Abordagem para Redução de Complexidade de RNA usando Reconfiguração Dinâmica

Luiz Brunelli

Tese de Doutorado submetida à Coordenação dos Cursos de Pós-Gradução em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Prof. **Elmar Uwe Kurt Melcher** Dr.
Orientador

Prof. **Raimundo Carlos Silvério Freire** Dr.
Orientador

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

B894a Brunelli, Luiz
2005 Abordagem para redução de complexidade de RNA usando reconfiguração dinâmica / Luiz Brunelli. ─ Campina Grande: UFCG, 2005.
183p.

Inclui Bibliografia

Tese (Doutorado em Engenharia Elétrica) ─ Centro de Ciências e Tecnologia, Universidade Federal de Campina Grande.

1─ Rede Neural 2─ Inteligência Artificial 3─ Eletrônica Digital
4 – Hardware I─ Título

CDU 004.032.26

Abordagem para Redução de Complexidade de RNA usando Reconfiguração Dinâmica

Luiz Brunelli

Tese de Doutorado apresentada em Fevereiro de 2005

Prof. **Elmar Uwe Kurt Melcher** Dr.
Orientador

Prof. **Raimundo Carlos Silvério Freire** Dr.
Orientador

Edna Natividade da Silva Barros, Dr. rer nat.
Componente da Banca

José Homero Feitosa Calvalcanti, Dr.
Componente da Banca

Marius Strum, Dr.
Componente da Banca

Sebastian Yuri Cavalcanti Catunda, Dr.
Componente da Banca

Campina Grande, Paraíba, Brasil, Fevereiro de 2005

Dedicatória

A minha esposa **Maria Valéria** e ao meu filho **Lucas**.

Agradecimentos

Ao meu orientador prof. *Elmar Uwe Kurt Melcher* por sua amizade, generosidade, discussões e suporte durante todo este trabalho.

Ao meu orientador prof. *Raimundo Carlos Silvério Freire* por sua amizade, pela oportunidade de fazer este doutorado e pelos meios colocados a minha disposição para a realização deste trabalho.

A profa. *Teresa Bernarda Ludemir* (DI-UFPE), pelas sugestões dadas durante a fase de pesquisa inicial do tema desta tese.

Ao prof. *Antônio Berto Machado* (CE-UFCG) por suas aulas de docência universitária.

Ao prof. *Ângelo Perkusich* (DEE-UFCG) por sua atenção, discussões e apoio durante todos estes anos.

Aos profs. *Paulo Cesar Centoducatte* e *Rodolfo Jardim de Azevedo* (UNICAMP - IC) pelas sugestões e criticas feitas durante a fase inicial deste trabalho.

Ao prof. *Wang Jiang Chau* (USP) por sua amizade e gentileza em enviar-me artigo e tese.

Ao Sr. *Bin Liu* (Atmel Corporation) por sua atenção, suporte técnico e apoio durante todos estes anos.

Ao Sr. *Philippe Butel* (MBDA France) por sua gentileza em enviar-me os documentos do projeto RECONF 2.

Aos amigos: *Antenor, Ricardo, Guilherme, Marcos, Alisson, Francisco, Leonardo, Alés-sio, Ivan, Wilson e Zurita* por todas as discussões, incentivos e conselhos durante este período.

A COPELE, as Miniblio da Engenharia Elétrica e Ciências da Computação pelo apoio dado durante todos estes anos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa de doutorado e taxa de bancada.

A *banca examinadora* por suas valiosas contribuições feitas durante o exame de qualificação e defesa da tese.

Resumo

Abordagem para Redução de Complexidade de RNA usando Reconfiguração Dinâmica

Luiz Brunelli

Nesta tese descreve-se uma nova solução para o tratamento da complexidade das interconexões entre os elementos de processamento das redes neuronais artificiais (RNAs). Ela possibilita implementar RNAs em hardware, de tecnologia digital, com um número maior de neurônios do que se faz atualmente. As RNAs têm sido usadas como solução em vários problemas complexos. Em alguns destes problemas faz-se necessário a sua implementação em hardware. Vários são os compromissos que devem ser satisfeitos durante o projeto e implementação das RNAs, dentre eles o das interconexões entre os neurônios. Atualmente encontram-se implementações neuronais utilizando circuitos integrados especificamente desenvolvidos para uma dada arquitetura de rede neuronal e também o uso de circuitos integrados configurados pelo usuário. Dentre estes circuitos existem os FPGAs reconfiguráveis dinamicamente (DR-FPGAs) que podem ter suas características alteradas durante a sua operação, sem sofrer interrupções em seu funcionamento normal. Estes dispositivos têm sido utilizados na implementação de RNAs. Propõe-se uma solução para o *problema das interconexões* entre os neurônios artificiais utilizando os DR-FPGAs e uma nova forma de computação: as Figuras de Execução (F.E.). As F.E. permitem teoricamente reduzir o impacto das interconexões através da eliminação do transporte de dados via barramento, além de outras vantagens e desvantagens durante o processamento da computação. As F.E. não parecem estar restritas apenas as aplicações de RNAs. Elas podem ser utilizadas pela computação reconfigurável em problemas massivamente paralelos e/ou que necessitem trocar informações entre os vários elementos de processamento do sistema.

Palavras chaves: Redes neuronais artificiais, VLSI, FPGA, reconfiguração dinâmica, computação reconfigurável, implementação em hardware.

Abstract

Approach for Complexity Reduction of ANN using Dynamic Reconfiguration

Luiz Brunelli

In this thesis a new solution for the treatment of the complexity in the interconnections among the processing elements of the artificial neural networks (ANNs) is described. It enables realize ANNs digital hardware implementation with a larger number of neurons than does nowadays. The ANNs have been used as a solution in various complex problems. Some of these problems require hardware implementation. A lot of constraints must be satisfied during the project flow of the implementations of ANNs, such as the neural interconnections. Nowadays, neural implementations are done using integrated circuits, specifically developed for a given neural network architecture or integrated circuits configured by the user. Among these circuits exist the dynamically reconfigured FPGAs (DR-FPGAs) which can have their characteristics changed during operation without suffering interruptions in their execution. These devices have been used for ANN implementations. It presents a proposal to solve the *interconnection problem* for artificial neurons using DR-FPGAs in a new computational way: the Execution Patterns[1] (EPs). The EPs allow, theoretically, to reduce the influence of interconnections through the removal of data transport via busses, besides other advantages and disadvantages. The EPs does not seem to be restricted only to ANN applications. They can be used by reconfigurable computation in massive parallel problems and/or problems that demand information exchange among the various elements in a processing system.

Keywords: Artificial neural networks, VLSI, FPGA, dynamic reconfiguration, reconfigurable computing, hardware implementation.

[1]Not to be confused with the new method for visualizing execution traces of object-oriented programs.

Sumário

Lista de Símbolos e Abreviaturas

ASIC	-	*Application Specific Integrated Circuit*
CI	-	Circuito Integrado
CMOS	-	*Complementary Metal Oxide Semiconductor*
CPS	-	Conexões Por Segundo
CUPS	-	Conexões Atualizadas (*Updates*) Por Segundo
DCS	-	*Dynamic Circuit Switching*
DR-FPGA	-	*Dynamic Reconfiguration Field Programmable Gate Array*
DRL	-	*Dynamically Reconfigurable Logic*
DSP	-	*Digital Signal Processor*
EP	-	Elemento de Processamento
EPs	-	*Execution Patterns*
F.E.	-	Figuras de Execução
FPGA	-	*Field Programmable Gate Array*
FPU	-	*Floating Point Unit*
HDL	-	*Hardware Description Language*
LUT	-	*Look Up Table*
MAC	-	*Multiply and Accumulate*
MC	-	Multicamadas
MSI	-	*Medium Scale Integration*
NL0	-	Nível Lógico 0
NL1	-	Nível Lógico 1
OCR	-	*Optical Character Recognition*
PAL	-	*Programmable Array Logic*

RNA	-	Rede Neuronal Artificial
RTL	-	*Register Transfer Level*
RTR	-	*Run-Time Reconfiguration*
SIMD	-	*Single Instruction Multiple Data*
SOC	-	*System On a Chip*
SRAM	-	*Static Random-Access Memory*
SSI	-	*Small Scale Integration*
T_P	-	Tempo de Propagação
TTL	-	*Transistor-Transistor Logic*
UC	-	Unidade de Controle
UCP	-	Unidade Central de Processamento
ULA	-	Unidade Lógica e Aritmética
VLSI	-	*Very Large Scale Integration*

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

As redes neuronais[1] artificiais (RNAs) têm sido utilizadas na solução de problemas que muitas vezes não possuem abordagens satisfatórias utilizando as técnicas tradicionais da computação[2]. Dentre estes problemas encontram-se: reconhecimento e agrupamento de padrões, predição e análise de dados, controle e otimização, aproximação de funções[3,4], etc.

