PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMAS DINAMICAMENTE RECONFIGURÁVEIS COM COMUNICAÇÃO VIA REDES INTRA-CHIP

Plano de Estudo e Pesquisa

Mestrando: Leandro Heleno Möller

Orientador: Prof. Fernando Gehm Moraes

Porto Alegre, Junho de 2005. (versão revisada)

Sumário

Lista de Figuras

Listas de Abreviaturas

API	*Application Programming Interface*
BRAM	*Block Select RAM*
CAD	*Computer Aided Design*
CLB	*Configurable Logic Block*
DCM	*Digital Clock Manager*
FPGA	*Field Programmable Gate Array*
ICAP	*Internal Configuration Access Port*
IOB	*Input Ouput Block*
LUT	*Look-Up Table*
NoC	*Network on Chip*
RAM	*Random Access Memory*
RHOS	*Reconfigurable Hardware Operating Systems*
SDR	Sistema Dinamicamente Reconfigurável
SoC	*System on Chip*
VCI	*Virtual Component Interconnect*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VLSI	*Very Large Scale Integration*

1 INTRODUÇÃO

Dispositivos reconfiguráveis são utilizados tanto na prototipação de sistemas digitais, quanto em produtos finais. A vantagem da utilização destes dispositivos em lugar de outras alternativas é a capacidade de poder modificar o hardware, permitindo a atualização do produto no campo e, por conseqüência, podendo assegurar um tempo de vida maior ao mesmo. Outra vantagem associada à utilização de dispositivos reconfiguráveis em produtos finais é a redução do tempo de chegada ao mercado, pois a fase de fabricação do circuito integrado não esta incluída no ciclo de fabricação do produto, já que o dispositivo reconfigurável é comprado pronto, necessitando apenas a configuração do dispositivo com a lógica específica do produto. O ponto negativo de dispositivos reconfiguráveis em relação a outras tecnologias disponíveis no mercado é um maior consumo de área de silício e potência, devido ao fato de possuírem componentes adicionais para habilitar a modificabilidade do hardware.

A definição de qual operação o dispositivo reconfigurável executará dá-se através de um arquivo de configuração (*bitstream*). Chama-se de *reconfiguração total* a modificação completa do hardware através de um novo bitstream que descreve uma nova operação para o dispositivo reconfigurável, e *reconfiguração parcial* a alteração de uma parte das funções implementadas no dispositivo reconfigurável. Esta reconfiguração pode ser *estática*, necessitando que o sistema pare de operar para que ocorra a modificação do hardware, ou *dinâmica*, permitindo que partes do sistema continuem executando enquanto as alterações são efetuadas. *Sistemas reconfiguráveis* são sistemas eletrônicos, digitais ou analógicos, que utilizam em sua implementação dispositivos reconfiguráveis, onde a reconfiguração é uma característica importante no projeto.

As principais justificativas para a utilização de *sistemas reconfiguráveis* são o aumento do tempo de vida do produto, a miniaturização do hardware e a possibilidade de especialização de funções em produtos genéricos. O tempo de vida do produto pode ser estendido com a reconfiguração do hardware para adicionar novas funcionalidades, novos protocolos de comunicação ou corrigir defeitos do equipamento. A miniaturização do produto é obtida com uma administração inteligente do dispositivo reconfigurável, mantendo configurado apenas o hardware que está sendo atualmente utilizado. O fato de não necessitar que todo o sistema esteja ocupando o hardware no mesmo instante de tempo permite economizar energia, sendo esta uma questão importante na durabilidade e tamanho das baterias de equipamentos portáteis. A especialização de funções em produtos genéricos pode ser facilmente exemplificada. Considere-se em um produto genérico como sendo um computador de mão e como produto específico uma máquina fotográfica digital. Existem computadores de mão que tiram fotografias, mas a sua qualidade não se compara a uma máquina fotográfica digital, pois o computador de mão faz todo o processamento em software. Com a utilização de dispositivos reconfiguráveis, o computador de mão (sistema genérico) pode ser configurado para possuir o hardware necessário de uma máquina fotográfica digital (sistema específico), assim obtendo melhor desempenho que a alternativa somente software.

1.1 Objetivos

O *objetivo estratégico* do presente trabalho é dominar a tecnologia de sistemas digitais reconfiguráveis e a utilização de redes intra-chip como meio de comunicação entre núcleos de propriedade intelectual em circuitos integrados em um único chip. É também objetivo estratégico do presente trabalho a união destes dois domínios para a obtenção de SDRs de grande porte.

O *objetivo específico* é implementar uma plataforma multiprocessada dinamicamente reconfigurável construída sobre uma rede intra-chip eficiente e de baixo consumo de área. A característica desta plataforma ser dinamicamente reconfigurável permite que o sistema adapte o seu hardware durante o funcionamento do mesmo, adicionando flexibilidade à plataforma. As qualidades esperadas desta plataforma são ser capaz de executar tanto aplicações genéricas simultaneamente quanto executar uma aplicação específica que necessite elevado poder computacional. Esta plataforma será implementada como um sistema em um único chip (*System on Chip* - SoC) e utilizará uma rede intra-chip (*Network on Chip* - NoC) para interconectar os diversos núcleos do sistema. Este sistema será implementado utilizando dispositivos FPGAs que permitam reconfiguração parcial e dinâmica, de forma que os núcleos do sistema possam ser substituídos em tempo execução.

A implementação desta plataforma visa o avanço tecnológico tanto de arquiteturas paralelas que demandam *alto poder de processamento* quanto para a construção de *dispositivos eletrônicos portáteis*. Nesta plataforma, um maior *poder de processamento* em relação a arquiteturas paralelas (e.g. *clusters*) pode ser teoricamente obtido porque os elementos de processamento (EPs) estão fisicamente mais próximos (no mesmo chip), assim reduzindo o atraso na comunicação entre os EPs. Somado a isso, cada EP pode ainda ser implementado em hardware, permitindo assim um funcionamento interno paralelo, em contraponto à execução seqüencial de processadores (busca-decodifica-executa instruções). Da mesma forma os *dispositivos portáteis* podem se beneficiar da redução do atraso da comunicação e do paralelismo de hardware provido pela plataforma, além de reduzir o consumo de potência, premissa básica quando sistemas são alimentados por baterias. A redução de potência pode ser justificada pelo fato que apenas os módulos configurados estarão consumindo energia.

1.2 Organização do documento

O Capítulo 2 apresenta o referencial teórico para uma melhor compreensão do trabalho a ser realizado. As atividades a serem executadas no decorrer do presente trabalho são abordadas no Capítulo 3.

2 REFERENCIAL TEÓRICO

Arquiteturas reconfiguráveis de grão grande [HAR01] formam a classe de sistemas abordados no presente trabalho. Estas arquiteturas possuem como elementos de processamento de ULAs [MIR96] a processadores RISC [WAI97], conectados através de redes simples (conexão de vizinhança). Arquiteturas de grão grande tendo seus núcleos conectados por uma NoC representam um avanço para este tipo de arquitetura, pois agregam maior escalabilidade quando comparada a barramentos e maior gerência da comunicação entre os EPs da arquitetura. Um breve estudo sobre arquiteturas reconfiguráveis de grão grande é apresentado na Seção 2.1.

Para a implementação do sistema proposto neste trabalho será utilizado o FPGA Virtex-II da Xilinx [XIL04b]. Este dispositivo atende os requisitos de ser parcial e dinamicamente reconfigurável e comportar EPs relativamente grandes (eg. processadores). Mais informações sobre os dispositivos Virtex II e a justificativa para utilização do mesmo em relação a outros existentes são apresentados na Seção 2.2.

Complementar ao dispositivo escolhido para prototipação, deve-se ter um fluxo de projeto que permita criar *Sistemas Dinamicamente Reconfiguráveis* (SDRs). Um passo importante deste fluxo é que seja possível restringir o posicionamento e o roteamento de núcleos reconfiguráveis em uma área específica do FPGA. O fluxo também deve suportar a geração de bitstreams parciais dos núcleos reconfiguráveis do sistema. A Seção 2.3 apresenta o fluxo de implementação de projetos VLSI para FPGAs.

Por fim, são apresentados, na Seção 2.4, conceitos básicos de NoCs e informações sobre a NoC Hermes, a qual será utilizada no presente trabalho.

2.1 Arquiteturas Reconfiguráveis de Grão Grande

Arquiteturas reconfiguráveis de grão pequeno possuem como menor bloco reconfigurável *flip-flops* e LUTs. Estas arquiteturas reconfiguráveis se popularizaram com os FPGAs, tornando-se naturais candidatas para a prototipação de sistemas digitais por sua flexibilidade. No entanto, projetos de sistemas digitais atuais tratam operações com palavras de dezenas de bits, tornando arquiteturas de grão pequeno ineficientes em termos de área, roteamento, consumo de energia e desempenho.

Arquiteturas reconfiguráveis de grão grande possuem unidades funcionais em nível de palavra e menos chaves e roteamento interconectando elementos de processamento (*Processing Element* - PE). Isto simplifica o roteamento e reduz o atraso entre os PEs, aumentando o desempenho da arquitetura. Além disso, chaves consomem mais energia e área que conexões diretas. Os processos de síntese e reconfiguração também são mais rápidos em dispositivos de grão grande, pois a complexidade é reduzida, devido a um menor número de chaves de roteamento e elementos reconfiguráveis.

Arquiteturas reconfiguráveis de grão grande possuem uma estrutura regular de bloco de dados replicados e interconectados por um meio de comunicação. Alguns exemplos de arquiteturas reconfiguráveis de grão grande são apresentados na Tabela 1. Dentre estes se percebe que apenas topologias malha, array e anel foram utilizadas para a interconexão de PEs. Acredita-se que esta escolha deve-se à regularidade da implementação física e à facilidade de implementação dos

algoritmos responsáveis pela transferência de dados.

A organização dos PEs em malha favorece a conexão com os vizinhos mais próximos, o que pode tornar o tempo de comunicação com PEs distantes muito longo, em termos de ciclos de relógio. Por este motivo, algumas arquiteturas que utilizam a topologia malha apresentam interconexões secundárias. Exemplos destes casos são o KressArray, o MATRIX e o DReAM.

O KressArray possui blocos de dados de 32 bits, um barramento global que interconecta todos os PEs e conexões locais para a comunicação dos PEs vizinhos. O MATRIX foi o primeiro a estender os PEs a pequenos processadores, dispondo de memórias locais. O RAW apresenta como PEs processadores RISC completos e memórias de instruções e dados. O MorphoSys é um sistema híbrido, que possui uma malha reconfigurável com três níveis de interconexão. O CHESS interconecta os PEs na forma de um tabuleiro de xadrez, possuindo memórias embarcadas. O DReAM apresenta uma rede de interconexão complexa voltada para aplicações multimídia.

Tabela 1 – Resumo das arquiteturas reconfiguráveis de grão grande.

Projeto e Referência	Ano	Topologia	Granularidade	Conexão	Aplicação Alvo	Memória Embarcada
KressArray-I [HAR95]	1995	Malha	32 bits	Conexão aos vizinhos e barramento segmentado	Indefinida	Não
RaPiD [EBE96]	1996	Array Linear	16 bits	Barramento segmentado	Indefinida	Sim
MATRIX [MIR96]	1996	Malha	8 bits Parametrizável	Conexão aos vizinhos, conexão a cada 4, barramento global	Indefinida	Sim
RAW [WAI97]	1997	Malha	8 bits Parametrizável	Conexão aos vizinhos	Indefinida	Sim
KressArray-III [HAR97]	1997	Malha	32 bits	Conexão aos vizinhos, linhas longas, barramento global	Indefinida	Não
Morphosys [SIN98]	1998	Malha	16 bits	Conexão aos vizinhos, conexão a cada 2 ou 3, barramento global	Indefinida	Não
CHESS [MAR99]	1999	Malha	4 bits	Conexão aos vizinhos	Multimídia	Sim
DReAM [BEC00]	2000	Malha	8 e 16 bits	Conexão aos vizinhos e barramentos segmentados	Multimídia	Sim
Systolic Ring [SAS01]	2001	Anel	16 bits	Conexão aos vizinhos	Multimídia	Não

Uma forma mais simples de interconexão de PEs é por array linear. Essa estrutura é interessante para aplicações com um fluxo uniforme e geralmente são utilizadas como *pipelines*, onde cada PE executa um estágio do *pipeline*. Essa é uma abordagem que funciona bem para aplicações com apenas uma linha de execução. A Tabela 1 apresentou apenas o RaPiD como exemplo de arquitetura que interconecta os PEs em array. O RaPiD é baseado na idéia de prover aceleração em aplicações que executam computações intensivas e de estrutura regular. Outra arquitetura que poderia ser citada como array, porém com uma estrutura circular, é o Systolic Ring [SAS01], voltado para aplicações multimídia.

