontam como aplicações alvo problemas que acessam muito a memória.

2.11 PipeRench

O PipeRench [17] foi proposto por Seth Goldstein et al. da Universidade de Carnegie Mellon. A arquitetura é composta por uma rede de elementos processamento (*Processing Element* - PE) e de armazenamento interconectados, que atuam como um coprocessador. Esta arquitetura é organizada de forma a implementar um pipeline de configurações, sendo particularmente interessante para aplicações baseadas em fluxos uniformes (como processamento de imagens ou sons), ou qualquer computação simples e regular sobre grandes conjuntos de pequenos elementos de dados.

A Figura 9 apresenta uma visão abstrata de dois estágios do PipeRench e uma visão detalhada de um estágio. Cada PE contém uma ULA e um conjunto de registradores para comunicação entre estágios (registrador de passagem). Cada ULA contém LUTs e controle extra para propagação de carry, detecção de zero, e outros bits de controle. Lógica combinacional pode ser implementada usando um conjunto de N ULAs de B bits. Funções combinacionais complexas podem ser obtidas com o cascadeamento das linhas de propagação de carry, através da rede de interconexão.

A partir das redes de interconexão os PEs podem acessar operandos dos estágios anteriores que estão armazenados nos registradores de saída, bem como saídas de outros PEs do mesmo estágio. O barramento global é utilizado tanto pela aplicação para entrada e saída de dados quanto pelos PEs para alcançar os seus destinos, porque os estágios do pipeline de uma dada aplicação

podem estar fisicamente em qualquer um dos estágios da arquitetura.

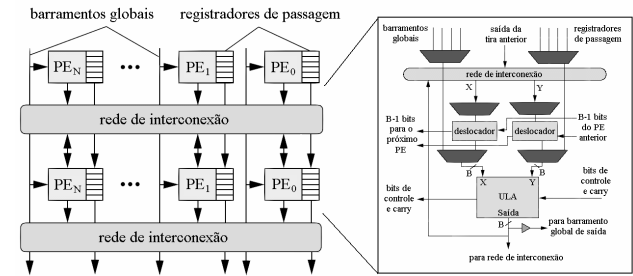


Figura 9 – Arquitetura do PipeRench.

Com este dispositivo foi obtido um ganho de desempenho de 189 vezes sobre um Ultrasparc II rodando a 300 MHz para a execução de um algoritmo ATR. Enquanto estes forçam uma serialização de operações intrinsecamente paralelas, PipeRench explora o paralelismo dessas operações, além de aumentar a flexibilidade e diminuir o tempo de projeto do sistema.

3. RESUMO DOS PROCESSADORES RECONFIGURÁVEIS

A Seção anterior percorreu os principais trabalhos em processadores reconfiguráveis. Os processadores PRISM e Nano Processor tinham como principal objetivo mostrar que era possível obter algum ganho de desempenho acoplado processadores com lógica reconfigurável. O DISC, ao contrário das outras implementações, não possui um conjunto de instruções fixas. Nele todas as instruções são dinamicamente chamadas, sendo removidas do dispositivo quando não há mais espaço para a instrução seguinte. O OneChip tem como principal objetivo reduzir o tempo de transferência de dados entre processador e lógica reconfigurável, a partir da inserção de lógica reconfigurável dentro do processador. O Garp foi projetado com o intuito de acelerar laços de aplicação de propósito geral. O Chimera utiliza uma técnica que permite iniciar a execução da instrução reconfigurável antes mesmo da operação ter sido chamada pelo processador. O MorphoSys é um sistema híbrido que possui uma malha reconfigurável com três níveis de interconexão. O NAPA apresenta uma arquitetura que permite interconectar por um barramento múltiplos processadores com lógica reconfigurável dentro do processador. O PipeRench implementa um pipeline que funciona como um coprocessador.

Processadores reconfiguráveis podem ser diferenciados por características como grau de acoplamento entre o processador e a lógica reconfigurável, tipo de instrução a ser executado pela lógica reconfigurável, formato de codificação e passagem de parâmetros das instruções reconfiguráveis, acesso à memória por parte da lógica reconfigurável e outras características relevantes para o

desempenho dos processadores reconfiguráveis. Estas e outras características são resumidas na Tabela 1 e detalha-

das nas seções seguintes.