As RNAs são sistemas de processamento de informações que possuem as seguintes características básicas em comum com as redes neuronais biológicas[5–7]:

- O processamento da informação ocorre em diversos elementos simples chamados neurônios ou unidades de processamento;
- Os sinais são passados de um neurônio a outro por meio de conexões chamadas sinapses;

[1]Nota: utilizou-se a palavra neuronais ao invés de neurais, pois neurais refere-se a nervos e não neurônios.

- Cada conexão possui uma *força* (peso) que amplifica ou atenua o sinal transmitido;
- Cada neurônio possui uma função de ativação (geralmente não linear) responsável pelo disparo do sinal de saída e tendo como estímulo a ponderação de suas entradas.

Na figura 1.1 representa-se graficamente um neurônio artificial, descrito pelas equações 1.1 a 1.3.

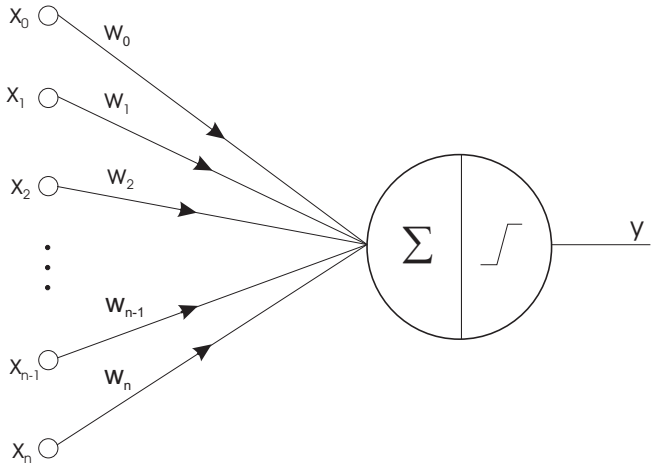

Figura 1.1: Modelo do neurônio artificial generalizado de McCulloch e Pitts

$$y = f\left(y_{in}\right) \tag{1.1}$$

$$y_{in} = \sum_{i=1}^{n} x_i w_i \tag{1.2}$$

$$f\left(y_{in}\right) = \frac{2}{1 + e^{-y_{in}}} - 1 \tag{1.3}$$

Na equação 1.3 descreve-se uma função de ativação muito utilizada em RNA. Sua representação gráfica é dada na figura 1.2.

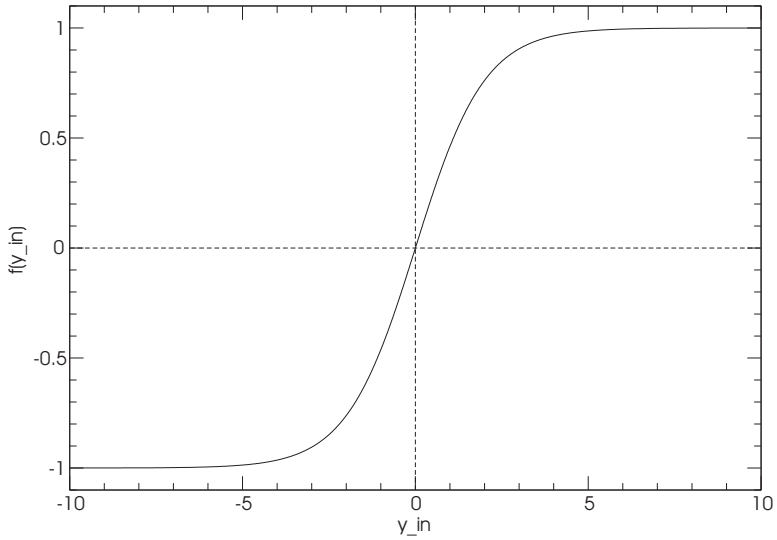

Figura 1.2: Função sigmoide bipolar

Uma RNA, antes de ser utilizada, deve passar por um processo de treinamento[5,6]. O processo de treinamento permite ajustar o valor dos pesos das conexões entre os neurônios de forma a *capturar* a solução do problema.

Atualmente, na grande maioria das aplicações das RNA, um programa seqüencial de computador *simula* o treinamento e a execução da RNA. Tal abordagem não leva em conta o paralelismo natural encontrado na topologia da RNA (figura 1.3) tanto na fase de execução como na fase do treinamento. Implementações de RNA em hardware podem ajudar na solução desta nesta questão[8–13].

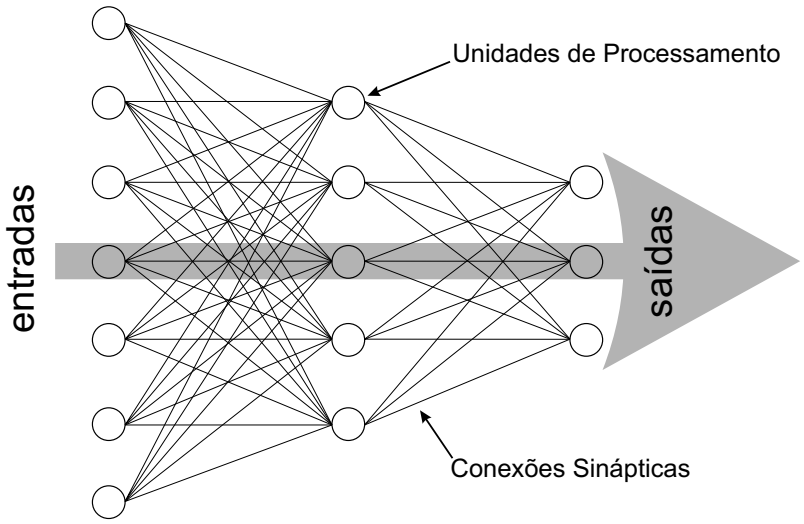

Figura 1.3: RNA de duas camadas de pesos

1.1 Interconexão entre neurônios em uma RNA

O projeto e a implementação de uma RNA em hardware deve satisfazer os seguintes requisitos principais [14–16]:

- Possibilitar a implementação de um grande número de neurônios[2];
- Alta conectividade entre os neurônios;
- Os *pesos* das sinapses devem ser alteráveis.

Dentre os requisitos citados, o que possui o maior impacto na implementação em hardware é o segundo, chamado doravante de *interconexão entre os neurônios*.

[2]O cérebro possui entre 10^{10} e 10^{11} neurônios e 10^4 e 10^5 conexões por neurônio [7].

Suponha-se duas camadas de neurônios, de tamanho n, totalmente conectadas (1.4) obtém-se uma complexidade de interconexões: $O(n^2)$ [17–19].

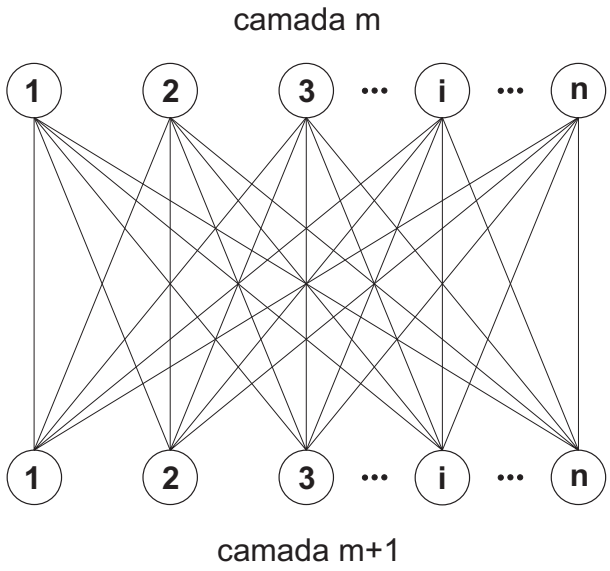

Figura 1.4: Camada de uma RNA totalmente conectada

Considerando a implementação de uma RNA, em circuitos integrados VLSI, a área de silício ocupada pelas interconexões pode à cerca de 90% da área total[3] da pastilha[20, 21]. A área de metal necessária para as interconexões entre os neurônios cresce com uma complexidade dada por $O(n^3)$, em que O é definido como uma cota assintótica superior[22] e n é o número de neurônios[2, 10, 23–26]. Este problema de crescimento da área de interconexão entre neurônios, em função do número de neurônios da camada será referenciado como *problema das interconexões*.

[3] O autor não menciona o número de camadas de metal. Em 1997, a tecnologia de circuitos integrados usava até 4 camadas de metal.

O problema das interconexões constitui-se em uma das grandes limitações à implementação de RNA em hardware, pois limita o número de neurônios que podem ser implementados em um CI.