Todos os exemplos providos na Tabela 1 são propostas de arquiteturas reconfiguráveis acadêmicas. Estes dispositivos possuem ferramentas de CAD limitadas e pouco documentadas para

viabilizar seu uso em reconfiguração parcial e dinâmica.

A carência de dispositivos reconfiguráveis de grão grande que se adaptem ao sistema proposto neste trabalho induz a utilização de dispositivos FPGA de grão pequeno. Os dois maiores fabricantes de dispositivos reconfiguráveis de grão pequeno disponíveis no mercado que suportam reconfiguração parcial e dinâmica são a Xilinx [XIL04a] e a Atmel [ATM04]. Dentre estes se optou pelos dispositivos Xilinx por diversos motivos: disponibilidade de documentação, trabalhos relacionados ao assunto tanto no meio acadêmico [HUB04a] [DYE02] [MÖL04] quanto na indústria, existência de ferramentas que permitem interagir nas fases intermediárias entre a descrição inicial de hardware até a prototipação em FPGAs, diversidade de plataformas e a alta densidade em termos de portas lógicas dos dispositivos. A Seção 2.2 apresenta maiores detalhes da família de dispositivos Virtex-II da Xilinx, selecionada para embarcar o SDR proposto neste trabalho.

2.2 Virtex-II

Esta Seção apresenta a arquitetura da família Virtex-II da Xilinx, selecionada para embarcar o SDR proposto neste trabalho. É dada ênfase nesta Seção aos recursos de configuração do dispositivo, já que esta é a característica primária para a construção de um SDR. Todas as informações contidas foram retiradas de [XIL04b]. A Figura 1 apresenta uma visão geral dos dispositivos Virtex-II.

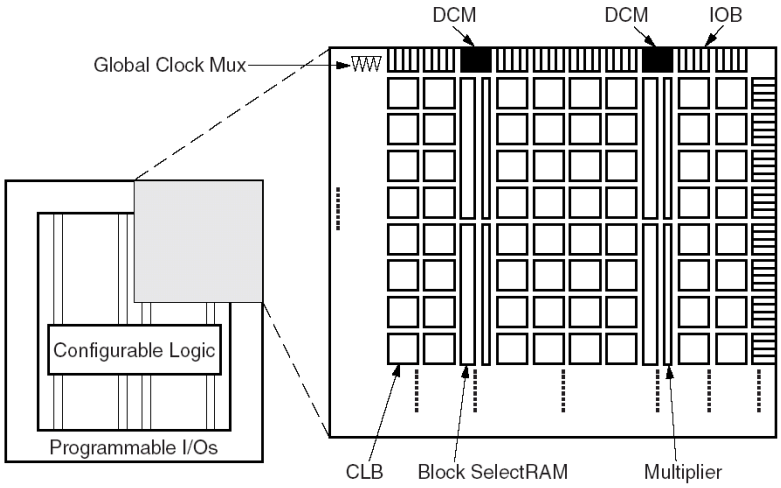


Figura 1 – Arquitetura interna do dispositivo Virtex-II.

A Virtex-II é composta basicamente por blocos lógicos configuráveis (CLBs – *Configurable Logic Blocks*), blocos de entrada e saída (IOBs – *Input/Output Blocks*), blocos de memória (BRAMs - *Block Select RAMs*), gerenciadores digitais de relógios (DCMs – *Digital Clock Manager*), multiplicadores e recursos de roteamento que interconectam todos estes componentes. As CLBs provêem elementos funcionais para a implementação de lógica combinacional e/ou seqüencial. Cada CLB possui quatro *slices* e dois *tristates*. A Figura 2 apresenta a metade superior de uma CLB e compreende geradores de função de quatro entradas (Look-Up Tables - *LUTs*), flip-flops para armazenamento de um bit e recursos como propagação rápida de *carry*. Cada BRAM possui 18Kb de memória dupla porta, programável em diversas larguras de dados e endereços. A Tabela 2 apresenta a quantidade desses e outros recursos existentes em cada dispositivo da família Virtex-II.

Tabela 2 – Número de recursos internos de cada dispositivo da família Virtex-II.

Dispositivo	Quantidade de Portas Lógicas Equivalentes (típica)	1 CLB = 4 slices = Máximo 128 bits			Blocos Multiplicadores	BRAMs		DCMs	Número Máximo de pinos de I/O
		Matriz linhas x colunas	Slices	Máximo de RAM distribuída (Kbits)		Blocos de 18 Kbits	Máximo de RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5120	160	40	40	720	8	432
XC2V1500	1,5M	48 x 40	7680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10752	336	56	56	1008	8	624
XC2V3000	3M	64 x 56	14336	448	96	96	1728	12	720
XC2V4000	4M	80 x 72	23040	720	120	120	2160	12	912
XC2V6000	6M	96 x 88	33792	1.056	144	144	2592	12	1104
XC2V8000	8M	112 x 104	46592	1.456	168	168	3024	12	1108

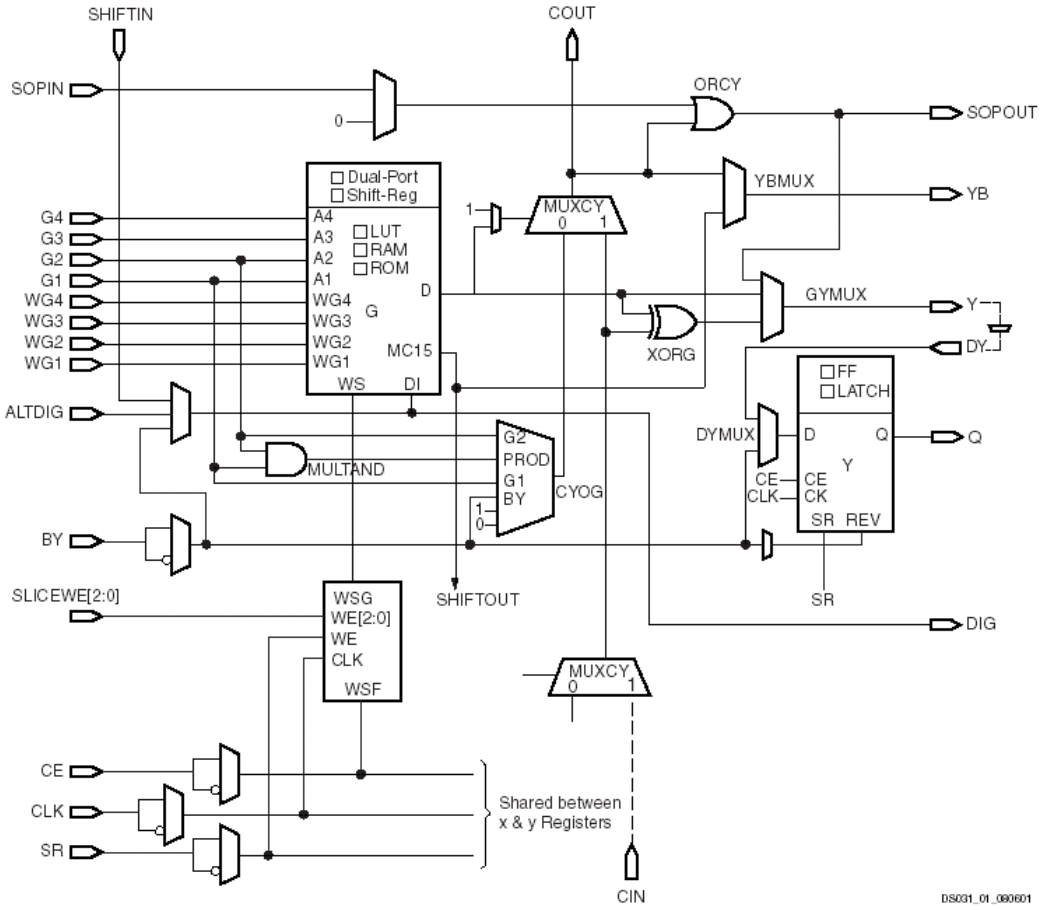


Figura 2 – Metade superior de um slice da Virtex-II.

A arquitetura de (re)configuração dos dispositivos Virtex é organizada como uma matriz bidimensional de bits. No entanto, o dispositivo só pode ser configurado unidimensionalmente em colunas, onde cada coluna é composta por frames. O frame é a menor unidade que pode ser lida ou escrita do FPGA. Quanto menor o número de linhas de CLBs do FPGA, menor é o tamanho do frame e conseqüentemente mais rápido é o tempo de reconfiguração de um frame. A Tabela 3 apresenta o número de bits de um frame dos dispositivos apresentados na Tabela 2, assim como o tempo de reconfiguração de um bitstream total para diversos modos de configuração.

Tabela 3 – Informações para a reconfiguração de dispositivos Virtex-II.

Dispositivo	Número de Frames	Tamanho do Frame em bits	Bits de Configuração	Total de bits (incluindo cabeçalho)	Tempo de download SelectMap (50 MHz) em ms	Tempo de download Serial (50 MHz) em ms	Tempo de download JTAG (33 MHz) em ms
XC2V40	404	832	360.096	339.040	0,84	6,72	10,19
XC2V80	404	1472	635.296	598.880	1,49	11,89	18,02
XC2V250	752	2112	1.697.184	1.593.696	3,97	31,76	48,13
XC2V500	928	2752	2.761.888	2.560.608	6,38	51,08	77,39
XC2V1000	1104	3392	4.082.592	3.752.800	9,36	74,90	113,48
XC2V1500	1280	4032	5.659.296	5.170.272	12,90	103,22	156,39
XC2V2000	1456	4672	7.492.000	6.813.024	17,01	136,05	206,13
XC2V3000	1804	5312	10.494.368	9.594.720	23,96	191,66	290,39
XC2V4000	2156	6592	15.659.936	14.226.784	35,53	284,25	430,68
XC2V6000	2508	7872	21.849.504	19.759.968	49,36	394,86	598,27
XC2V8000	2860	9152	29.063.072	26.194.272	65,44	523,49	793,17

Tabela 4 – Significado dos pinos de entrada e saída do ICAP.

Porta ICAP	Descrição
I [0:7]	Barramento de entrada de dados por onde é feita uma configuração. O pino I [0] é o bit mais significativo.
O [0:7]	Barramento de saída de dados por onde é feito *readback* e por onde saem informações de estado durante uma configuração. O pino O [0] é o bit mais significativo do byte.
BUSY	Indica quando o FPGA pode aceitar outro byte. Quando BUSY está em nível lógico baixo, o FPGA lê o barramento de dados na próxima borda de subida de CCLK quando ambos, CE e WRITE estiverem em nível lógico baixo. Se BUSY estiver em nível lógico alto, o byte atual é ignorado e precisa ser mantido no barramento até a próxima borda de descida de CCLK, quando BUSY estiver em nível lógico baixo.
CE	Habilita o ICAP. Este pino é ativo em nível lógico baixo, embora esta polaridade possa ser invertida através da ferramenta FPGA Editor.
WRITE	Indica se está sendo feita uma configuração ou um *readback* do FPGA. É ativado em nível lógico baixo, podendo ser invertido via FPGA Editor, assim como o sinal CE. Quando em nível lógico baixo, WRITE indica que um byte está sendo escrito através do barramento I [0:7]. Quando em nível lógico alto, WRITE indica que um byte está sendo lido através do barramento O [0:7].
CCLK	Sincroniza as leituras e escritas dos barramentos de configuração e *readback*.

Os dispositivos da família Virtex-II possuem uma porta interna de acesso de configuração denominada ICAP (*Internal Configuration Access Port*). Este componente, apresentado na Figura 3, está localizado na parte inferior direita do FPGA e permite que o dispositivo se auto reconfigure. O ICAP pode funcionar a uma velocidade máxima de 66 MHz [BLO03]. A interface de comunicação do ICAP, apresentada na Tabela 4, é semelhante à interface SelectMAP, não possuindo os pinos utilizados exclusivamente para reconfiguração total (DONE, INIT e PROGRAM) e os pinos de modo de configuração (M0, M1 e M2).