Tabela 1 – Resumo dos Processadores Reconfiguráveis.

Projeto e Referência	Prism-I	Prism-II	Nano Processor	DISC	OneChip	Garp	Chimaera	Morpho Sys	NAPA	OneChip 98	Pipe Rench
Ano	1993	1993	1994	1995	1996	1997	1997	1998	1998	1999	1999
Processador	M68010	Am29050	RISC	CISC	DLX	MIPS	MIPS	TinyRISC	RISC	DLX	Genérico
Aplicação	Genérico	Genérico	Genérico	Genérico	Genérico	Genérico	Multi-mídia	Genérico	Genérico	Genérico	Multi-mídia
Acoplamento	Anexado	Coproc	RFU	RFU	RFU	Coproc	RFU	Coproc	Coproc	RFU	Anexado
Tipos de Instruções	Todas	Todas	Personalizadas	Todas	Todas	Stream	Personalizada	Todas	Todas	Stream	Stream
Tabela de Configurações	Não	Não	Não	Sim	Não	Sim	Sim	Sim	Não	Sim	-
Operandos	Fixo	Fixo	Fixo	Flexível	Fixo	Fixo	Pré-determinado	Fixo	Fixo	Fixo	Fixo
Banco de Registradores	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Não	Sim	Não
Portas de Memória	Não	Não	Sim (1)	Sim (1)	Sim (1)	Sim (1)	Não	Sim (1)	Sim (2)	Sim (1)	Sim (1)
Granularidade	Fina	Fina	Fina	Fina	Fina	Fina	Fina	Grande	Fina	Fina	Grande
Tipo de Lógica Reconfigurável	3090	4010	3000	CLAY-31	4010	Personalizada	Personalizada	Personalizada	Personalizada	Personalizada	Personalizada
Dinamicam. Reconfigurável	Não	Não	Não	Parcial	Não	Sim	Sim	Sim	Sim	Sim	Sim
Método de Reconfiguração	Soft	Soft	Soft	Soft	Soft	Soft	Controlador	Controlador	Controlador	Controlador	Controlador
Realocável	Não	Não	Não	Sim	Não	Sim	Sim	Não	Sim	Não	Sim
Aceleração	54	86	240	23,5	50	43	160	6,5	-	32	189

3.1 Acoplamento entre processador e lógica reconfigurável

A posição da lógica reconfigurável em relação ao processador tem um impacto direto no desempenho do sistema e no tipo de aplicações que são beneficiadas pela lógica reconfigurável. O benefício da execução de uma instrução em lógica reconfigurável depende de dois aspectos: tempo de comunicação e tempo de processamento [18]. O tempo necessário para executar uma operação é a soma do tempo necessário para enviar dados e receber dados da/lógica reconfigurável, e o tempo de processamento. Se este tempo total é menor que o tempo que o processador leva para executar em software, então um ganho de desempenho é obtido. Se a lógica reconfigurável ainda não foi configurada para realizar aquela instrução específica, então é ainda necessário somar o tempo de configuração.

A lógica reconfigurável pode ser posicionada em três posições em relação ao processador:

- Anexada ao processador. A lógica reconfigurável é conectada a um barramento.
- Coprocessador. A lógica reconfigurável é conectada diretamente ao processador, utilizando um protocolo próprio de comunicação.

- Unidade funcional reconfigurável (RFU). A lógica reconfigurável é colocada dentro do processador.

Os dois primeiros modos de acoplamento são ditos fracamente acoplados, pois a lógica reconfigurável tem que compensar a velocidade de transferência de dados para que a execução em hardware obtenha um ganho de desempenho em relação a software. O último modo é chamado de fortemente acoplado, pois o custo de comunicação entre o processador e a lógica reconfigurável é quase inexistente, já que a lógica reconfigurável encontra-se dentro do processador.