1.2 Figuras de Execução

Nesta Tese contribuir-se com uma solução, para o problema das interconexões entre neurônios utilizando uma nova abordagem chamada de *Figuras de Execução* (F.E.). O escopo deste trabalho é limitado a implementação digital de RNA do tipo propagação direta (*feedforward*) por questões de simplificação do problema.

As Figuras de Execução combinam os conceitos de computação reconfigurável [27–30] e FPGA (*Field Programmable Gate Array*) reprogramáveis dinamicamente [1,31,32], mas *não* estão restritos à eles. Estes dois conceitos já foram utilizados na implementação digital de RNA em FPGA reconfiguráveis [17,27] e dinamicamente reconfiguráveis [33,34].

A computação reconfigurável baseia-se em um processador fixo que realiza a *computação temporal* e em um hardware reconfigurável que realiza a *computação espacial*. Entende-se por computação espacial a execução de um bloco funcional ou algoritmo diretamente em hardware[29].

O conceito de FPGA reprogramáveis dinamicamente[4] permite o mapeamento do hardware virtual no hardware físico, à semelhança do esquema de memória virtual

[4]No capítulo 4 adotar-se-a uma outra terminologia mais adequada.

dos computadores atuais, permitindo assim um aumento da densidade funcional do hardware[35].

As aplicações tradicionais que utilizam computação reconfigurável e FPGAs reprogramáveis ou reprogramáveis dinamicamente visam a aceleração de computações numéricas intensivas, produtos reconfiguráveis com diferentes personalidades ou projetos que se beneficiem da densidade funcional.

As Figuras de Execução utilizam estes dois conceitos de forma **totalmente nova**, com o objetivo de *eliminar* as interconexões entre os neurônios. Isso é obtido concentrando-se a atenção nos dados e não nas transformações sobre eles, à semelhança do que ocorreu durante a transição da Programação Modular para a Programação Orientada a Objetos. Os dados são fixos, não são transportados por barramentos ou ligações e uma vez criados não se movem pela arquitetura; e as operações aritméticas são as responsáveis pela *dinâmica* do processamento dos dados[5].

1.3 Organização do documento

Este documento está dividido em 6 partes:

- Introdução ao problema e a Tese (Introdução),
- Revisão da literatura e estado da arte (Capítulo 2, 3),
- Figuras de Execução (Capítulo 4),

[5]Pode-se estender este conceito ao processamento massivamente paralelo.

- Resultados experimentais (Capítulo 5),
- Conclusão e perspectivas (Capitulo 6),
- Apêndices (A, B, C e D).

Os conceitos e a evolução das idéias das **Figuras de Execução** (*Execution Patterns*) são apresentados no Capítulo 4.

No capítulo **Resultados experimentais** estão descritos dois experimentos de simulação de projetos utilizando reconfiguração dinâmica, a análise do impacto do tempo de reconfiguração e o estudo do impacto das interconexões da camada de entrada de uma RNA do tipo *Perceptron*.

Na **Conclusão e perspectivas** fecha-se o trabalho atual e deixam-se sugestões para futuros trabalhos.

Nos **Apêndices** do documento, encontra-se:

A : Os arquivos fontes da simulação de um bloco lógico que pode ser reconfigurado dinamicamente para apresentar o comportamento de uma porta "E" ou de uma porta "OU",

B : O roteiro para projetos e implementações de circuitos dinamicamente reconfiguráveis utilizado FPGAs da familia AT40K do fabricante Atmel,

C : Os arquivos fontes principais do estudo de caso MAC e os dados utilizados para a análise das interconexões de uma RNA,

D : São mencionadas outras contribuições feitas durante o desenvolvimento desta Tese.

Procurou-se objetivar o documento referenciando as técnicas e métodos já consagrados na literatura ao invés de apresentá-los no mesmo. Assume-se nesta Tese conhecimento prévio de redes neuronais artificiais e técnicas básicas de projeto de circuitos digitais.

oOo

Capítulo 2

Hardware para RNAs

A implementação de redes neuronais em hardware pode ser realizada utilizando-se diversas formas de processamento: analógica, digital, híbrida, óptica, etc. Cada uma das abordagens possui vantagens e desvantagens, conforme mostrado na tabela 2.1, mas atualmente a mais utilizada é a tecnologia digital seguida da analógica.

2.1 Medidas de desempenho

A comparação das diferentes implementações de RNA, quer sejam implementadas em software ou hardware, tem sido feita utilizando-se medidas de desempenho[12,38] voltadas a *velocidade* de execução/treinamento da RNA e/ou *benchmarks*.

Tabela 2.1: Formas de implementação de RNAs

Processamento	Vantagens	Desvantagens
Analógico	Permite a implementação de um grande número de neurônios[15]	Densidade das conexões em 2D[15]
	Consome menos potência e área da pastilha que as implementações digitais[36]	A area da pastilha do CI cresce com o cubo do feixe de entrada[10, 23–26]
	Neuromorfismo	Não uniformidade das características elétricas dos componentes implementados nos CI[16]
		Os componentes aritméticos implementados no CI são aproximações de operações matemáticas [16]
		Exatidão numérica limitada[16]
		Armazenamento dos coeficientes dos pesos da RNA, por um longo tempo, constitui-se um grande problema [37, 38]
		Ferramentas de síntese de circuitos analógicos[36] menos elaboradas
Digital	A exatidão numérica é função do número de bits usados na arquitetura	Densidade das conexões em 2D[15]
	O armazenamento dos coeficientes dos pesos da RNA é realizado utilizando células padrões RAM (estáticas ou dinâmicas)	A area da pastilha do CI cresce com o cubo do feixe de entrada[10, 23–26] e exponencialmente com o número total de bits necessários para representar os pesos[10]
	Ferramentas de projeto de circuitos VLSI mais elaboradas.	
Óptico	Permite implementar interconexões utilizando o espaço 3D[39]	Resposta não ideal dos componentes [16]
	Oferece a possibilidade de 10^{10} interconexões $/cm^3$ [39]	Não uniformidade dos componentes [16]
		Funções de ativação óptica com ganho muito inferior a 1 [16]

Dentre as medidas de desempenho mais utilizadas encontram-se:

- CPS - Conexões Por Segundo: parâmetro que indica a "velocidade" da RNA durante a fase de execução. O valor é obtido dividindo-se o número de conexões da RNA pelo tempo total decorrido entre a apresentação dos padrões de entrada e a geração dos valores das saídas da rede[40, pág. 294],

- CUPS - Conexões Atualizadas (*Updates*) Por Segundo: parâmetro que indica a "velocidade" da RNA durante a fase de treinamento. O valor da medida é calculado dividindo-se o número total de conexões da RNA pelo tempo total necessário para a atualização dos pesos da rede[40, pág. 294].

O *benchmark* mais utilizado[12,41] é a NETtalk, uma rede neuronal treinada com o algoritmo de *backpropagation* que traduz textos em fonemas. Possui 203 entradas, de 60 a 120 neurônios na camada intermediária e 26 saídas.

2.2 Classificação dos hardwares neuronais

Diversas metodologias de classificação de hardware neuronal[1] foram propostas[12,13,37,38,41], mas nenhuma tornou-se padrão. Neste documento são adotados os conceitos básicos, nos quais se baseiam estas metodologias, visando a organização em grupos e/ou detalhamento das diversas implementações neuronais. Ainda, devido ao escopo do trabalho,

[1] Todo dispositivo físico que permite executar uma RNA.

serão consideradas apenas as implementações eletrônicas digitais em hardware e para tal adotou-se dividi-las em quatro grandes grupos:

- *Computadores*: arquiteturas de processamento de propósito geral paralelas ou monoprocessadas (tabela 2.2);
- *Neuroaceleradores*: placas aceleradoras de hardware conectadas a um computador hospedeiro (tabela 2.3);
- *Neurocomputadores*: computadores de arquitetura massivamente paralela[2], voltados a executar e treinar RNA (tabela 2.4);
- *CI neuronais*: circuitos integrados dedicados que podem ser utilizados em aplicações embutidas, placas aceleradores e neurocomputadores (tabela 2.5).

Convém ressaltar que os exemplos apresentados nas tabelas devem ser vistos apenas como referências de implementações de RNAs em hardware, sendo que os mesmos não têm a intenção de esgotar o assunto.

[2]De acordo com a classificação de Nordoström e Svensson[41], um sistema é massivamente paralelo se possuir mais do que 4096 elementos de processamento.