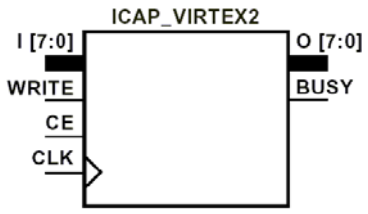


Figura 3 – Interfaces de entrada e saída do dispositivo ICAP.

A Tabela 5 apresenta, além de alguns dados já informados na Tabela 3, o tempo de reconfiguração parcial de um frame de diferentes dispositivos da família Virtex-II. As duas últimas colunas da tabela apresentam os tempos de reconfiguração parcial aproximados para reconfiguração a partir da interface SelectMAP e a partir do módulo ICAP. É importante ressaltar que com exceção de pequenas reconfigurações, por exemplo reconfiguração de parâmetros, geralmente uma coluna inteira será modificada. Os diferentes tipos de colunas e o número de frames gasto para reconfigurar um tipo de coluna é apresentado na Tabela 6. A partir da informação de que um núcleo ocupa uma coluna de CLBs, sabe-se que são necessários reconfigurar 22 frames, onde 22 frames de um dispositivo XC2V40, utilizando a interface ICAP, demoram (22 x 1,726) 37,972 μs para serem reconfigurados.

Tabela 5 – Tempos de reconfiguração de diferentes dispositivos da família Virtex-II.

Dispositivo	Número de frames por dispositivo	Tamanho do frame em bits	Tempo de reconfiguração total SelectMAP (50 MHz) em μs	Tempo de reconfiguração de um frame SelectMAP (50 MHz) em μs	Tempo de Reconfiguração de um frame ICAP (66 MHz) em μs
XC2V40	404	832	840	2,079	1,726
XC2V80	404	1472	1490	3,688	3,061
XC2V250	752	2112	3970	5,279	4,382
XC2V500	928	2752	6380	6,875	5,707
XC2V1000	1104	3392	9360	8,478	7,037
XC2V1500	1280	4032	12900	10,078	8,365
XC2V2000	1456	4672	17010	11,683	9,697
XC2V3000	1804	5312	23960	13,282	11,024
XC2V4000	2156	6592	35530	16,480	13,678
XC2V6000	2508	7872	49360	19,681	16,335
XC2V8000	2860	9152	65440	22,881	18,992

Tabela 6 – Número de frames por tipo de coluna para dispositivos da família Virtex-II.

Tipo de coluna	Número de frames por coluna	Número de colunas por dispositivo
IOB1	4	2
IOB2	22	2
CLB	22	Nro de colunas de CLBs
BRAM	64	Nro de colunas de BRAMs
BRAM Interconnect	22	Nro de colunas de BRAMs
GCLK	4	1

2.3 Fluxo de projetos VLSI em dispositivos FPGAs

Esta Seção apresenta uma introdução ao fluxo utilizado na implementação de projetos VLSI em FPGAs e respectivas alternativas de ferramentas para cada um dos passos. Este fluxo inicia com um projeto em linguagem de descrição de hardware e é finalizado pela geração do bitstream. É importante conhecer cada um desses passos e as respectivas ferramentas, pois algumas destas farão uso de parâmetros avançados e específicos quando aplicadas a SDRs.

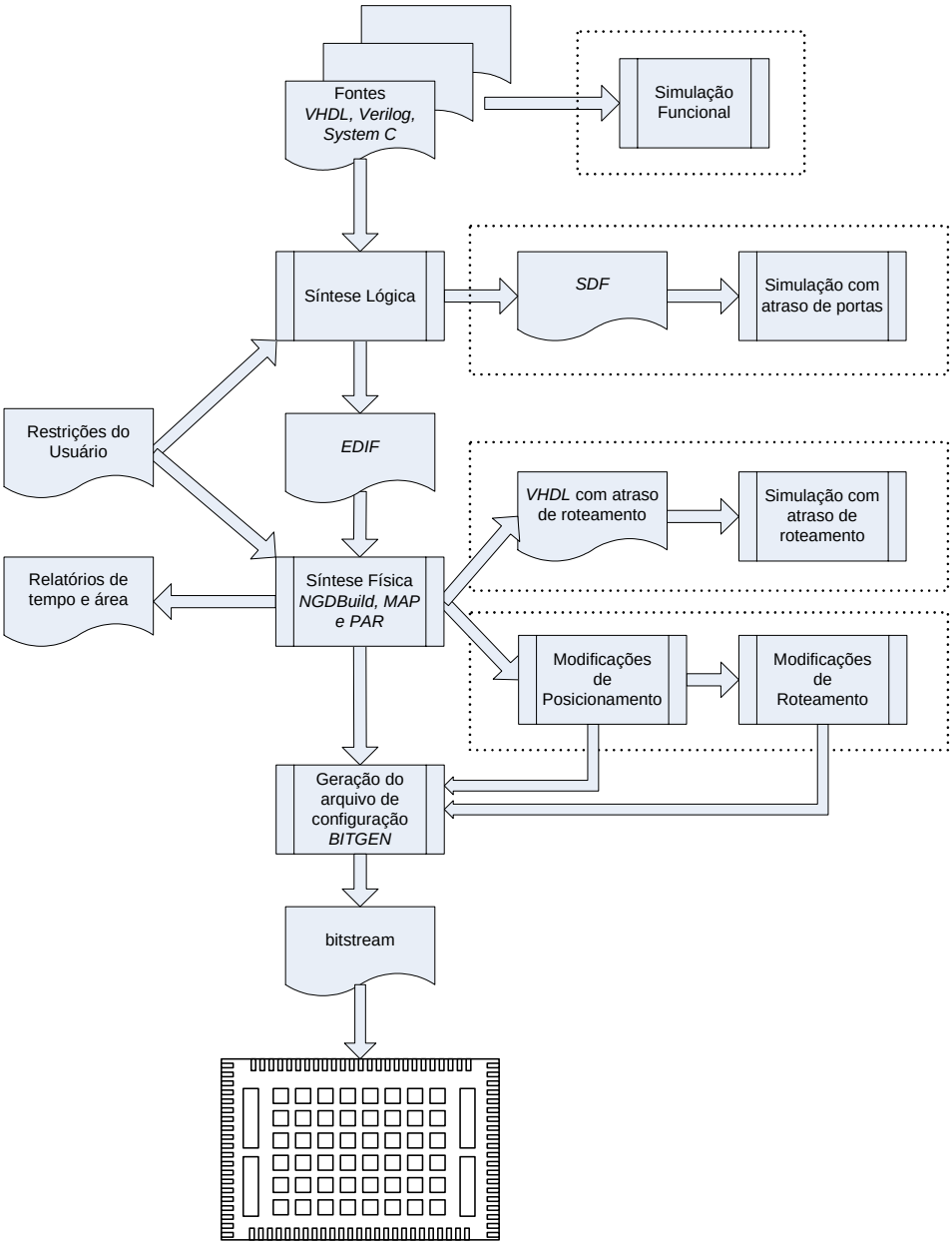


Figura 4 – Fluxo básico de projeto de FPGAs Xilinx que inicia com o projeto em linguagem de descrição de hardware e termina com o bitstream gerado. Caixas pontilhadas representam passos opcionais do fluxo.

O primeiro passo apresentado na Figura 4 é a simulação funcional do projeto descrito em linguagem de descrição de hardware (VHDL, Verilog ou System-C RTL). Nesta etapa, é possível verificar o correto funcionamento lógico do projeto a partir da criação de vetores de teste ligados à entrada do mesmo. O passo seguinte é a síntese lógica do hardware. Algumas alternativas de ferramentas de síntese lógica são o XST da Xilinx, o Leonardo Spectrum da Mentor e o Synplify da Synplicity. Estas ferramentas suportam uma série de otimizações selecionáveis pelo usuário, como maior controle do nível de esforço para redução de área, velocidade ou potência. Estas ferramentas geram uma descrição lógica do circuito em formato EDIF (*Electronic Digital Interchange Format*). A ferramenta de síntese lógica também pode gerar um arquivo SDF (*Standard Delay Format*), que contém os atrasos das portas lógicas. Isto permite fazer uma simulação mais precisa, possibilitando encontrar erros de temporização, não encontrados na simulação funcional. O arquivo EDIF, gerado pela ferramenta de síntese lógica, acompanhado do arquivo de restrições de recursos do usuário

(UCF – *User Constraints File*), específico do FPGA alvo, são as entradas para a fase de síntese física. A síntese física é responsável pelo mapeamento das portas lógicas do arquivo EDIF para os recursos disponíveis no FPGA, pelo posicionamento e pelo roteamento destes recursos. As ferramentas da Xilinx que executam estes passos são NGDBuild, MAP e PAR. Após o término da execução destas ferramentas um bitstream total e uma série de relatórios são gerados. Entre os principais relatórios analisados estão o de velocidade e o de área, que informam os maiores atrasos do projeto e a ocupação de área deste no FPGA. O bitstream total do projeto é gerado pela ferramenta BitGen da Xilinx, podendo este ser configurado no FPGA a partir da execução da ferramenta Impact.

Esta Seção versou sobre a execução do fluxo de projeto para um projeto de hardware genérico sem características reconfiguráveis. As seções 2.3.1 a 2.3.5 apresentam o fluxo de projeto para SDRs. Neste contexto faz-se antes necessário definir os termos *sistema* e *projeto*. Utilizar-se-á *sistema* como sendo o equivalente a SDR e possuindo diversos projetos. Cada *projeto* é um dos possíveis estados atuais de configuração do sistema.

2.3.1 Definir projeto inicial do SDR

O primeiro passo do fluxo de projeto é definir a estrutura geral do SDR, escolhendo a quantidade de áreas reconfiguráveis que farão parte deste e quais núcleos estarão inicialmente configurados nestas áreas reconfiguráveis. É aconselhável prototipar este projeto usando configuração total, de forma a garantir que as restrições temporais deste sejam atendidas e que o mesmo funcione em hardware. O motivo para esta prototipação inicial é garantir a operacionabilidade do ponto de partida do SDR, uma vez que serão posteriormente adicionados novos elementos, agregando complexidade ao todo.

2.3.2 Definir e inserir componentes de comunicação entre áreas reconfiguráveis

O passo seguinte é a adição de componentes provedores de comunicação controlada entre os núcleos reconfiguráveis e o restante do projeto. Estes componentes estabelecem um ponto em comum conhecido entre cada um dos núcleos instanciados em diferentes projetos do sistema. Portanto, componentes de comunicação devem ser fixos no sistema e os núcleos devem respeitar a semântica de cada pino deste para o correto funcionamento do sistema. Outra atribuição destes componentes é isolar o restante do sistema enquanto um núcleo está sendo reconfigurado. Este isolamento é necessário porque os dados recebidos da reconfiguração modificam transitoriamente o núcleo, podendo assim ocorrer ações espúrias de chaveamento que modifiquem o valor da saída do núcleo, interferindo no funcionamento do restante do sistema que continua operando. Com um componente de comunicação adequadamente projetado é possível isolar a área que está sendo reconfigurada de forma que este não interfirá com o funcionamento do restante do sistema.

O componente proposto pela Xilinx para esta tarefa é a *bus macro* [LIM04] [XIL04c]. A Figura 5(a) apresenta o diagrama esquemático da bus macro, sendo esta formada por oito *tristates* e disponibilizando quatro fios de um bit para a comunicação do sistema com o núcleo reconfigurável ou vice-versa. Vários trabalhos [MÖL04] [BLO04] [WAL04] já utilizaram com sucesso as bus macros como componente de interconexão de áreas reconfiguráveis utilizando o fluxo de Projeto Modular, proposto pela Xilinx.