3.2 Tipos de Instruções

O projeto da interface de comunicação entre processador e lógica reconfigurável depende das características das instruções implementadas. Dois tipos principais de instruções podem ser implementados em lógica reconfigurável:

- Stream. Processam grande quantidade de dados em seqüência ou blocos. Este tipo de instrução se adapta bem a abordagem de acoplamento de coprocessador, pois permite paralelismo entre o processador e a lógica reconfigurável.
- Personalizadas. Este tipo de instrução leva um curto período de tempo para executar e produz pouca

quantidade de dados como resultado. Este tipo de instrução se adapta bem a quase todas aplicações, já que impõe poucas restrições as aplicações.

3.3 Formato de codificação das instruções

Instruções reconfiguráveis normalmente possuem um opcode especial e um campo extra. Este campo extra pode especificar:

- O endereço de memória que contém os dados para configurar a instrução.
- Um número que indexa uma tabela de configuração, e as posições desta tabela apontam para a posição de memória onde a configuração está armazenada.

A primeira abordagem possui a desvantagem de necessitar um maior número de bits para endereçar uma memória inteira, enquanto a segunda necessita de uma estrutura de dados especial para armazenar os endereços onde as configurações estão armazenadas.

3.4 Operandos das instruções reconfiguráveis

A palavra de instrução reconfigurável também especifica os operandos da função a ser executada pela lógica reconfigurável. Os operandos podem ser valores imediatos, endereços ou registradores. Existem algumas formas de codificar os operandos:

- Pré-determinado. Os registradores (ou um subconjunto de registradores) do processador são sempre repassados para a lógica reconfigurável. Os registradores utilizados pela instrução dependem do hardware configurado na lógica reconfigurável.
- Fixo. Os operandos estão em uma posição fixa da palavra da instrução. Diferentes formatos de codificação possuem diferentes opcodes.
- Flexível. A posição dos operandos é configurável.

3.5 Acesso à memória

Provendo acesso da hierarquia de memória à lógica reconfigurável é possível implementar carregamento e armazenamento de dados especiais ou instruções de stream. Existem duas técnicas básicas para acesso à memória a partir da lógica reconfigurável:

- Lógica reconfigurável como gerador de endereços. A lógica reconfigurável é utilizada para gerar endereços de posições de memória a serem acessadas.
- Lógica reconfigurável com acesso direto ao barramento de memória. A lógica reconfigurável gerencia acessos à memória gerando dados, endereços e comandos de leitura e escrita.

4. ESTUDO DE CASO: R82R

A Figura 10 apresenta a arquitetura do sistema R8R. O sistema é composto por três módulos: (i) um computador hospedeiro, provendo uma interface para o sistema do usuário; (ii) uma memória de configurações, contendo

todos os bitstreams parciais a serem utilizados durante a execução do sistema; (iii) o FPGA, contendo áreas fixas e reconfiguráveis do sistema R8R.

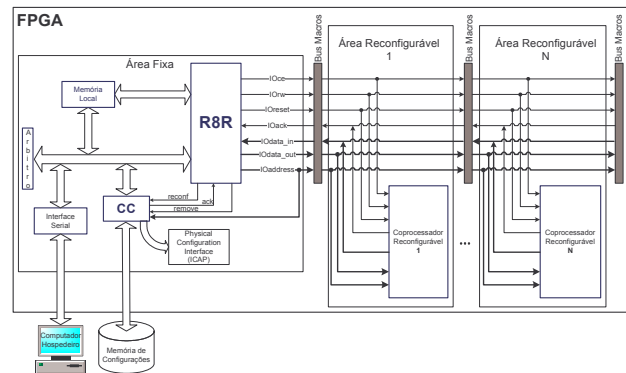


Figura 10 – Arquitetura do sistema R8R.

A área fixa do FPGA é um sistema computacional completo, incluindo o processador R8R, a sua memória local contendo instruções e dados, um barramento do sistema controlado por um árbitro e periféricos (interface serial e controlador de configurações). A interface serial provê comunicação com o computador hospedeiro através de uma interface RS232 padrão. O controlador de configurações (CC) é um periférico desenvolvido para agir como um módulo escravo do processador R8R ou do computador hospedeiro. O computador hospedeiro normalmente preenche ou modifica a configuração da memória antes de iniciar a execução do sistema.