Tabela 2.2: Implementações em computadores

Arquitetura do Computador	No. de Processadores	*Benchmark*	CPS	CUPS
Warp (1988)	10	NETtalk		17M
NEC SX-2[41] (1988 ou 1990)	Supercomputador Vetorial	NETtalk		72M
Connection Machine (1990)	64K	NETtalk	180M	38M
MasPar (1990)	16K	NETtalk	176M	42M
IBM GF11[3] (1990)	356	NETtalk	900M	
Transputers (1991)	6			207K
Hypercube - iPSC/860[42](1991)	32	NETtalk		11M
Array Sistólico (1992)	13K	NETtalk		248M
NEC SX-3[10] (1995)	Supercomputador Vetorial			130M
Fujitsu AP1000[42](1995)	512	NETtalk		86M
IBM 80486 @ 50 MHZ [10] (1995)	1		1,1M	0,47M
Sun - SPARCstation 10 [10] (1995)	1		3M	1.1M
AMD Athlon @ 800 MHz [43] (2001)	1		80M	20M

Obs: Esta tabela foi modificada e atualizada a partir de Šerbedžija:1996[12].

Tabela 2.3: Neuroaceleradores

Nome	Processador/Arquitetura	CPS	CUPS
HNC Anza Plus (1989)	MC68020 (UCP) + MC6881 (FPU)	6M	1,5K
SAIC Sigma-1 (1989)	Delta FPU		11M
HNC Balboa (1993)	i860	25M	9M
Heinz Nixdorf Institute RAPTOR2000[43] (2000)	FPGA, PCI, Hardware Reconfiguravel, Mapas Auto-Organizaveis de Kohonem	50G	5G

Obs: Esta tabela foi modificada e atualizada a partir de Šerbedžija:1996[12].

Tabela 2.4: Neurocomputadores

Nome	Arquitetura/Tecnologia	CPS	CUPS
Siemens Synapse (1992)	*Array* Sistólico 2D 8 x MA16(CI) + 2 x MC68040 + 128M DRAM[37]	5,12G	33M
Int. Computer Science Institute RAP (1992)	DSP (TMS 320C30) SPMD	574M	106M
Adaptive Solutions CNAPS (1993)	Matriz Linear SIMD[38] CI N64000	5,7G	1,4G
HNC SNAP (1993)	SIMD (CI 100NAP)	500M	128M
Tampere Univ. of Technology PARNEU[44] (2000)	DSP (ADSP-21062 SHARC) *Partial Tree Shape*		13.0M (Estimado)

Obs: Esta tabela foi modificada a partir de Šerbedžija:1996[12].

Tabela 2.5: CI neuronais

Nome	Arquitetura	Neurônios	Sinapses	Desempenho
Micro Devices MD - 1220 (1989)	*Feedforward* Perceptron MC	1 EP	8	8,9 MCPS
Philips Lneuro-1 (1989)	*FeedForward* MC	16 EP	64	26MCPS
Hitachi WSI (1989)	*Wafer*, SIMD, Hopfield	576	32K	138MCPS
Hitachi WSI (1990)	*Wafer*, SIMD, *Backpropagation*	144		300MCUPS
HNC 100-NAP (1991)	SIMD	100 EP		250MCPS 64MCUPS
Inova N64000 (1991)	SIMD	64 EP	128K	870MCPS 220MCUPS
Nestor/Intel NI1000 (1991)	RBF	1 EP	256×1024	40K padrões/s
Siemens MA-16 (1991)	4 cadeias sistólica	16 EP	16×16	400MCPS
Neuralogix NLX-420 (1992)	*Feedforward* MC	16		300CPS
IBM ZISC036 (1994)	RBF	36	64×36	250K padrões/s
Heinz Nixdorf Institute NBX microprocessor[45] (2003)	SOMs/SIMD	8 EP		576MCPS 48MCUPS

Obs: Esta tabela foi elaborada a partir de Baratta:1998[46] e complementada com informações de Beiu:1997 [47].

Na *paralelização* e mapeamento da RNA em computadores paralelos[3,41,48] podem ser adotadas as seguintes técnicas:

- *Paralelismo do conjunto de treinamento*: divide-se o conjunto de treinamento entre várias cópias de uma uma mesma RNA. Elas armazenam localmente a matriz de pesos da rede sob treinamento e no final a variação dos pesos (Δw). As variações de todas as cópias são somadas (aprendizado por bloco/epoca[3]) gerando a nova matriz de pesos;

- *Paralelismo por camada*: uma RNA *multicamadas* pode usar a técnica de *pipeline*, permitindo assim processar mais de um padrão de entrada ao mesmo tempo. Enquanto uma camada está processando um padrão, a camada anterior processa o padrão seguinte;

- *Paralelismo por neurônio*: para cada valor de entrada calcula-se sua contribuição parcial para os vários neurônios da proxima camada (figura 2.1). Este processo de *difusão* das entradas (figura 2.2) é bastante utilizado (ex: [17,19]), pois reduz a complexidade computacional espacial de $O(n^2)$ para $O(n)$ mas transforma a complexidade computacional temporal de $O(1)$ para $O(n)$, em que n é o números de neurônios;

- *Paralelismo por sinapse*: o paralelismo é feito neurônio a neurônio seqüencialmente (figura 2.1).

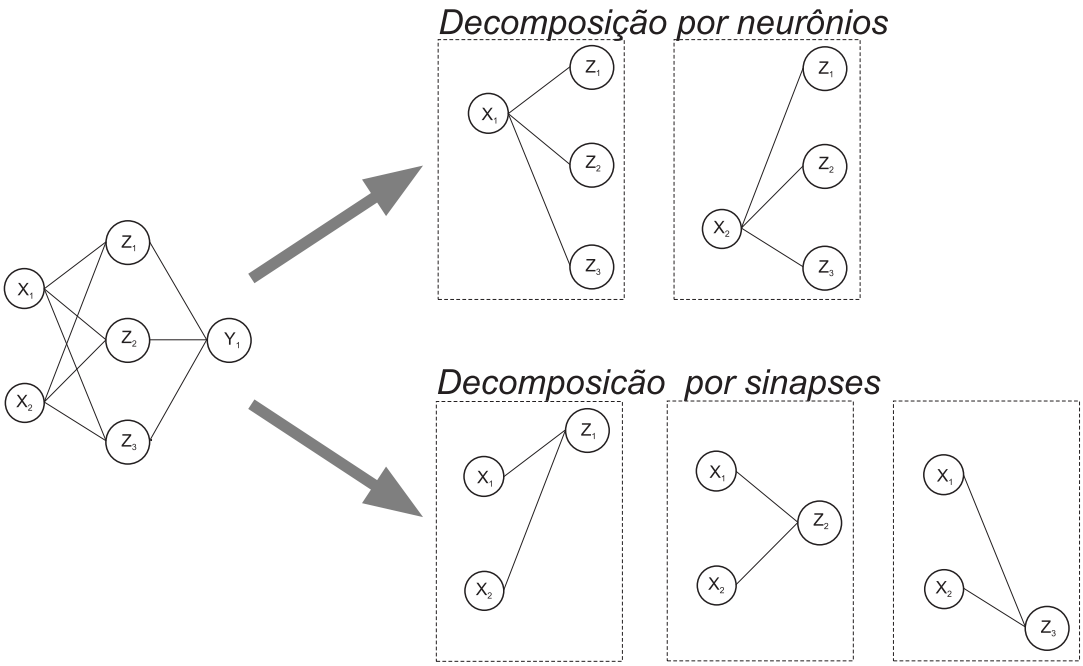

Figura 2.1: Paralelismo de uma RNA

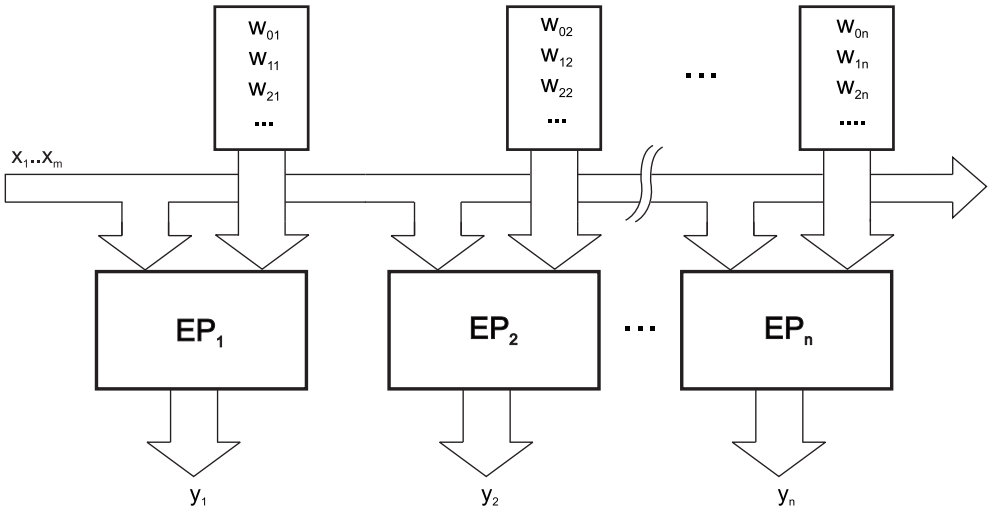

Figura 2.2: Implementação típica de uma RNA

Atualmente a implementação de RNAs em CI ou *wafers* está limitada[3][10] a redes neuronais com: de 10^2 a 10^3 entradas totais e de 10^3 a 10^4 neurônios[4] e mesmo com tecnologia sub-micron é muito difícil em um única pastilha integrar sistemas digitais massivamente paralelos[49]. Uma outra constatação é que o grande número de interconexões entre neurônios é um dos fatores principais de consumo de área de silício, na implementação digital de RNAs[49].