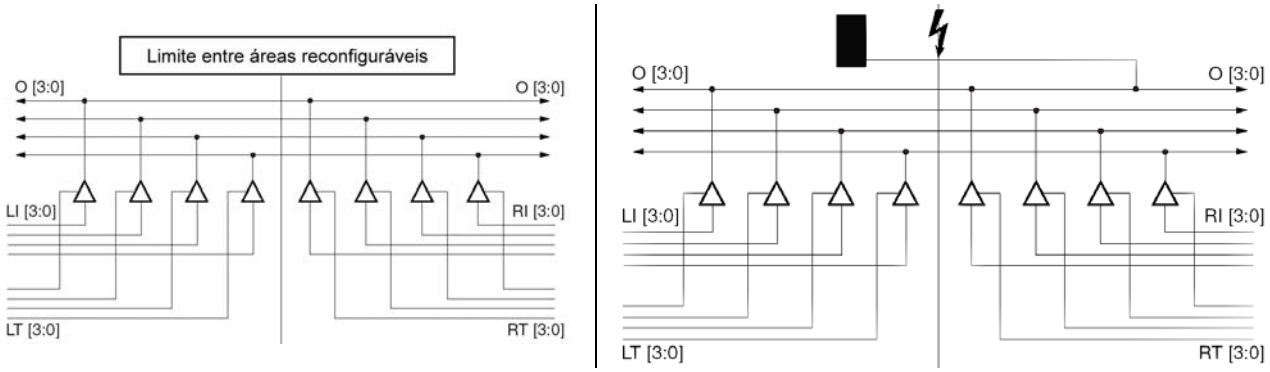


Figura 5 – (a) implementação física da bus macro; (b) problema reportado por Huebner de cruzamento de áreas reconfiguráveis sem a passagem por macros.

A bus macro possui quatro saídas (O) bidirecionais. Cada sinal de saída pode estar ligado na entrada esquerda (LI) ou na entrada direita (RI). Desta forma, apenas o *toggle* esquerdo (LT) ou o *toggle* direito (RT) de um sinal de saída podem estar ativados simultaneamente, de forma a não causar um curto circuito. É importante ressaltar que os *tristates* são ativos em nível lógico baixo e que para desativar uma determinada saída da bus macro basta colocar LT e RT em nível lógico alto.

Um ponto negativo da utilização de bus macros implementadas com *tristates* é que estes recursos são escassos nos FPGAs Virtex-II, existindo apenas dois por CLB. Além disto, Huebner [HUB04b] aponta que a ferramenta de roteamento algumas vezes não respeita as restrições feitas pelo projetista e cruza o limite entre as áreas reconfiguráveis na implementação de bus macros com *tristates*, conforme apresentado na Figura 5(b). O principal motivo para a ocorrência de erros é o fato da linha de roteamento do sinal de saída ser fisicamente a mesma em ambos os lados da macro. O retângulo preto é um componente do núcleo esquerdo. A ferramenta de roteamento conectou o sinal indicado utilizando um fio conectado ao lado direito, pelo fato do sinal ser equivalente em ambos os lados. A conseqüência deste ato é o mau funcionamento do sistema ou até um curto circuito que pode danificar o FPGA.

Devido aos problemas citados outras alternativas de macros foram propostas por Huebner [HUB04b] e Dyer [DYE02].

2.3.3 Restringir posicionamento e verificar roteamento

O passo seguinte no fluxo de implementação de SDRs é a *restrição da área física* dos núcleos reconfiguráveis. Nesta fase é importante fixar, em cada um dos projetos do sistema, os núcleos reconfiguráveis dentro das áreas reconfiguráveis. Deve-se tomar cuidado em restringir tanto os recursos lógicos que foram alocados para cada núcleo quanto os fios internos aos núcleos, caso contrário um ou mais fios poderão ser incorretamente conectados no momento da reconfiguração, podendo causar o não funcionamento do projeto, ou até um dano físico no dispositivo. A ferramenta Floorplanner da Xilinx auxilia no processo de restrição do posicionamento dos recursos lógicos dos núcleos. Infelizmente, ao restringir o núcleo a uma determinada posição, não obrigatoriamente os fios que interconectam os componentes internos do núcleo estarão posicionados dentro dos limites desta mesma área, acontecendo com freqüência que estes invadam áreas de outros núcleos. Parâmetros devem ser passados para as ferramentas de síntese física para forçar os fios a ficarem em uma área definida. A ferramenta FPGA Editor da Xilinx permite visualizar os fios do sistema e rotear manualmente ou automaticamente conexões que apresentem erros.

2.3.4 Gerar bitstreams parciais

O quarto passo para a geração de SDRs é a geração dos *bitstreams parciais* de cada um dos núcleos reconfiguráveis do sistema, sendo este um processo específico até mesmo para dispositivos de uma mesma família de um mesmo fabricante. Com freqüência, apenas o próprio fabricante cria ferramentas para a geração de bitstreams parciais, pois para efetuar este passo é necessário conhecer em profundidade a organização do bitstream e sua relação com a arquitetura de reconfiguração do dispositivo. Como os fabricantes não têm o interesse de divulgar detalhes sobre estas arquiteturas, apenas eles possuem informações suficientes para prover ferramentas capazes de configurar parcialmente os seus dispositivos. Outro motivo que inibe os fabricantes de dispositivos reconfiguráveis de fornecer mais informações de suas arquiteturas é que seja feita engenharia reversa de algum projeto, podendo assim colocar em risco a propriedade intelectual de produtos dos clientes.

A ferramenta de geração de bitstreams tanto parciais quanto totais da Xilinx é o BitGen. Outra alternativa disponibilizada pela Xilinx é o JBits, uma API da linguagem Java. Esta acessa o bitstream para a modificação e criação de um bitstream parcial a partir de um bitstream já existente. A ferramenta CoreUnifier [MÖL03], é uma ferramenta construída pelo autor deste trabalho para gerar bitstreams parciais com uma abordagem diferente dos métodos adotados pelo fabricante.

2.3.5 Realocar núcleos do sistema

Existe ainda um passo opcional para a implementação de SDRs que é a relativização do posicionamento dos núcleos reconfiguráveis do sistema, que neste trabalho é chamada de realocação. A realocação pode ser útil quando a sugestão do passo 2 de gerar um bitstream para cada núcleo em determinada área reconfigurável não for seguida. Alguns motivos do não cumprimento desta sugestão podem ser o elevado número de combinações a serem sintetizadas ou a dificuldade de restringir um determinado núcleo em uma determinada área reconfigurável. Neste caso existe a alternativa de se utilizar uma ferramenta de realocação de núcleos. Até o presente momento não se conhece uma ferramenta que permita realocar núcleos para o dispositivo Virtex-II.

Assim como sistemas microprocessados alocam e desalocam memória segundo a demanda do sistema operacional, SDRs configuram e reconfiguram o hardware conforme a exigência das aplicações que estão sendo executadas sobre ele. Por este motivo SDRs sofrem de problemas semelhantes, tais como fragmentação interna e externa. É denominada fragmentação interna a lógica configurável não utilizada dentro da área reservada a um núcleo de hardware. Fragmentação externa ocorre quando existe lógica configurável livre suficiente para um núcleo de hardware, entretanto esta lógica se encontra particionada, impossibilitando a configuração do núcleo.

Uma alternativa para contornar o problema de fragmentação em SDRs é utilizar a reconfiguração parcial e dinâmica para realocar a posição dos núcleos de hardware do sistema e liberar espaço para que outros núcleos sejam configurados. Outro motivo para realocar núcleos de hardware, ao invés de sintetizá-los novamente para cada nova posição do dispositivo, é que os núcleos podem possuir restrições temporais que só podem ser obtidas depois de longas etapas de posicionamento e roteamento. Neste caso, o núcleo é sintetizado uma única vez, sendo posteriormente realocado para alguma posição do sistema, com a vantagem de manter as suas características temporais. Núcleos que possuem alta taxa de comunicação com núcleos distantes do

sistema podem ser realocados para ocuparem posições próximas, reduzindo o congestionamento do meio de comunicação, aumentando a eficiência do sistema como um todo.

Embora realocação seja uma boa alternativa para solucionar o problema de fragmentação do hardware, a implementação desta técnica envolve o tratamento de uma série de questões para que ela efetivamente possa ser utilizada em SDRs: (*i*) como descobrir se o sistema está fragmentado? (*ii*) se estiver fragmentado, como salvar o estado atual do núcleo para continuá-lo do ponto onde havia parado quando for posicionado em uma nova posição? (*iii*) como descobrir uma nova posição para realocar o núcleo? (*iv*) quem será responsável por gerar um novo bitstream parcial com o núcleo realocado? um computador hospedeiro, um processador do sistema ou um núcleo dedicado do sistema? (*v*) se for um processador do sistema ou um núcleo dedicado do sistema, como ativar a reconfiguração de dentro do próprio sistema para que o núcleo realocado reinicie a sua execução? (*vi*) se nem mesmo realocando outros núcleos existir espaço suficiente para um núcleo que deve entrar em execução no sistema, como escolher um núcleo do sistema a ser retirado? (*vii*) a forma do núcleo a ser realocado pode ser modificada?

A resposta a estas questões tem motivado o estudo em quatro diferentes linhas de pesquisa: (*i*) controladores de configuração; (*ii*) sistemas auto-reconfiguráveis; (*iii*) escalonamento de núcleos de hardware; (*iv*) posicionamento de núcleos. Englobando uma ou mais destas características, diversos autores justificam o trabalho na área de Sistemas Operacionais para Hardware Reconfigurável (RHOS – *Reconfigurable Hardware Operating Systems*) [WAL03] [NOL03] [MER98].

De tantas questões a serem tratadas sobre este assunto, esta seção apresenta apenas uma introdução sobre realocação e fatores relacionados à configuração parcial e dinâmica de núcleos realocados.

Para que um SDR suporte a realocação de núcleos é necessário que as interfaces dos núcleos possuam a mesma semântica que os componentes isoladores do sistema. Assim os núcleos podem ser realocados e a comunicação com o restante do sistema permanece inalterada. Por este motivo é importante criar uma interface padrão entre os núcleos de forma que todos utilizem o mesmo protocolo de comunicação.

A tendência do projetista é construir um SDR que se adapte aos núcleos que o mesmo deve conectar no sistema. Neste caso o meio de comunicação deve ocupar uma posição fixa no SoC, possuir a sua interface de comunicação com os núcleos bem definida e reservar espaço para que os núcleos sejam inseridos no sistema sem sobrepor o meio de comunicação. Quanto mais interfaces de comunicação forem adicionadas ao sistema, maior será o número de núcleos que podem executar simultaneamente.

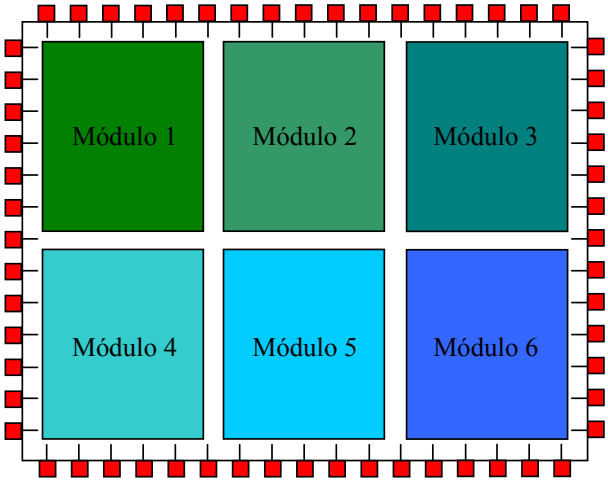


Figura 6 – Sistema hipotético que mapeia todos os pinos de entrada e saída, e que no entanto necessita que os núcleos sejam gerados com interface de comunicação na parte superior e inferior dos mesmos.

Outra questão importante em um SDR que permite realocação é que os núcleos devem possuir a interface padrão posicionada em uma determinada borda do núcleo. Com freqüência, a localização do meio de comunicação dita a posição das interfaces dos núcleos. A Figura 6 apresenta um SDR hipotético que utiliza bem a área do dispositivo e consegue mapear todos os pinos de entrada e saída, no entanto necessita que um mesmo núcleo seja sintetizado tanto com a interface na borda superior do núcleo quanto na borda inferior.