A operação normal do CC é aguardar do processador R8R pedidos de acesso a coprocessadores reconfiguráveis através do sinal *reconf*, enquanto o identificador do coprocessador específico é informado pelo sinal *IOaddress*. Se o coprocessador já está configurado no sistema (uma informação armazenada nas tabelas internas do CC), o sinal *ack* é imediatamente ativado, o qual libera o processador para continuar executando instruções. Se o coprocessador ainda não está configurado em nenhuma área reconfigurável, o processo de reconfiguração é disparado. O CC é responsável por localizar a área de memória que contém o bitstream correspondente ao coprocessador identificado. O bitstream é lido da memória e enviado, palavra por palavra, para a interface física de configuração. Para a família de dispositivos Virtex-II o módulo correspondente é o ICAP. Neste caso, apenas após o término do processo de reconfiguração o sinal *ack* é ativado. O sinal *remove* existe para permitir que o R8R invalide algum coprocessador reconfigurável. Isto é útil para ajudar o CC a escolher a área reconfigurável mais adequada a ser reconfigurada a seguir.

O processador R8R é baseado no processador R8, que é um processador load-store de 16 bits com 40 instruções. O R8 é uma máquina von Neumann, onde as operações lógicas e aritméticas são executadas entre registradores

apenas. O banco de registradores contém 16 registradores de propósito geral. Cada instrução executa em no máximo quatro ciclos de relógio.

O processador é praticamente uma máquina RISC, faltando contudo algumas características que existem em qualquer máquina RISC, tal como *pipelines*. O processador R8 original foi modificado para o processador R8R a partir da adição de cinco novas instruções, responsáveis por dar suporte a utilização de coprocessadores reconfiguráveis. As instruções adicionadas são definidas e descritas abaixo:

- SELR *end*: Seleciona o coprocessador identificado por *end*.
- DISR *end*: Informa ao CC que o coprocessador *end* pode ser removido.
- INTR *end*: Reseta o coprocessador especificado por *end*.
- WRR RS1 RS2: Envia os dados armazenados em RS1 e RS2 ao coprocessador.
- RDR RS RT: Lê dados do coprocessador, armazenando eles no registrador RT.

O processador R8R foi “empacotado” para prover comunicação com (i) a memória local; (ii) o barramento do sistema; (iii) o CC; (iv) as áreas reconfiguráveis. A interface com as áreas reconfiguráveis incluem um conjunto de sinais conectados a bus macros. Bitstreams parciais para o sistema R8R são gerados utilizando o fluxo de Projeto Modular [19].

O protocolo de comunicação entre o processador e o coprocessador é baseado em operações de leitura e escrita. Operações de escrita são utilizadas pelo processador reconfigurável para enviar dados para um coprocessador. Uma operação de escrita utiliza os sinais *IOce*, *IOrw*, *IOdata_out* e *IOaddress*, propagando os mesmos para todas as áreas reconfiguráveis. A área reconfigurável contendo o endereço especificado pelo sinal *IOaddress* captura os dados de *IOdata_out*. A operação de leitura funciona da mesma forma, com a diferença que o fluxo de dados é do coprocessador para o processador. O sinal *IOreset* é ativado pelo processador para inicializar um dos coprocessadores. A comunicação entre processador e coprocessador é feita a partir de bus macros. Neste esquema de comunicação o processador é o mestre do sistema, iniciando todas as comunicações de leitura e escrita. A próxima versão do sistema incluirá a utilização de interrupções para controlar a comunicação, na qual permitirá operações não bloqueantes e comunicações iniciadas pelo coprocessador.

O sistema descrito acima foi completamente prototipado e está operacional em duas versões, uma com um e outra com duas áreas reconfiguráveis. Foi utilizada a plataforma de prototipação V2MB1000 da Memec Insight, com um FPGA XC2V1000 da Xilinx. Para efeitos de validação todo o sistema foi particionado em dois blocos: um incluindo o CC e a interface de comunicação, e outro

com o restante do sistema. Cada bloco foi validado separadamente. A união dos blocos está em desenvolvimento.