2.3 Considerações gerais

Atualmente observa-se um *desinteresse* crescente por implementações de RNAs em hardware neuronal programável e placas aceleradoras[38]. Isto deve-se aos seguinte fatores:

- Alto custo do produto, tempo de desenvolvimento da solução (CI + hardware + software) e tempo de introdução no mercado[38, 50];
- Avanços constantes de desempenho dos processadores de propósito geral (ex: conjunto de instruções de processamento paralelo)[50];
- De modo geral, a grande maioria das aplicações existentes de RNAs possuem menos do que 100 neurônios e necessitam apenas de treinamento ocasional. Devido a isso, elas são implementadas em software[50], não necessitando de hardware

[3] O autor não menciona nenhuma informação sobre o consumo de energia/conexão.
[4] Sendo que cada neurônio pode ter 10^2 a 10^3 entradas.

paralelo[3];

- Soluções que utilizam redes neuronais necessitam de etapas de pré e pós- processamento dos dados. Suponha que um programa de OCR, gaste 23% do seu tempo no pré- processamento, 67% no processamento da RNA e 10% no pós-processamento. De acordo com a Lei de *Amdahl* (vide[51, pág. 541, *Software Metrics*]), se a RNA for executada paralelamente e possuir uma velocidade de execução infinita o sistema terá um fator de ganho global de apenas 3[38].

Por outro lado existem aplicações[38], para as quais, mesmo o estado da arte dos processadores seqüências atuais não é suficiente. Tais como:

- Aplicações de RNA em tempo real;
- Treinamento de RNA com grande números de neurônios e sinapses[5];
- Sistemas embutidos.

Nestas três situações podem ser utilizadas arquiteturas de hardware neuronal. Como solução, os FPGA estão assumindo grande importância nas implementações de RNA[43] devido às seguintes características:

[5]Convém ressaltar que não existem teorias e aplicações para grande redes neuronais artificiais (milhares de neurônios)[50].

- Elevada densidade de recursos lógicos[46];
- Possibilidade de diminuição do tempo de introdução dos produtos no mercado[43, 46];
- Possibilidade de reconfiguração dinâmica;
- Os progressos da microeletrônica irão prover novos dispositivos mais poderosos.

oOo

Capítulo 3

FPGA dinamicamente reconfigurável

3.1 Introdução e nomenclatura

Com o avanço da microeletrônica o projeto e a implementação de circuitos digitais passou por diversas fases[31]:

- Década de 1960: os componentes utilizados nos projetos digitais eram os CI das séries 74XX (TTL) e 40XX (CMOS). Estes componentes integravam portas lógicas (SSI) e subsistemas digitais (MSI). Nesta fase o projeto era feito utilizando CI de prateleira e tinha um cunho estrutural (blocos lógicos e interconexões);

- Década de 1970: nesta época apareceram os primeiros circuitos integrados VLSI: memórias dinâmicas, microprocessadores e microcontroladores. Com o advento do microprocessador o hardware tinha o seu comportamento ditado pelo software. Projetava-se uma placa de hardware de propósito geral e utilizava-se os compo-

nentes da geração anterior como agregadores lógicos (*glue logic*) das pastilhas VLSI;

- Década de 1980: os PAL (*Programmable Array Logic*) foram utilizados como substitutos da lógica de agregação. Este dispositivo precisa ser personalizado, antes de utilizado e para tal faz-se necessário utilizar um software de desenvolvimento. O software recebe como entrada, por exemplo, as equações lógicas booleanas, faz a minimização delas, fornece os dados necessários para a visualização das formas de onda do circuito (*timing*) e gera o arquivo de configuração do dispositivo. Para projetos mais complexos é possível descrever o hardware em nível comportamental, utilizando-se uma HDL (*Hardware Description Language*) e o software gera *automaticamente*[1] a descrição estrutural e o arquivo de configuração. Esta fase caracterizou-se pela introdução dos ASIC (*Application Specific Integrated Circuit*) e FPGA;

- Década de 1990: as tecnologias digitais foram aplicadas com grande sucesso em diversas áreas: telefonia celular, redes de computadores, produtos de consumo.

Atualmente vive-se a fase do SOC (*System On a Chip*) e o inicio do *Hardware/Software Co-Design*.

O FPGA[52,53] é um dispositivo programável pelo usuário que pode ser utilizado em um produto final, ou para fins de prototipagem. Ele tem sido uma alternativa aos ASIC

[1] O projetista de hardware foi liberado desta tarefa.

devido ao baixo custo no desenvolvimento e por permitir uma diminuição do tempo de introdução de novos produtos no mercado.

Internamente um FGPA possui os seguintes elementos básicos:

- Matriz de blocos lógicos: nestes elementos são implementadas as funções lógicas do sistema;
- Células de E/S: elas estão ligadas aos pinos do CI e quando programadas podem adquirir o comportamento de entrada ou saídas ou E/S;
- Recursos de interconexões: eles permitem interconectar os blocos lógicos entre si e também interligá-los as células de E/S;
- Estrutura de configuração: ela "fixa" o comportamento dos blocos lógicos e o caminho das interconexões entre eles, sejam eles blocos lógicos ou células de E/S.

O tipo mais comum de memória de configuração utilizadas pelos fabricantes de FPGA é a *Static Random-Access Memory* (SRAM)[54], que pode ter sua configuração definida apenas durante o ciclo de inicialização ou ser reconfigurado durante a sua execução.

Lysaght[1] propôs em 1993 uma nomenclatura que engloba os vários tipos de reconfiguração dos diversos FPGA:

- *Dinamicamente reconfigurável*: nestes FPGA pode-se reconfigurar qualquer elemento lógico ou de interconexão enquanto o restante do FPGA continua operando

normalmente;

- *Parcialmente reconfigurável*: permite uma reconfiguração seletiva enquanto o restante do dispositivo permanece inativo e retendo a sua informação de configuração;
- *Reconfigurável*: nesta categoria são incluídos todos os tipos de FPGA exceto os FPGA que são do tipo *one-time programming*;
- *Programável*: abrange todos os tipos de FPGA.

3.2 Reconfiguração dinâmica

O conceito de reconfiguração dinâmica permite endereçar duas situações que podem ocorrem em um FPGA[54]:

- A presença de um hardware inativo ou não mais utilizável (localidade temporal);
- Uma necessidade de hardware maior do que os recursos disponíveis no FPGA (localidade funcional).

Um dos problemas encontrados no campo da computação reconfigurável é a falta de ferramentas de síntese de alto nível (comportamental) que façam uso do conceito de reconfiguração dinâmica, pois as atuais foram projetadas para arquiteturas fixas ou estáticas encontrada nos ASIC[54]. Atualmente os projetos de sistemas dinamicamente

reconfiguráveis são feitos utilizando a metodologia tradicional: *capture-simule*, ferramentas de síntese lógica (nível RTL - *Register Transfer Level*). Num projeto típico indentificam-se as tarefas[2] que podem ser beneficiadas da localidade temporal e funcional e as de caracter fixo. Na figura 3.1 mostra-se um sistema constituído pelas tarefas T_1 a T_{11}, observa-se que T_2 e T_6 são constantes durante toda a execução do sistema e T_7 durante a Fase 2 e 3. Inicialmente o DR-FPGA (*Dynamic Reconfiguration Field Programmable Gate Array*) recebe o arquivo de configuração do primeiro *floorplan* carregando as tarefas T_1 a T_6, após um certo tempo o FPGA é reconfigurado[3] e são destruídas as tarefas T_1, T_3 a T_5, mantidas as tarefas T_2 e T_6 e alocadas T_7 e T_8. O processo de reconfiguração continua até ser executado o último *floorplan*.