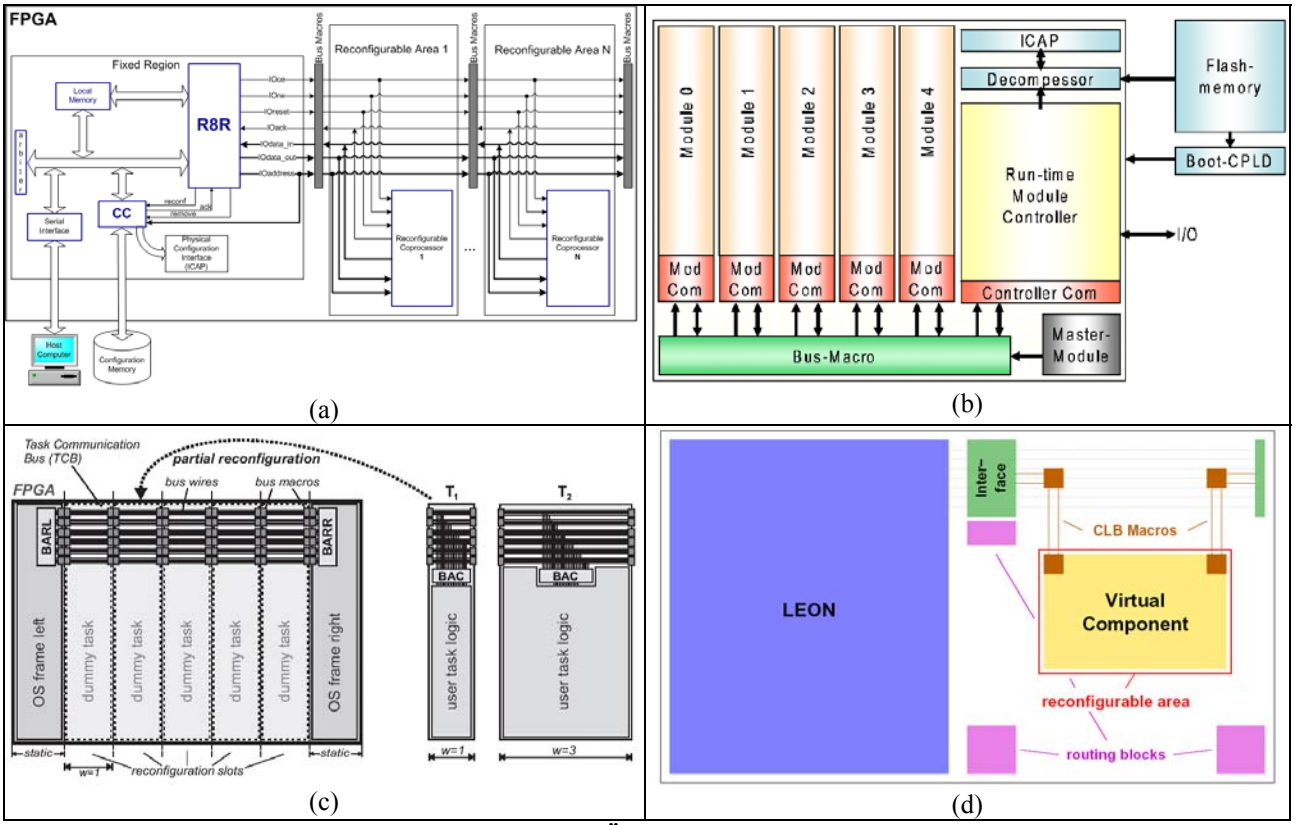


Figura 7 – (a) sistema de Möller [MÖL04]; (b) sistema de Huebner [HUB04a]; (c) sistema de Walder [WAL04]; (d) sistema de Dyer [DYE02].

A Figura 7 apresenta diagramas de blocos de SDRs que foram implementados sobre dispositivos reconfiguráveis da Xilinx. Estes sistemas ainda apresentam uma organização de

posicionamento de núcleos reconfiguráveis relativamente simples, pois estão restritas a reconfiguração em colunas do dispositivo reconfigurável, conforme visto na Seção 2.2. O fato de toda a coluna ser alterada durante a reconfiguração inibe a inserção de mais de um núcleo reconfigurável na mesma coluna, pois a reconfiguração de um núcleo pode interromper o funcionamento do outro.

2.4 NoCs

Sob o ponto de vista de implementação, a estrutura de interconexão de núcleos de hardware por barramento é a mais simples. No entanto, este meio de comunicação apresenta diversas desvantagens [BEN02]: (*i*) apenas uma troca de dados pode ser realizada por vez, pois o meio físico é compartilhado por todos os núcleos de hardware, reduzindo o desempenho global do sistema; (*ii*) necessidade de mecanismos inteligentes de arbitragem do meio físico para evitar desperdício de largura de banda; (*iii*) a escalabilidade é limitada, ou seja, o número de núcleos de hardware que podem ser conectados ao barramento é muito baixo, tipicamente na ordem da dezena; (*iv*) o uso de linhas globais em um circuito integrado com tecnologia submicrônica impõe sérias restrições ao desempenho do sistema devido às altas capacitâncias e resistências parasitas inerentes aos longos fios. Estas desvantagens podem ser parcialmente contornadas através do uso de, por exemplo, hierarquia de barramentos, onde o problema continua existindo, sendo apenas minimizado.

Uma maneira de solucionar os problemas oriundos da arquitetura de barramentos é através da utilização de redes de comunicação mais complexas internamente ao circuito integrado [DAL01] [WIN01], no que se denomina hoje de NoC – *Network on Chip*. Em NoCs, as informações trocadas pelos nodos fonte e destino de uma comunicação são organizadas sob a forma de mensagens. Tipicamente, as mensagens são quebradas em pacotes para transmissão. Um pacote é a menor unidade de informação que contém detalhes sobre o roteamento e seqüenciamento dos dados e possui três partes: um cabeçalho, um corpo de dados e um terminador, sendo que o cabeçalho e o terminador formam um envelope ao redor do corpo de dados do pacote. No cabeçalho são incluídas informações de roteamento e controle utilizadas pelos nodos de chaveamento para propagar os pacotes em direção ao nodo destino da comunicação. O terminador, por sua vez, inclui informações utilizadas para a detecção de erros e para a sinalização do fim da mensagem. Um pacote é constituído por uma seqüência de *flits* (menor unidade de transferência de dados), cuja largura depende da largura física do canal.

Duas partes compõem uma rede intra-chip: os *serviços* e o *sistema de comunicação*. Rijpkema [RIJ01] descreve como exemplos de *serviços* as garantias de integridade de dados, *throughput* mínimos e latência máxima. A implementação destes *serviços* é normalmente baseada em uma camada de protocolos como proposto no modelo de referência OSI. Neste modelo, cada nível oferece um conjunto de serviços ao nível superior, usando funções realizadas no próprio nível e serviços disponíveis nos níveis inferiores. No caso de NoCs freqüentemente apenas os níveis inferiores são abordados, ficando a cargo das aplicações a implementação dos níveis superiores de forma *ad hoc*. Algumas funções dos quatro níveis inferiores são descritas a seguir.

O *nível físico* é responsável por fornecer as especificações dos meios mecânico e elétrico para conectar diferentes entidades a nível de bit [DAY83]. No presente trabalho, este nível corresponde à comunicação entre os roteadores, como exemplificado na Figura 8. A largura do barramento de dados deve ser escolhida em função dos recursos de roteamento disponíveis e memória disponível

para a implementação de esquemas de buferização.

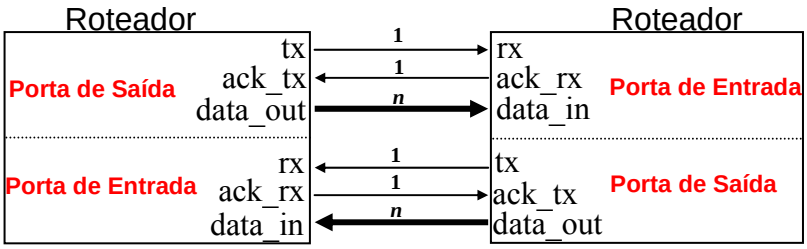


Figura 8 – Interface entre roteadores.

O *nível de enlace* tem o objetivo de estabelecer a conexão lógica entre entidades e de converter um meio não confiável em confiável. Para cumprir essas exigências, técnicas de controle de fluxo e detecção de erro são geralmente utilizadas. Este trabalho implementa no nível de enlace um protocolo simples de handshake construído sobre o nível físico, para tratar do controle de fluxo e do correto envio e recepção de dados. Neste protocolo, quando um roteador necessita enviar dados para um roteador vizinho, este coloca o dado no sinal *data_out* e ativa o sinal *tx*. Uma vez que o roteador vizinho armazena o dado do sinal *data_in*, este ativa o sinal *ack_rx*, e a transmissão está completa.

O *nível de rede* envolve-se com a troca de pacotes. Este nível é responsável pela segmentação e remontagem dos flits, roteamento ponto-a-ponto entre roteadores, e controle de congestionamento. O nível de rede neste trabalho implementa a técnica de chaveamento por pacotes.

O *nível de transporte* é responsável por estabelecer a comunicação fim-a-fim entre origem e destino. Serviços como segmentação e remontagem de pacotes são essenciais para prover comunicação confiável [DUA02]. Aqui, a comunicação fim-a-fim é implementada nos núcleos locais.

O *sistema de comunicação* permite que qualquer núcleo origem envie pacotes para qualquer núcleo destino conectado à NoC. A estrutura da NoC é um conjunto de roteadores interconectados por canais de comunicação. A forma como os roteadores estão conectados define a topologia da rede. De acordo com o padrão de interconexão de IPs e elementos de roteamento, as redes podem ser classificadas em uma de duas principais classes: *estáticas* ou *dinâmicas*. Hipercubo, anel, malha, torus e fat-tree são exemplos de redes utilizadas para implementar redes estáticas. Barramentos e crossbar são exemplos de redes dinâmicas.

Em função da topologia da rede são definidos o *mecanismo de comunicação*, o *modo de chaveamento* e o *algoritmo de roteamento*, que compõem os serviços providos pela NoC.

O *mecanismo de comunicação* especifica como as mensagens trafegam pela rede. Dois métodos para transferência de mensagens são *chaveamento por circuito* e *chaveamento por pacotes* [HWA93]. No *chaveamento por circuito*, um caminho é estabelecido pela alocação de canais entre a origem e o destino antes dos pacotes serem enviados. Este caminho é chamado de conexão. Depois de estabelecida a conexão, os pacotes podem ser enviados, e qualquer outra comunicação que tentar utilizar os canais alocados será negada, até que um processo de desconexão seja executado. No *chaveamento por pacotes*, pacotes são transmitidos sem necessidade de estabelecimento prévio de uma conexão.

Chaveamento por pacotes requer o uso de um *modo de chaveamento*, o qual define como os pacotes se movem através dos roteadores. Os modos mais empregados são *store-and-forward*, *virtual cut-through* e *wormhole* [NI93]. No modo *store-and-forward*, um roteador não pode repassar um pacote adiante até que este tenha sido completamente recebido. Cada vez que um roteador recebe um pacote, seu conteúdo é examinado para decidir o que fazer, implicando em uma

latência por roteador. No modo *virtual cut-through*, um roteador pode enviar um pacote adiante assim que o roteador seguinte der uma garantia que o pacote será aceito completamente [RIJ01]. Logo, é necessária uma fila para armazenar pelo menos um pacote completo, como no *store-and-forward*, mas neste caso a latência de comunicação é menor. O modo de chaveamento *wormhole* é uma variação do modo *virtual cut-through*, que evita a necessidade de uso de filas de grande dimensões. Neste modo, apenas o(s) flit(s) de cabeçalho contém as informações de roteamento e os flits restantes que compõem o pacote seguem o mesmo caminho reservado pelo cabeçalho.

O *algoritmo de roteamento* define o caminho percorrido por um pacote entre a origem e o destino. Dependendo de onde as decisões de roteamento são realizadas é possível classificar estes como *origem* ou *distribuído*. Quando o roteamento é efetuado na *origem*, o caminho inteiro é decidido antes da mensagem ser enviada. Quando o roteamento é *distribuído* cada roteador recebe um pacote e decide em que direção enviá-lo. Dependendo de como um caminho é definido para transmitir os pacotes, o roteamento pode ser classificado como *determinístico* ou *adaptativo*. No roteamento *determinístico*, o caminho é definido exclusivamente pelos endereços origem e destino. No roteamento *adaptativo*, o caminho é definido em função do tráfego da rede [DUA02] [NI93]. Esta última classificação pode ainda ser subdividida em *parcialmente* e *totalmente* adaptativos. No roteamento *parcialmente* adaptativo, apenas um subconjunto dos caminhos físicos disponíveis é alocado para a comunicação. No roteamento *totalmente* adaptativo, qualquer caminho físico da rede pode ser utilizado para enviar uma dada mensagem.

A Tabela 7, obtida de [MOR04], apresenta o estado da arte no desenvolvimento de redes intra-chip. O objetivo de apresentar esta Tabela é ilustrar ao leitor a diversidade de trabalhos em NoCs, e também mostrar que este é um tema recente de pesquisa, dado que as primeiras publicações datam de 2000.

Tabela 7 - Estado da arte em NoCs.