No sentido de avaliar o desempenho relativo do sistema reconfigurável implementado, um conjunto de experimentos foi conduzido. Para cada experimento, uma funcionalidade foi implementada tanto em software quanto em hardware, na forma de um coprocessador. Cada implementação foi executada em uma versão do sistema com uma área reconfigurável, e os tempos de execução foram comparados. O tempo de execução para a versão em hardware em cada experimento inclui não somente o tempo de execução do processador, como também o tempo de carga do coprocessador no FPGA a partir da reconfiguração. A Figura 11 apresenta uma comparação entre o número de operações efetuadas e o tempo executado tanto em hardware quanto em software nos três módulos de operações aritméticas de 16/32 bits: multiplicação, divisão e raiz quadrada. É importante ressaltar que para cada módulo, o tempo de execução cresce linearmente, mas com diferentes inclinações para hardware e software. É importante notar também que o tempo de reconfiguração adiciona uma latência fixa de 10ms para as implementações em hardware. O ponto de quebra para cada funcionalidade determina quando uma determinada implementação em hardware começa a ser vantajosa com relação a uma implementação em software, baseada no número de vezes que este hardware é empregado antes de ser reconfigurado. O multiplicador, a divisão e a raiz quadrada começam a ser vantajosas a partir de 750, 260 e 200 execuções, respectivamente. Este simples experimento mostra na prática o quanto é possível obter vantagem de SDRs, alcançando ganhos de desempenho, flexibilidade e redução em área.

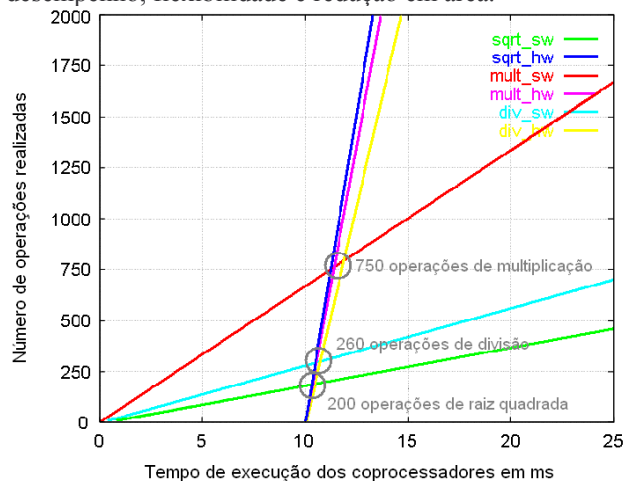


Figura 11 – Tempos de execução e número de operações para três módulos aritméticos, multiplicação, divisão e raiz quadrada, implementadas em hardware (sufixo *hw*) e em software (sufixo *sw*).

5. CONCLUSÃO

Os processadores reconfiguráveis, apresentados ao longo deste artigo, mostraram diversas formas de fazer uso da lógica reconfigurável sem perder a facilidade de programação dos processadores. A maior parte destes utiliza a lógica reconfigurável como um coprocessador, o que resulta em um dos casos estudados desempenho 240 vezes superior que uma abordagem sem lógica reconfigurável. Ganhos tão significativos são atingidos quando é possível contornar o problema de baixo desempenho do processador em situações críticas, como repetição de um pequeno trecho de código em software.

Ainda que processadores reconfiguráveis seja o principal aspecto a ser considerado para a construção de um sistema reconfigurável com elevado poder de processamento, é necessário que existam compiladores e ferramentas de CAD que auxiliem o processo de desenvolvimento. Os compiladores devem aceitar instruções de comunicação entre o processador e a lógica reconfigurável, de forma a permitir a troca de dados e as chamadas das diferentes funções implementadas em lógica reconfigurável. As ferramentas de CAD devem prover um ambiente para criar hardware para a arquitetura reconfigurável e suportar o co-desenvolvimento e a co-simulação de hardware e software.