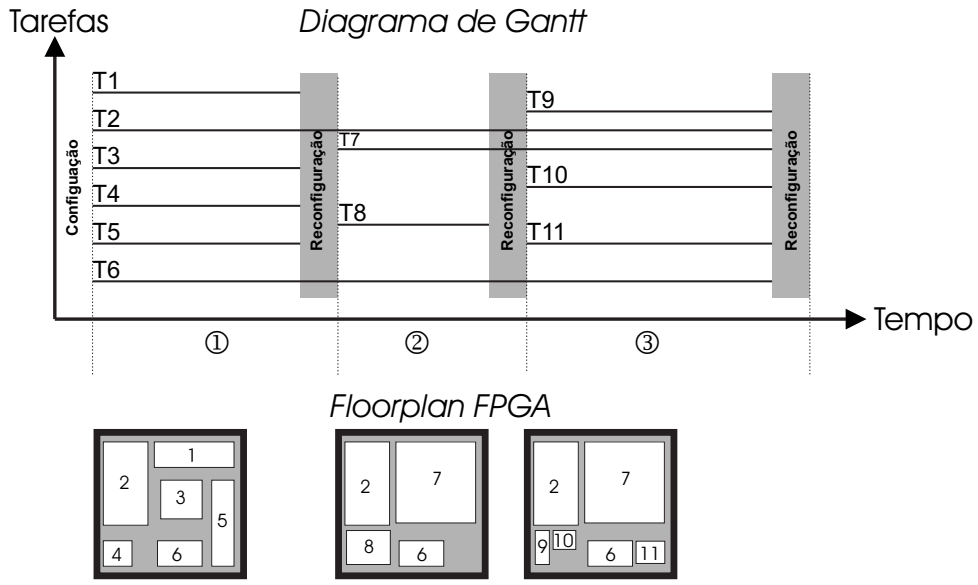

Figura 3.1: Exemplo de um sistema dinamicamente reconfigurável[1]

[2]Ou blocos do ponto de vista do hardware.
[3]As areas que não são alvo de modificação continuam operando normalmente.

Uma das formas de controlar o processo de configuração do DR-FPGA consiste na utilização de um microcontrolador[4] que forneça o arquivo de configuração do FPGA no instante adequado. Nesta abordagem H&S *Co-design* o FPGA executa as tarefas que precisam desempenho em hardware satisfazendo assim as especificações do projeto.

Conforme mencionado anteriormente, devido a falta de ferramentas de síntese de alto nível, o projetista de hardware necessita:

- Dividir *manualmente* o problema em cenários;
- Capturar (HDL), sintetizar (RTL)[5], fazer a verificação funcional do projeto, depura-lo e gerar o arquivo de configuração para cada um dos cenários, levando em conta os espaços liberados e livres do DR-FPGA;
- Fazer o teste de integração físico dos vários cenários *manualmente*.

3.3 Aplicações em redes neuronais artificiais

As implementações de RNA em DR-FPGA encontradas na literatura são poucas e a título de exemplo será vista a implementação RRANN2 da Brigham Young University (USA) e mencionadas outras duas.

A RRANN2[34, 55] é uma RNA multicamadas do tipo *Perceptron*, com treinamento em hardware do tipo *backpropagation* e implementada com FPGA parcialmente recon-

[4]Ou um SOC contendo um microcontrolador + DR-FPGA.
[5]Para uma dada família e fabricante.

figurável. Ela foi uma otimização e evolução da RRANN[17, 56] desenvolvida em 1993.

A RRANN utilizava um FPGA do tipo SRAM estaticamente reconfigurável[6], o XC3090 do fabricante Xilinx. Para reduzir a complexidade das interconexões foi adotado o esquema de difusão das entradas (figura 2.2). Sua arquitetura foi dividida em 3 módulos: *Feedforward*, *Backpropagation*, *Update* que são carregados no FPGA e executados em seqüencia, para cada um dos padrões do conjunto de treinamento. Os resultados alcançados pela arquiteturas foram:

- *Densidade funcional*: a reconfiguração total multiplicou-se por 6 o número de neurônios implementados por FPGA ao preço de uma diminuição de 30% no desempenho da rede, durante a fase de treinamento;

- *Impacto do tempo de reconfiguração*: o tempo de reconfiguração representou 88% do tempo de execução total, durante a fase de treinamento da rede; Cada uma das fase levou 7,0 ms para ser reconfigurada.

A RRANN2 utilizou como base a RRANN e o FPGA CLAy31 da National Semicondutor[7] que suporta reconfiguração parcial. O projeto do RRANN foi migrado para o novo FPGA e os três módulos reconfiguráveis foram otimizados manualmente para suportar a reconfiguração parcial. Os resultados obtidos pela RRAN2 foram:

- *Densidade funcional*: 50% maior do que a RRANN (9 neurônios/FPGA)[8];

[6] Nesta classe de FPGA só existe a possibilidade de reconfiguração total.
[7] Este FPGA foi descontinuado em 1999.
[8] Por causa do *place* e roteamento manual.

- *Impacto do tempo de reconfiguração*[55]: 25% menor do que a RRANN e de 53,5% sobre a reconfiguração total do FPGA. O CLAy31 possui um tempo de reconfiguração de 808 μs.

Lysaght et all[33] utilizando um DR-FPGA (Atmel AT6005) e aritmética de cadeia de pulsos (*Pulse-Stream Arithmetic*) implementaram uma RNA XOR dinâmicamente reconfigurável obtendo os seguintes resultados:

- Cada camada leva $6,5$ μs (relógio do sistema = 20 MHz) para produzir um dado de saída;
- O tempo de reconfiguração entre camadas é de $17,6$ μs (relógio de configuração = 10 MHz);
- A RNA fornece um dado a cada $41,67$ μs, incluído o tempo de reconfiguração do FPGA e o que correspondente a um desempenho de 0,77 MCPS.

Moreno et all[57] implementaram uma RNA em um SOC, contendo um DR-FPGA, utilizado um multiplicador serial de 8×8 bits reprogramado dinamicamente. A RNA é constituída de 12 elementos multiplicador-somador interligadas via esquema de difusão das entradas e apresentou um desempenho máximo de 70 MCPS, utilizando um relógio de 96 MHz.

oOo

Capítulo 4

Figuras de Execução

4.1 Idéias iniciais

As Figuras de Execução vieram do conceito de computação espacial[29] implementado pela computação reconfigurável. A computação espacial permite o mapeamento de um algoritmo ou tarefa diretamente em hardware através de um dispositivo de lógica programável. A título de exemplo pode-se supor a equação $y = ax + b$, mapeada em um FPGA, cuja implementação pode ser vista na figura 4.1.

Durante a execução da computação não foi necessário buscar instruções na memória, transportar dados até as Unidades Lógica e Aritmética (ULAs) o que se traduz por um desempenho maior do que se a computação fosse executada temporalmente em um processador de propósito geral.

O próximo passo, em direção as Figuras de Execução, foi encontrar uma forma de evitar o gargalo de von Neumann[58] utilizando os conceitos de computação espacial. Uma das maneiras de fazê-lo é aproximar as operações dos dados, evitando assim o

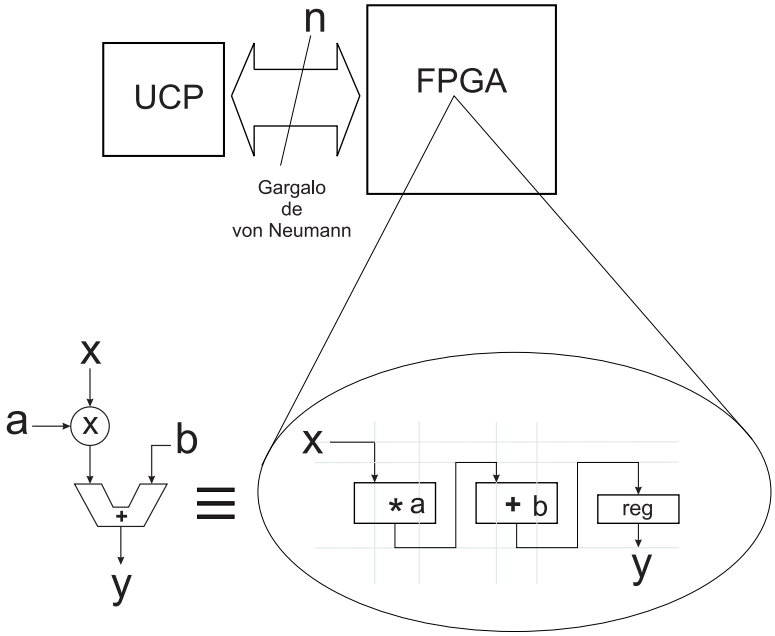

Figura 4.1: Implementação da expressão $y = ax + b$ utilizando computação espacial

transporte dos mesmos por um duto de comunicação. Duas novas operações surgiram, a partir desta aproximação: *separar* e *reunir*. Na figura 4.2 é vista a seqüência de execução da expressão $A[i] = 5 * A[i]$, $0 \leq i \leq 3$.