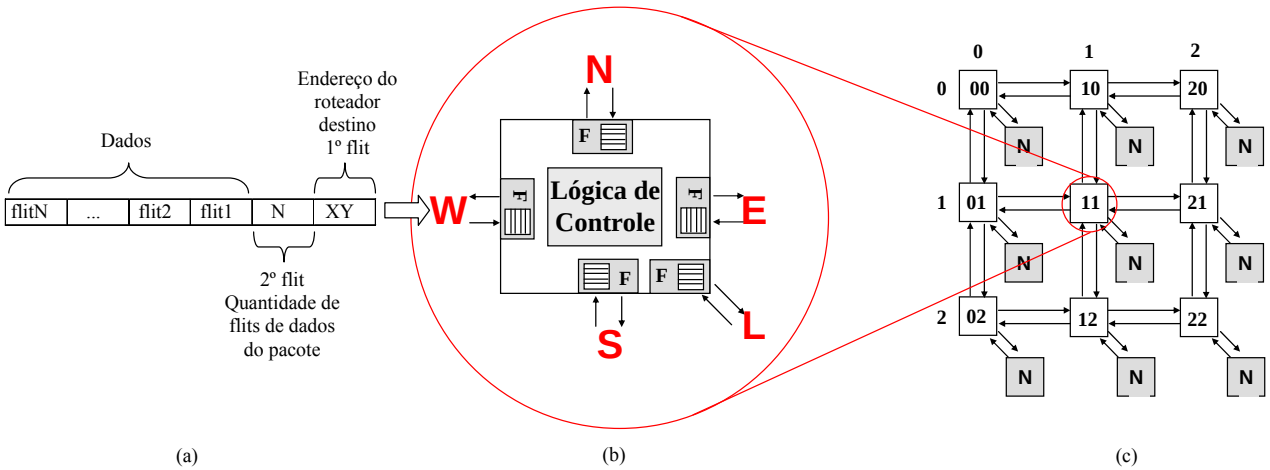
ND ou células hachuradas = dados não disponíveis. TG= throughput garantido.

NoC	Topologia / Roteamento	Tamanho do Flit	Buffer	Interface roteador núcleo	Área do roteador	Desempenho estimado de pico	Suporte a QoS	Implementação
***SPIN* - 2000** [GUE00] [AND03a] [AND03b]	Fat-tree / Deterministico e adaptivo	32 bits dados + 4 bits controle	Fila na entrada + 2 filas compartilhadas na saída	VCI	0,24 mm^2 CMOS 0,13μm	2 Gbits/s por roteador		Leiaute ASIC 4,6 mm^2 CMOS 0,13μm
***aSOC* - 2000** [LIA00]	Malha / Determinado pela aplicação	32 bits	Nenhum		50K transistores		Chaveamento por circuito	Leiaute ASIC CMOS 0,35μm
Dally - 2001 [DAL01]	Torus dobrado / XY na origem	256 bits dados + 38 bits controle	Fila na entrada		0,59 mm^2 CMOS 0,1μm	4 Gbits/s por fio	TG – canais virtuais	Não
***Nostrum* - 2001** [KUM02] [MIL04]	Malha / Batata Quente	128 bits dados + 10 bits controle	Fila na entrada e na saída		0,01 mm^2 CMOS 65nm			
Sgroi - 2001 [SGR01]	Malha / ND	18 bits dados + 2 bits controle		OCP				
***Octagon* - 2001** [KAR01] [KAR02]	Anel cordal / Distribuído e adaptativo	Dados Variáveis + 3 bits controle				40 Gbits/s	Chaveamento por circuito	Não
Marescaux - 2002 [MAR02] [MAR03]	Torus / XY bloqueante, baseado em hops, determínistico	16 bits dados + 3 bits controle	Fila na entrada	Personalizada	611 slices Virtex-II (6,58% de ocupação da XC2V6000)	320Mbits/s por canal virtual a 40 MHz	2 canais virtuais (para evitar deadlock)	FPGA Virtex-II / Virtex-II Pro

Bartic - 2003 [BAR03]	Arbitrário (links parametrizáveis) / Determinístico, virtual-cut-through	Dados Variáveis + 2 bits controle	Fila na saída	Persona-lizada	552 slices + 5 BRAMs Virtex-II Pro para um roteador bidirecional de 5 portas	800Mbits/s por canal para flits de 16 bits a 50 MHz	Taxa de controle de fluxo, controle de congestio-namento	FPGA Virtex-II Pro
***Æthereal* - 2002** [RIJ01] [RIJ03]	Malha / na origem	32 bits	Fila na entrada	DTL (Formato Philips)	0,26 mm^2 CMOS 0,12μm	80Gbits/s por roteador	Chavea-mento por circuito	Leiaute ASIC
***Eclipse* - 2002** [FOR02]	Malha Esparsa Hierárquica / ND	68 bits	Fila na saída					Não
***Proteo* - 2002** [SAA02] [SAA03] [SIG02]	Bidirecional ring / ND	Tamanho de dados e controle variáveis	Fila na entrada e na saída	VCI				Leiaute ASIC CMOS 0,18μm
SOCIN - 2002 [ZEF03]	Malha / XY na origem	n bits dados + 4 bits controle	Fila na entrada parametrizável	VCI	420 LCs APEX FPGAs (Estimado para n=8, sem fila)	1 Gbits/s por roteador a 25 MHz	Não	Não
***SoCBus* - 2002** [WIK03]	Malha / XY adaptativa	16 bits dados + 3 bits controle	Fila de uma posição na entrada e na saída	Persona-lizada		2,4 Gbits/s por roteador	Chavea-mento por circuito	Não
QNoC - 2003 [BOL04]	Malha regular ou irregular / XY	16 bits dados + 10 bits controle (parametrizá vel)	Fila na entrada parametrizável + Fila na saída (uma posição)	Persona-lizada	0,02 mm^2 CMOS 90nm (Estimado)	80 Gbits/s por roteador para flits de 16 bits a 1GHz	TG – canais virtuais (4 tráfegos diferentes)	Não
T-SoC - 2003 [GRE04] [PAN03]	Fat-tree / Adaptiva	Máximo 38 bits	Fila na entrada e na saída	Persona-lizada / OCP	27K a 36K portas lógicas de 2 entradas equivalentes		TG - 4 canais virtuais	
Xpipes - 2002 [DAL03]	Arbitrária (tempo de projeto) / Estática na origem	32, 64 ou 128 bits	Fila de saída virtual	OCP	0,33 mm^2 CMOS 100nm (Estimado)	64 Gbits/s por roteador para flits de 32 bits a 500MHz	Não	Não
***Hermes* – 2003** [MOR03]	Malha / XY	8 bits dados + 2 bits controle (parametrizá vel)	Fila na entrada parametrizável	Persona-lizada / OCP	555 LUTs 278 slices Virtex-II	500 Mbits/s por roteador a 25 MHz	Não	FPGA Virtex-II

2.4.1 NoC Hermes

A topologia de uma NoC é definida pela estrutura de conexão dos seus roteadores. Na topologia malha utilizada neste trabalho, roteadores distintos podem possuir um número de portas diferentes, dependendo de sua posição na rede, como mostrado na Figura 9(c). Por exemplo, o roteador central de uma NoC Hermes 3x3 possui cinco portas, conforme apresentado na Figura 9(b). As cinco 5 portas bidirecionais do roteador são: East, West, North, South e Local. Cada porta possui uma fila para o armazenamento temporário de flits. A porta Local estabelece a comunicação entre o roteador e seu núcleo local. As demais portas ligam o roteador aos roteadores vizinhos. O modo de chaveamento adotado pelo roteador da NoC Hermes é wormhole, onde cada pacote que trafega pela rede é passado flit a flit pelos canais físicos. O formato dos pacotes que trafegam pela rede é apresentado na Figura 9(a). A lógica de controle implementa a *lógica de arbitragem* e o *algoritmo de chaveamento por pacotes*.



(a) (b) (c)

Figura 9 – (a) formato dos pacotes que trafegam pelos roteadores; (b) arquitetura simplificada do roteador; (c) exemplo da NoC Hermes 3x3. Em uma NoC 3x3, apenas o roteador do meio (11) possui as cinco portas.

Um roteador pode ser requisitado a estabelecer até 5 conexões simultâneas (uma conexão por porta de entrada do roteador). A *lógica de arbitragem* é utilizada para escolher qual das portas de entrada será roteada primeiro. Para isto, é utilizado um algoritmo de arbitragem dinâmica rotativa, onde a prioridade de uma determinada porta é dada em função da última porta a ter a requisição atendida. Este algoritmo previne que uma mesma porta de entrada tenha sempre uma porta de saída alocada a ela, não permitindo acesso das outras portas de entrada às portas de saída.

O *algoritmo de chaveamento por pacotes* utilizado pelo roteador é o XY determinístico, no qual os pacotes trafegam pela rede sempre em uma mesma ordem: (*i*) trafegam pela rede no eixo X até chegar na coluna do roteador destino; (*ii*) trafegam pela rede no eixo Y até chegar na linha do roteador destino; (*iii*) estando no roteador destino, o pacote é roteado para a porta local para alcançar o núcleo destino.

Quando um flit é bloqueado em um determinado roteador, o desempenho da rede é afetado, pois os flits do mesmo pacote são bloqueados em outros roteadores intermediários da rede. Para diminuir esta perda de desempenho, uma fila circular é adicionada a cada porta de entrada do roteador, reduzindo o número de roteadores afetados pelos flits bloqueados. O tamanho da fila é parametrizável em tempo de projeto. Mais detalhes sobre a NoC Hermes podem ser obtidos em [MOR04].

3 CRONOGRAMA E ATIVIDADES

As tarefas previstas para o trabalho a ser desenvolvido no segundo ano do Mestrado são apresentadas na Tabela 8. Estas atividades compreendem duas macro-atividades:

- suporte de software: atividades 1, 2, 4 e 9;
- definição e prototipação da arquitetura do SDR que emprega NoC como meio de comunicação: atividades 5 e 6.

Tabela 8 – Cronograma de atividades.

Atv.	Objetivo	Jan		Fev		Mar		Abr		Mai		Jun		Jul		Ago		Set		Out		Nov		Dez	
1	*CoreUnifier-II*																								
2	*Macro Möller*																								
3	*Escrita Artigo*																								
4	*XDLAnalyzer*																								
5	*Modificações na NoC*																								
6	*Integr. Macro/NoC*																								
7	*Seminário Andamento*																								
8	*Escrita Artigo*																								
9	*Realocação*																								
10	*Escrita Artigo*																								
11	*Escrita Volume*																								

Atv. 1 – Ferramenta CoreUnifier-II

A ferramenta CoreUnifier-II, a ser desenvolvida, tem por objetivo gerar bitstreams parciais a partir da seleção de áreas de bitstreams totais de dispositivos Virtex-II. Um dos motivos para a criação desta ferramenta é que o fluxo sugerido pela Xilinx para a criação de bitstreams parciais, fluxo de Projeto Modular, reporta erros de difícil compreensão e correção, inviabilizando a utilização deste fluxo para projetos de grande porte. O anexo B de [BRI04] apresenta alguns erros reportados pelo fluxo de Projeto Modular. Outro motivo para a criação desta ferramenta é a posterior investigação de realocação de núcleos.

A elaboração desta ferramenta envolve entender a organização de bitstreams do dispositivo Virtex-II (material este não existente na literatura), implementar o algoritmo que calcula o CRC do bitstream específico para este dispositivo e descobrir o protocolo para reconfiguração parcial que de fato funcione no FPGA. Estas tarefas são facilitadas pelo conhecimento da organização dos dispositivos da família anterior, utilização das classes JBits e um acordo de cooperação de pesquisa assinado recentemente com a empresa Xilinx que poderá prover documentos com maiores informações sobre o dispositivo e o seu bitstream.

Resultado esperado: ferramenta CoreUnifier-II funcionando para a geração de bitstreams parciais dos núcleos do SDR.

Atv. 2 – Macro proposta por Möller

Conforme apresentado na Seção 2.3.2, uma macro possui a função de isolar a área que está sendo reconfigurada e prover uma interface de comunicação entre esta e o restante do sistema. Esta atividade apresenta uma nova proposta de macro para substituir a bus macro sugerida pela Xilinx. A

bus macro, como visto na Seção 2.3.2, possui dois problemas: (*i*) é constituída de *tristates*, um componente disponível em quantidade muito limitada nos dispositivos Xilinx e (*ii*) possui problemas de roteamento, conforme apresentado na Figura 5(b). A macro proposta nesta atividade utilizará sinais unidirecionais, solucionando estes problemas, ao custo de eventual gasto adicional de recursos.