6. REFERÊNCIA BIBLIOGRÁFICA

- [1] J. Hennessy; D. Patterson. "Computer Architecture: A Quantitative Approach". Estados Unidos, San Mateo, Morgan Kauffmann Publishers, 1990, 594 p.
- [2] P. Athanas; H. Silverman. "Processor Reconfiguration through Instruction-Set Metamorphosis". *Computer*, Volume: 26 (3), Mar. 1993, pp.11-18.
- [3] Xilinx, Inc. <http://www.xilinx.com>.
- [4] M. Wazlowski; L. Agarwal; T. Lee; A. Smith; E. Lam; P. Athanas; H. Silverman; S. Ghosh "PRISM-II Compiler and Architecture". In: *FPGAs for Custom Computing Machines (FCCM)*, 1993, pp. 9-16.
- [5] M Wirthlin; B. Hutchings; K. Gilson. "The Nano Processor: a Low Resource Reconfigurable Processor". In: *FPGAs for Custom Computing Machines (FCCM)*, 1994, pp. 23-30.
- [6] M. Wirthlin; B. Hutchings. "A Dynamic Instruction Set Computer". In: *FPGAs for Custom Computing Machines (FCCM)*, 1995, pp. 99-107.
- [7] R. Wittig; P. Chow. "OneChip: an FPGA Processor with Reconfigurable Logic". In: *FPGAs for Custom Computing Machines (FCCM)*, 1996, pp. 126-135.
- [8] D. Lewis; D. Galloway; M. Ierssel; J. Rose; P. Chow. "The Transmogripher-2: a 1 Million Gate Rapid Prototyping System". In: *IEEE Transactions on Very Large Scale Integration Systems*, 1998, p. 188-198.
- [9] J. Hauser; J. Wawrzynek. "Garp: a MIPS Processor with a Reconfigurable Coprocessor". In: *FPGAs for Custom Computing Machines (FCCM)*, 1997, pp. 12-21.
- [10] S. Hauck; T. Fry; M. Hosler; J. Kao. "The Chimaera Reconfigurable Functional Unit". In: *FPGAs for Custom Computing Machines (FCCM)*, 1997, pp. 87-96.
- [11] C. Ebeling; L. McMurchie; S. Hauck; S. Burns. "Placement and Routing Tools for the Triptych FPGA". *IEEE Transactions on Very Large Scale Integration Systems*, Volume: 3 (4), Dec. 1995, pp. 473-482.
- [12] Altera, Inc. <http://www.altera.com>.
- [13] R. Razdan; K. Brace; D. Smith; "PRISC software acceleration techniques". In: *IEEE International Conference on Computer Design (ICCD), VLSI in Computer & Processors*, 1994, pp. 145-149.
- [14] H. Singh; M. Lee; G. Lu; F. Kurdahi; N. Bagherzadeh; E. Filho. "MorphoSys: a Reconfigurable Architecture for Multimedia Applications". In: *Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI)*, 1998. pp. 134-139.
- [15] C. Rupp; M. Landguth; T. Garverick; E. Gomersall; H. Holt; J. Arnold; M. Gokhale. "The NAPA Adaptive Processing Architecture". In: *FPGAs for Custom Computing Machines (FCCM)*, 1998, pp. 28-37.
- [16] J. Jacob; P. Chow. "Memory Interfacing and Instruction Specification for Reconfigurable Processors". In: *Field Programmable Gate Arrays (FPGA)*, 1999, pp. 145-154.
- [17] S. Goldstein; H. Schmit; M. Moe; M. Budiu; S. Cadambi; R. Taylor; R. Laufer. "PipeRench: a Coprocessor for Streaming Multimedia Acceleration". In: *International Symposium on Computer Architecture (ISCA)*, 1999, pp. 28-39.
- [18] F. Barat; R. Lauwereins; G. Deconinck. "Reconfigurable Instruction Set Processors from a Hardware/Software Perspective". *IEEE Transactions on Software Engineering*, Volume: 28 (9), Sep. 2002, pp. 847-862.
- [19] D. Lim; M. Peattie. "Two Flows For Partial Reconfiguration: Module Based or Small Bit Manipulations". *Xilinx Application Note 290 (XAPP290 v1.2)*, 2004, 28 p.