Inicialmente, o espaço de execução recebe o vetor no estilo von Neumann[1], aplica-se a operação de separar aos dados. Em seguidas faz-se as operações de: *juntar forças*, instanciar, executar e reunir. Ao término do processo a Unidade Central de Processamento (UCP) recolhe os dados.

[1]Para um melhor aproveitamento do conceito de F.E. faz-se necessário desenvolver periféricos e dispositivos de armazenagem que não utilizem dutos para o transporte dos dados.

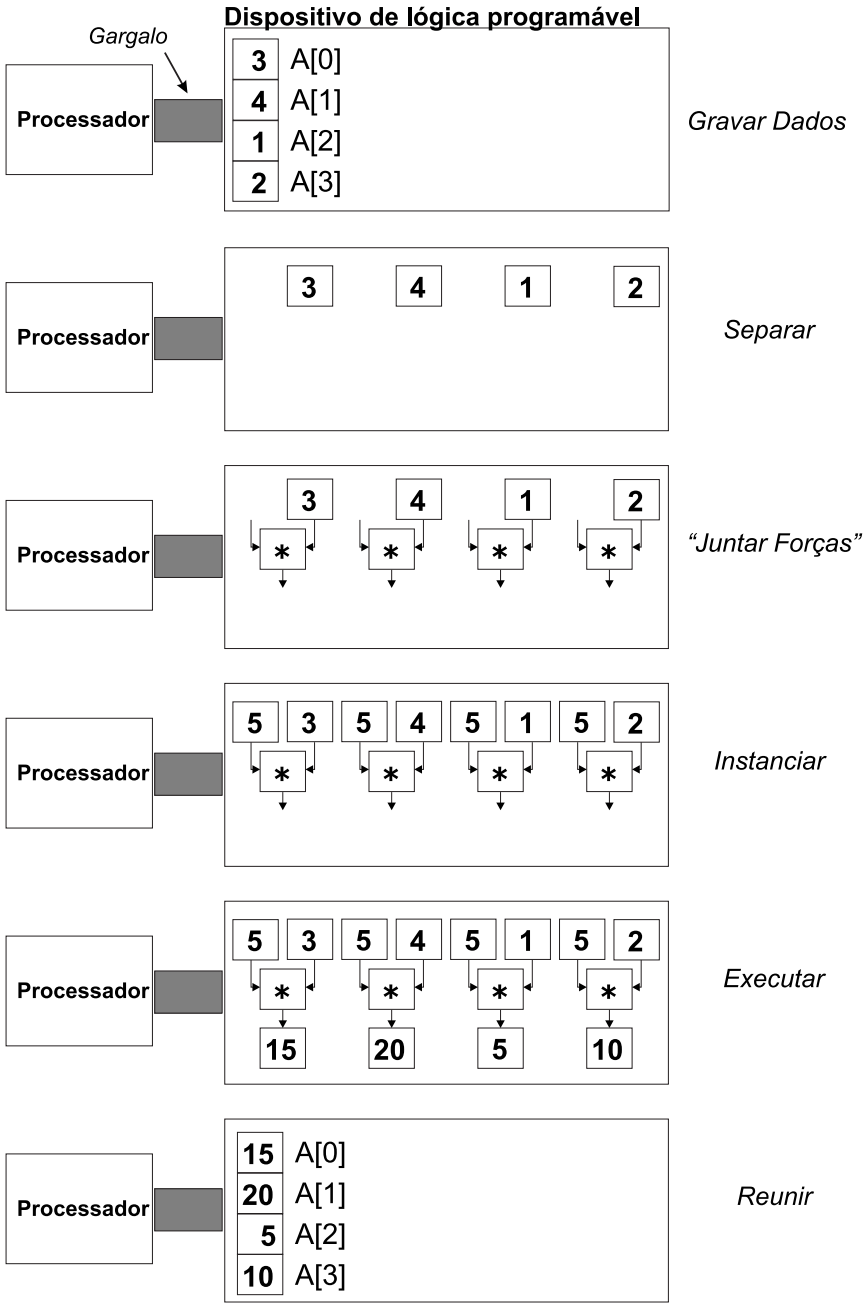

Figura 4.2: Implementação da $A[i] = 5 * A[i]$

Esta abordagem inicial veio mostrar que o importante são os dados e não as ope-

rações que são aplicadas sobre eles. Outra evidência é que se os dados estiverem disponíveis nas proximidade dos operadores as várias "atividades computacionais" ocupam regiões independentes. Quanto a implementação tecnológica, atualmente ela pode ser feita utilizando-se um FPGA com capacidade de reconfiguração parcial.

4.2 Figuras de Execução e RNAs

Retomando as RNAs e utilizando-se os conceitos acima descritos, é possível implementar o paralelismo dos neurônios da rede, mas ainda se fez necessário encontrar uma solução para o problema das interconexões entre eles. Uma mudança de paradigma foi necessária:

> "Se o problema são as interconexões porque não eliminá-las!"

Isso foi obtido *levando-se as operações aos dados* e fixando-os. Eliminando-se assim, a necessidade de transportá-los por fios e/ou barramentos e com isso as interconexões.

A título de exemplo pode-se supor a execução das seguintes equações 4.1 utilizando-se o conceito de Figuras de Execução e um FPGA reprogramado dinamicamente ideal. Mostra-se na figura 4.3 as diversas fases de carga e execução da computação.

$$\begin{cases} z_1 = x_1 w_{11} + x_2 w_{21} \\ z_2 = x_1 w_{12} + x_2 w_{22} \\ z_3 = x_1 w_{13} + x_2 w_{23} \end{cases} \tag{4.1}$$

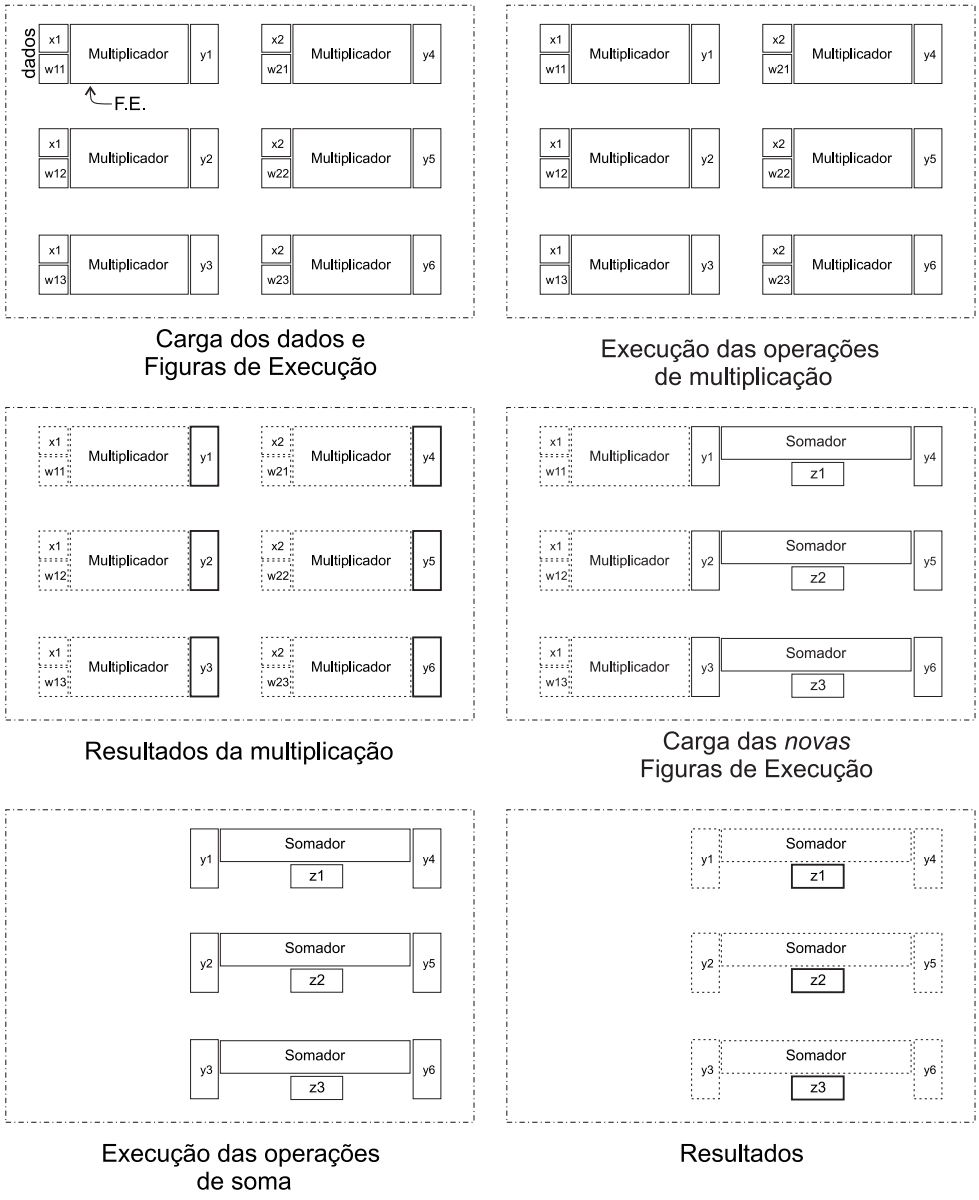

Figura 4.3: Computação por Figuras de Execução

Por meio da figura 4.3 pode-se constatar, de modo qualitativo, que o *processamento foi realizado sem o transporte de dados e com complexidade* O(1)[2]. A redução da

[2]Por questões de simplificação, considerou-se que a área (células) do DR-FPGA é muito maior que o necessário para se realizar a computação.

complexidade permite implementar RNAs em hardware, de tecnologia digital, com um número maior de neurônios do que se faz atualmente.