Resultado esperado: circuito de média complexidade, prototipado, utilizando a ferramenta CoreUnifier-II e as macros.

Atv. 3 – Escrita de artigo

É planejado neste ponto a escrita de um artigo que apresente a ferramenta desenvolvida na atividade 1 e a macro de Möller proposta na atividade 2. Uma aplicação simples mostrando a execução do fluxo com a inserção da macro proposta será utilizada como estudo de caso.

Atv. 4 – XDLAnalyzer

As versões atuais das ferramentas da Xilinx não permitem uma modificação automatizada das conexões físicas efetuadas entre os componentes do dispositivo. No entanto, existe uma ferramenta pouco documentada e até hoje não utilizada no grupo GAPH, denominada XDL da Xilinx. Esta ferramenta permite transformar um arquivo em formato proprietário para formato texto e vice-versa.

Planeja-se nesta atividade estudar o funcionamento da ferramenta XDL e criar a ferramenta XDLAnalyzer que analisa as conexões do sistema em formato texto. O XDLAnalyzer será implementado para auxiliar na tarefa de verificação do roteamento efetuado pelo sistema quando o mesmo não funcionar no dispositivo.

Resultado esperado: ferramenta XDLAnalyzer funcionando para a verificação de roteamento entre os núcleos reconfiguráveis e a macro proposta.

Atv. 5 – Modificações no roteador da NoC Hermes

A NoC Hermes desenvolvida inicialmente por Mello e o Autor [MEL03] e parametrizável pela ferramenta MAIA [OST05] será utilizada como meio de comunicação dos núcleos reconfiguráveis. Optou-se por NoCs como meio de comunicação dos núcleos do SDR por esta possuir alta escalabilidade quando comparada a barramentos, e ser apontada como arquitetura de comunicação que será utilizada em futuros SoCs [HEN03]. A NoC Hermes será utilizada porque seu código fonte já é conhecido pelo Autor e porque esta NoC pode ser parametrizada para ocupar pouca área. O baixo consumo de área da NoC é desejado para que um maior número de núcleos sejam colocados no FPGA, assim aumentando o paralelismo do sistema a ser verificado.

Para que a Hermes possa ser utilizada como meio de comunicação do SDR será necessário modificar os roteadores atuais. Uma alteração prevista é com relação aos sinais de entrada das portas locais do roteador aos quais os núcleos reconfiguráveis serão conectados. Será necessário que o roteador desconsidere dados que sejam recebidos quando a reconfiguração do núcleo conectado ao roteador estiver em andamento. O motivo disto é que modificações nas conexões do núcleo estarão sendo feitas, causando transições nos dados enviados ao roteador e conseqüentemente levando este a um estado inválido.

Não pertence ao escopo deste trabalho implementar o controlador de configurações para efetuar a reconfiguração. A dissertação de mestrado do aluno Rafael Soares, em andamento paralelo

a esta, apresenta a implementação de um controlador de configurações em software. Um controlador configurações em hardware já foi desenvolvido na dissertação de Ewerson Carvalho [CAR04]. No presente trabalho a reconfiguração será feita manualmente pelo próprio usuário. A seqüência de operações a ser utilizada no momento em que um novo núcleo for necessário é: (*i*) enviar uma mensagem específica para o roteador do núcleo a ser reconfigurado para isolar a comunicação com o núcleo; (*ii*) efetuar a reconfiguração a partir do software Impact; (*iii*) liberar a comunicação do roteador isolado no passo (*i*) com o núcleo conectado na porta local.

Resultado esperado: roteador da NoC capturando mensagens específicas enviadas por um controlador de configurações, isolando/liberando a comunicação com o núcleo conectado a porta local no momento da reconfiguração.

Atv. 6 – Integração macro/NoC

A NoC Hermes com interface nativa se comunica com um núcleo através de seis sinais, conforme apresentado na Figura 8. Este comportamento não deve ser alterado quando a macro implementada na atividade 5 for posicionada entre o roteador e o núcleo, como ilustra a Figura 10. Nesta atividade será feita a verificação se as modificações efetuadas no roteador na atividade 5 foram suficientes para que o núcleo seja reconfigurado sem interromper o meio de comunicação, que continua operando. Como os efeitos da reconfiguração não podem ser previstos, esta verificação não pode ser facilmente simulada. Portanto, pretende-se validar o sistema no próprio dispositivo FPGA e utilizar o analisador lógico e/ou a ferramenta ChipScope da Xilinx para verificação do estado interno do FPGA durante a execução do sistema.

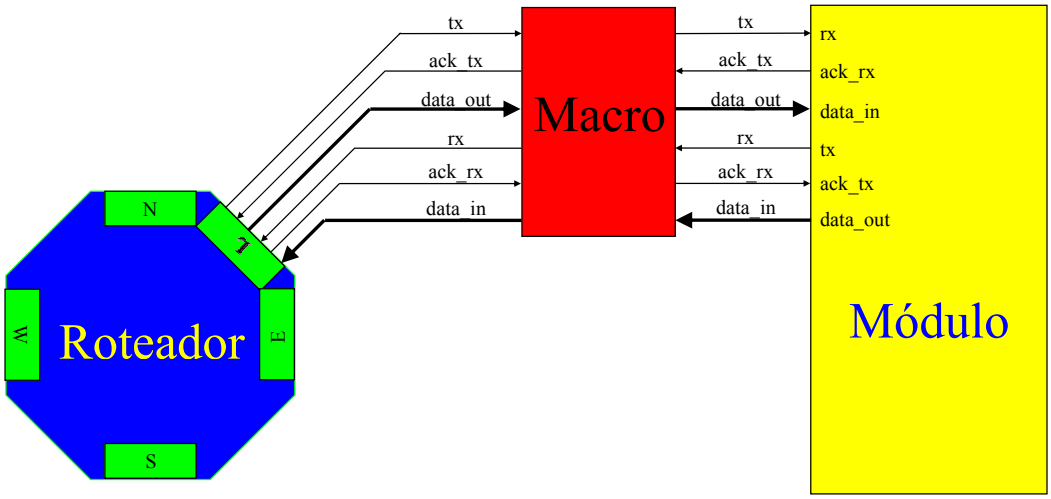


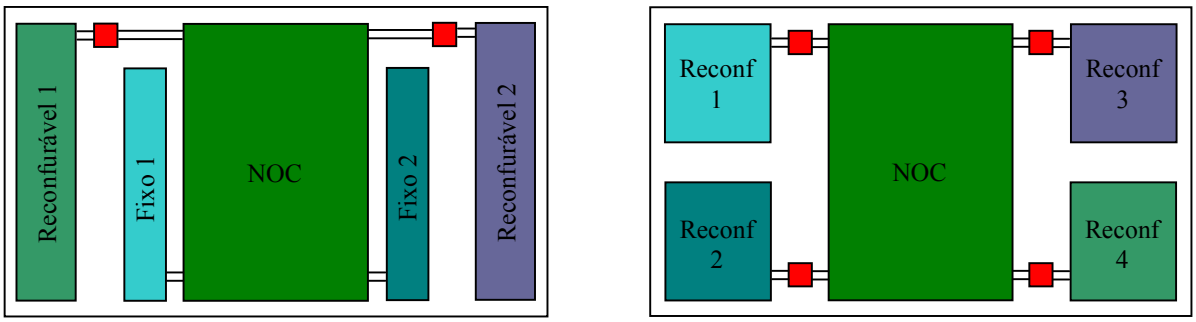
Figura 10 – Posicionamento da macro: entre o roteador e o núcleo reconfigurável.

A Figura 11(a) apresenta propostas de posicionamento das áreas reconfiguráveis no SDR a implementar. A Figura 11(b) apresenta um SDR opcional a ser desenvolvido. A dificuldade inerente ao desenvolvimento do SDR da Figura 11(b) é como salvar e recuperar o contexto do núcleo que não foi reconfigurado e que está posicionado na mesma coluna do núcleo reconfigurável, já que toda a coluna é modificada. Para solucionar esta dificuldade, tentar-se-á realizar um readback de toda a coluna antes da reconfiguração e gerar um novo bitstream que contenha tanto as linhas do núcleo a ser configurado quanto as linhas do núcleo em que o contexto foi salvo a partir do readback. O correto funcionamento deste sistema será verificado caso os núcleos reconfiguráveis funcionem corretamente após a reconfiguração.

Resultado esperado: NoC funcionando com duas áreas reconfiguráveis.

Atv. 7 – Seminário de andamento

No seminário de andamento será apresentado o andamento das atividades de 1 a 6 e a previsão para a execução das atividades 8 a 11.



(a) (b)

Figura 11 – (a) proposta de SDR com duas áreas reconfiguráveis; (b) proposta opcional de SDR com quatro áreas reconfiguráveis, sendo que existem duas áreas em uma mesma coluna.

Atv. 8 – Escrita de artigo

Será enfocado neste artigo as atividades 5 e 6, referentes às modificações feitas na NoC para que a mesma possua núcleos reconfiguráveis e a inserção das macros para especificar o ponto de conexão do núcleo reconfigurável com o restante do sistema.

Atv. 9 – Realocação

Conforme visto na Seção 2.3.5, existem diversas questões a serem respondidas sobre realocação. Por isso, restringe-se esta atividade apenas transladar um núcleo de uma posição para outra. As posições de origem e destino devem ser providas pelo usuário, assim como garantir que existe uma conexão do núcleo com o restante do sistema na posição onde ele for colocado.

Esta atividade implica estudar de forma mais aprofundada o bitstream, buscando encontrar quais os bits referentes aos recursos de um determinado CLB e como transladar CLBs horizontalmente e verticalmente. Estas funções de translação serão adicionadas à ferramenta CoreUnifier-II.

Propõe-se como tarefa opcional a implementação de um núcleo de hardware que faça a realocação. Este núcleo deve ler o *bitstream* parcial da memória de configurações, modificar o registrador do *bitstream* que contenha a coluna inicial do núcleo, recalcular o CRC e reconfigurar o sistema parcialmente pelo ICAP. As duas maiores dificuldades desta tarefa são implementar o algoritmo de cálculo de CRC dos dispositivos Virtex-II em hardware e o acesso por hardware ao componente ICAP.

A validação desta atividade será feita copiando um núcleo conectado a um roteador para um roteador diferente.

Resultado esperado: SDR funcionando após ter feito a reconfiguração com um núcleo realocado a partir da ferramenta CoreUnifier-II.

Atv. 10 – Escrita de artigo

Após verificado o funcionamento da realocação de núcleos para a família Virtex-II planeja-se escrever um artigo explicando a relação entre o bitstream e arquitetura do dispositivo.

Atv. 11 – Escrita do volume

A atividade de escrita do volume será feita em paralelo à escrita de artigos e à pesquisa de realocação.

4 REFERÊNCIAS BIBLIOGRÁFICAS

[AND03a] Andriahantenaina, A.; Greiner, A. "**Micro-network for SoC: Implementation of a 32-port SPIN network"**. In: Design Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 1128-1129.

[AND03b] Andriahantenaina, A.; Charlery, H.; Greiner, A.; Mortiez, L.; Zeferino C. "**SPIN: a Scalable, Packet Switched, On-chip Micro-network**". In: Design Automation and Test in Europe Conference and Exhibition (DATE), 2003, pp. 70-73.

[ATM04] Atmel Corporation. **http://www.atmel.com**. 2004.

[BAR03] Bartic, A.; Mignolet, J-Y.; Nollet, V.; Marescaux, T.; Mignolet, J-Y.; Verkest, D.; Vernalde, S.; Lauwereins, R. "**Highly Scalable Network on Chip for Reconfigurable Systems**". In: International Symposium on System-on-Chip (SoC), 2003, pp. 79-82.

[BEC00] Becker, J.; Piontek, T.; Glesner, M.; "**DReAM: A Dynamically Reconfigurable Architecture for Future Mobile Communications Applications**". In: Field Programable Logic and Applications (FPL), 2000, pp. 312-321.

[BEN02] Benini, L.; De Micheli, G. "**Networks on chips: a new SoC paradigm**". Computer, v. 35(1), 2002, pp. 70-78.

[BLO03] Blodget, B.; James-Roxby, P.; Keller, E.; McMillan, S.; Sundararajan, P. "**A Self-reconfiguring Platform**". In: Field Programmable Logic and Applications (FPL), 2003, pp. 565-574.

[BLO04] Blodget, B.; Bobda, C.; Huebner, M.; Niyonkuru, A. "**Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs**". In: Field Programmable Logic and Applications (FPL), 2004, pp. 801-810.

[BOL04] Bolotin, E.; Cidon, I.; Ginosar, R.; Kolodny, A. "**QNoC: QoS architecture and design process for Network on Chip**". The Journal of Systems Architecture, Special Issue on Networks on Chip (in preparation), 2004.

[BRI04] Brião, E. "**Reconfiguração Parcial e Dinâmica para Núcleos de Propriedade Intelectual**". Dissertação de mestrado, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brasil, 2004, 123 p.

[CAR04] Carvalho, E. "**RSCM – Controlador de Configurações para Sistemas de Hardware Reconfigurável**". Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2004, 127 p.

[DAL01] Dally, W.; Towles, B. "**Route packets, not wires: on-chip interconnection networks**". In: Design Automation Conference (DAC), 2001, pp. 684-689.

[DAL03] Dall'Osso, M.; Biccari, G.; Giovannini, L.; Bertozzi, D.; Benini, L. "**Xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multi-Processor SoCs"**. In: International Conference on Computer Design (ICCD), 2003, pp. 536-539.

[DAY83] Day, J.; Zimmermman, H. "**The OSI reference model**". Proceedings of IEEE, v. 71, 1983, pp. 1334-1340.

[DUA02] Duato, J.; Yalamanchili, S.; Ni, L. "**Interconnection Networks**". Morgan Kaufmann, 2002, 624 p.

[DYE02] Dyer, M.; Plessl, C.; Platzner, M. "**Partially Reconfigurable Cores for Xilinx Virtex**". In: Field Programmable Logic and Applications (FPL), 2002, pp. 292-301.

[EBE96] Ebeling, C.; Cronquist, D.; Franklin, P. "**RaPiD - Reconfigurable Pipelined Datapath**". In: Field Programmable Logic and Applications (FPL), 1996, pp. 126-135.

[FOR02] Forsell, M. "**A Scalable High-Performance Computing Solution for Networks on Chips"**. IEEE Micro, v. 22(5), 2002, pp. 46-55.

[GRE04] Grecu, C.; Pande, P.; Ivanov, A.; Saleh, R. **"A Scalable Communication-Centric SoC Interconnect Architecture"**. In: IEEE International Symposium on Quality Electronic Design (ISQED'2004), 2004, pp. 343-348.

[GUE00] Guerrier. P.; Greiner. A. "**A generic architecture for on-chip packet-switched interconections"**. In: Design Automation and Test in Europe (DATE), 2000, pp. 250-256.

[HAR95] Hartenstein, R.; Kress, R. "**A Datapath Synthesis System for the Reconfigurable Datapath Architecture**". In: Asia and South Pacific Design Automation Conference (ASP-DAC), 1995, pp. 479-484.

[HAR97] Hartenstein, R.; Becker, J.; Herz, M.; Nageldinger, U. "**An Innovative Platform for Embedded System Design**". In: Architekturen von Rechensystemen (ARCS), 1997, pp. 143-152.

[HAR01] Hartenstein, R. "**A Decade of Reconfigurable Computing: a Visionary Restrospective**". In: Design Automation and Test in Europe (DATE), 2001, pp. 642-649.

[HEN03] Henkel, J. "**Closing the SoC Design Gap**". Computer, v. 36(9), 2003, pp. 119-121.

[HER01] Herz, M. "**High Performance Memory Communication Architectures for Coarse-Grained Reconfigurable Computing Architectures**". Tese de doutorado, Universidade de Kaiserslautern, Alemanha, 2001.

[HUB04a] Huebner, M.; Ullmann, M.; Braun, L.; Klausmann, A.; Becker, J. "**Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems**". In: Field Programmable Logic and Applications (FPL), 2004, pp. 1037-1041.

[HUB04b] Huebner, M.; Becker, T.; Becker, J. "**Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration**". In: Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI), 2004, pp. 28-32.

[HWA93] Hwang, K. "**Advanced Computer Architecture: Parallelism, Scalability, Programmability**". New York : McGraw-Hill, 1993, pp. 213-256.

[KAR01] Karim, F.; Nguyen, A.; Dey, S.; Rao, R. **On-chip communication architecture for OC-768 network processors.** In: Design Automation Conference (DAC), 2001, pp. 678-683.

[KAR02] Karim, F.; Nguyen, A.; Dey S. "**An interconnect architecture for network systems on chips**." IEEE Micro, v. 22(5), 2002, pp. 36-45.

[KUM02] Kumar, S.; Jantsch, A.; Soininem, J-P.; Forsell, M.; Millberg, M.; Öberg, J.; Tiensyrjä, K.; Hemani, A. "**A Network on Chip Architecture and Design Methodology"**. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2002, pp. 105-112.

[LIA00] Liang, J.; Swaminathan, S.; Tessier, R. "**aSOC: A Scalable, Single-Chip communications Architecture"**. In: IEEE International Conference on Parallel Architectures and Compilation Techniques, 2000, pp. 37-46.

[LIM04] Lim, D.; Peattie, M. "**Two Flows For Partial Reconfiguration: Module Based or Small Bit Manipulations**". Xilinx Application Note 290 (XAPP290 v1.2), 2004, 28 p.

[MAR99] Marshall, A.; Stansfield, T.; Kostarnov, I.; Vuillemin, J.; Hutchings, B. "**A Reconfigurable Arithmetic Array for Multimedia Applications**". In: Field Programmable Gate Arrays (FPGA), 1999, pp. 135-143.

[MAR02] Marescaux, T.; Bartic, A.; Verkest, D.; Vernalde, S.; Lauwereins, R. "**Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs**". In: Field Programmable Logic and Applications (FPL), 2002, pp. 795-805.

[MAR03] Marescaux, T.; Mignolet, J-Y.; Bartic, A.; Moffat, W.; Verkest, D.; Vernalde, S.; Lauwereins, R. "**Networks on Chip as Hardware Components of an OS for Reconfigurable Systems**". In: Field Programmable Logic and Applications (FPL), 2003, pp. 595-605.

[MEL03] Mello, A.; Möller, L. "**Arquitetura Multiprocessada em SoCs: Estudo de Diferentes Topologias de Conexão**". Trabalho de conclusão de graduação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brasil, 2003, 110 p.

[MER98] Merino, P.; Lopez, J.; Jacome, M.; "**A Hardware Operating System for Dynamic Reconfigurations of FPGAs**". In: Field Programmable Logic and Applications (FPL), 1998, pp. 431-435.

[MIL04] Millberg, M.; Nilsson, E.; Thid, R.; Kumar, S.; Jantsch, A. "**The Nostrum backbone - a communication protocol stack for networks on chip".** In*:* VLSI Design, 2004, pp. 693-696.

[MIR96] Mirsky, E.; DeHon, A. "**MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources**". In: FPGAs for Custom Computing Machines (FCCM), 1996, pp. 157-166.

[MÖL03] Möller, L. "**Ferramentas de Reconfiguração Parcial, Remota e Dinâmica de FPGAs Virtex**". Relatório Técnico, Pontifícia Universidade Católica do Rio Grande do Sul (Programa de Pós-Graduação em Ciência da Computação), Porto Alegre, Brasil, 2003, 29 p.

[MÖL04] Möller, L.; Calazans, N.; Moraes, F.; Brião, E.; Carvalho, E.; Camozzato, D. "**FiPRe: An Implementation Model to Enable Self-Reconfigurable Applications**". In: Field Programmable Logic and Applications (FPL), 2004, pp. 1042-1046.

[MOR03] Moraes, F.; Mello, A.; Möller, L.; Ost, L.; Calazans, N. "**A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping"**. In: IFIP Very Large Scale Integration (VLSI-SOC), 2003, pp 318-323.

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. "**HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip**". Integration The VLSI Journal, v. 38 (1), 2004, pp. 69-93.

[NI93] Ni, L.; McKinley, P. "**A Survey of Wormhole Routing Techniques in Direct Networks**". IEEE Computer, v. 26 (2), 1993, pp. 62-76.

[NOL03] Nollet, V.; Coene, P.; Verkest, D.; Vernalde, S.; Lauwereins, R. "**Designing an Operating System for a Heterogeneous Reconfigurable SoC**". In: Parallel and Distributed Processing Symposium (IDPDS), 2003, 7 p.

[OST05] Ost, L.; Mello, A.; Palma, J.; Calazans, N.; Moraes, F. "**MAIA - A Framework for Networks on Chip Generation and Verification**". In: Asia South Pacific Design Automation Conference (ASP-DAC), 2005, 4 p.

[PAN03] Pande, P.; Grecu, C.; Ivanov, A.; Saleh, R. "**Design of a switch for network on chip applications**". In: International Symposium on Circuits and Systems (ISCAS), 2003, pp. 217-220.

[RIJ01] Rijpkema, E.; Goossens, K; Wielage, P. "**Router Architecture for Networks on Silicon**". In: Workshop on Embedded Systems (PROGRESS), 2001, pp. 181-188.

[RIJ03] Rijpkema, E.; Goossens, K.; Rădulescu, A. "**Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip"**. In: Design, Automation and Test in Europe (DATE), 2003, pp. 350-355.

[SAA02] Saastamoinen, I.; Alho, M.; Pirttimäki, J.; Nurmi, J. "**Proteo Interconnect IPs for Networks-on-Chip**". In: IP Based SoC Design, 2002.

[SAA03] Saastamoinen, I; Alho, M.; Nurmi, J. "**Buffer Implementation for Proteo Networks-on-Chip"**. In: International Symposium on Circuits and Systems (ISCAS), 2003, pp. II-113 - II-116.

[SAS01] Sassatelli, G.; Torres, L.; Galy, J.; Cambon, G.; Diou, C. "**The Systolic Ring: A Dynamically Reconfigurable Architecture for Embedded Systems**". In: Field Programmable Logic and Applications (FPL), 2001, pp. 409-419.

[SGR01] Sgroi, M.; Sheets, M.; Mihal, A.; Keutzer, K.; Malik, S.; Rabaey, J.; Sangiovanni-Vincentelli, A. "**Addressing the System-on-Chip Interconnect Woes Through Communication-Based Design"**. In: Design Automation Conference (DAC), 2001, pp. 667-672.

[SIG02] Sigüenza-Tortosa, D.; Nurmi, J. "**Proteo: A New Approach to Network-on-Chip**". In: IASTED International Conference on Communication Systems and Networks (CSN), 2002.

[SIN98] Singh, H.; Lee, M.; Lu, G.; Kurdahi, F.; Bagherzadeh, N.; Filho, E. "**MorphoSys: a Reconfigurable Architecture for Multimedia Applications**". In: Brazilian Symposium on Integrated Circuits and Systems Design (SBCCI), 1998. pp. 134-139.

[WAI97] Waingold, E.; Taylor, M.; Srikrishna, D.; Sarkar, V.; Lee, W.; Lee, V.; Kim, J.; Frank, M.; Finch, P.; Barua, R.; Babb, J.; Amarasinghe, S.; Agarwal, A. "**Baring It All to Software: Raw Machines**". IEEE Computer, v. 30 (9), Sep. 1997, pp. 86-93.

[WAL03] Walder, H.; Platzner, M. "**Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations**". In: Eng. of Reconfigurable Systems and Algorithms (ERSA), 2003, pp. 284-287.

[WAL04] Walder, H.; Platzner, M. "**A Runtime Environment for Reconfigurable Hardware Operating Systems**". In: Field Programable Logic and Applications (FPL), 2004, pp. 831-835.

[WIN01] Wingard, D. "**MicroNetwork-based integration for SOCs**". In: Design Automation Conference (DAC), 2001, pp. 673-677.

[WIK03] Wiklund, D.; Liu D. "**SoCBUS: Switched Network on Chip for Hard Real Time Systems"**. In: International Parallel and Distributed Processing Symposium (IPDPS). 2003, pp. 78a.

[XIL04a] Xilinx, Inc. **http://www.xilinx.com**. 2004.

[XIL04b] Xilinx, Inc. "**Virtex-II Platform Platform FPGA Handbook**". Xilinx User Guide 002 (UG002 v1.9), 2004, 490 p.

[XIL04c] Xilinx, Inc. "**Development System Reference Guide – Modular Design**". Xilinx, 2004, 484 p.

[ZEF03] Zeferino, C.; Susin, A. "**SoCIN: A Parametric and Scalable Network-on-Chip"**. In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2003, pp. 169-174.