As figuras de execução permitem:

- A eliminação das interconexões internas entre os módulos funcionais;
- Uma descrição mais natural do paralelismo de certas aplicações computacionais;
- A eliminação do gargalo de von Neumann;
- A libertação do estilo von Neumann de programação[58];
- A construção de novas arquiteturas computacionais e hardwares específicos VLSI que a implementem;
- Adicionar uma dimensão geométrica ao processamento, pela exploração das arestas dos módulos funcionais.

Como limitações atuais[3] podem-se citar:

- Atualmente o processamento de entrada/saída é feito seqüencialmente por barramentos de dados. Como as Figuras de Execução não utilizam canais de dados se faz necessário interligar os dois conceitos[4];
- A unidade de controle tem um aspecto seqüencial no trato do programa de controle das Figuras de Execução;

[3]Questões abertas para pesquisa.

[4]Sugere-se estudar uma nova forma de comunicação massivamente paralela entre periféricos e as F.E.

- O tratamento da alocação das Figuras de Execução no espaço de execução é n-dimensional, o que introduz $(n-1)$ graus a mais no gerenciamento deste;
- Escalonamento do processamento para n graus de liberdade;
- Estudo de um mecanismo eficiente de carregar as Figuras de Execução no espaço de execução.

As Figuras de Execução **não estão limitadas** apenas ao domínio das RNAs. Devido as suas características próprias, tais como: paralelismo natural, ausência de transporte de dados e o conceito de arestas de dados, elas também podem contribuir em outros domínios, nos quais sejam necessário processamento paralelo e o impacto das conexões entre os elementos de processamento seja grande.

No capítulo 5 é feita uma análise quantitativa das F.E. visando identificar suas vantagens e desvantagens.

oOo

Capítulo 5

Resultados experimentais

5.1 Simulação da reconfiguração dinâmica

A simulação da lógica dinamicamente reconfigurável[1] tem sido feita principalmente das seguintes formas:

- Simulação do DR-FPGA[59];
- *Clock morphing.* Esta técnica baseia-se em dois princípios: a reconfiguração dinâmica é modelada tendo em vista os elementos de armazenamento e na criação de um novo sinal lógico que indica o estado virtual dos elementos durante a reconfiguração.[60];
- Pares casados de blocos de controle (ex: multiplex/demutiplex) que chaveiam grupos de tarefas dinâmicas e mutuamente exclusivas[61];

[1] *Dynamically reconfigurable logic (DRL).*

- Chaveamento dinâmicos de circuitos[2] que usam chaves de isolação para modelar a reconfiguração dinâmica[62–64].

Dentre as vantagens e desvantagens de cada abordagem escolheu-se a técnica DCS devido a capacidade única de modelar o intervalo de reconfiguração(vide [63, pág 176 item 2]).

5.1.1 Chaveamento dinâmico de circuitos

No projeto de circuitos reconfigurados dinamicamente encontramos dois tipos de tarefas: as estáticas e as dinâmicas. As tarefas estáticas, são aquelas que permanecem constantes durante todo o tempo de execução do circuito, tais como: interfaces de entrada/saida, circuitos de controle da arquitetura, etc. Já as tarefas dinâmicas constituem-se das atividades funcionais que são executadas durante um certo intervalo de tempo e que darão lugar a outras tarefas quando retiradas.

As tarefas dinâmicas são agrupadas em conjuntos ativos e inativos, os quais abrigam *tarefas mutuamente exclusivas*. Elas são conectadas as tarefas estáticas por meio de chaves de isolação (figura 5.1) que permitem colocar um conjunto de tarefas em um dos seguintes estados: inativa, ativa e transição.

[2] *Dynamic circuit switching* (DCS).

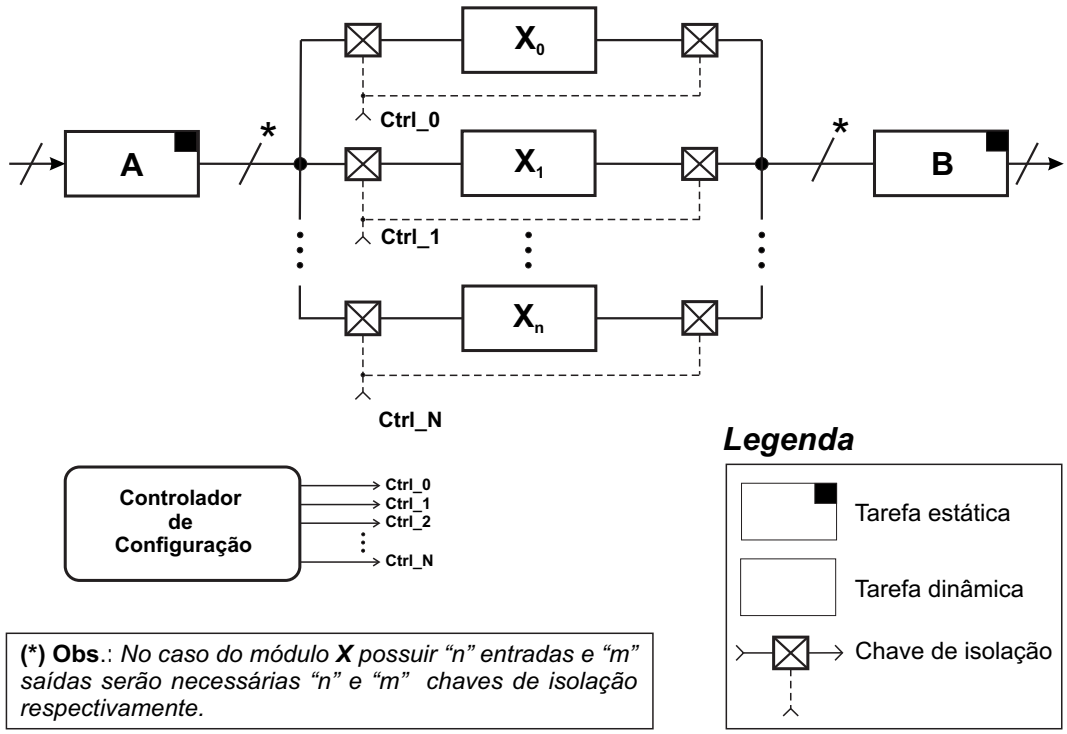

Figura 5.1: Estrutura de simulação utilizando DCS

A chave de isolação possui uma entrada (entrada), uma saída (saida) e uma variável de controle (reconf_b) e sua tabela da verdade pode ser vista na tabela 5.1.

Tabela 5.1: Tabela da verdade da chave de isolação

reconfig_b	saida	Estado do conjunto
0	entrada	Ativo
1	x	Transição
z	z	Inativo
x	z	Inativo (adotado)

Obs.: Conjunto são blocos em operação.

A chave_isolação quando ativa ($reconf\_b = 0$) conecta a entrada a saída; se $reconf\_b = 1$ a saída passa para o estado x (representa um valor lógico desconhecido) indicando que

durante a reconfiguração a saída do bloco é indeterminada e no estado inativo o bloco encontra-se desconectado (alta impedância).

A título de exemplo o comportamento da chave de isolação foi modelado em Verilog (veja o apêndice A) e simulado como conforme pode ser visto na figura 5.2 . Pode-se ver pelo diagrama de tempo que a chave passa por 3 estados: inativo ($controle\_b = z$ e $b \leftarrow z$), transição ($controle\_b = 1$ e $b \leftarrow x$) e ativo ($controle\_b = 0$ e $b \leftarrow a$).

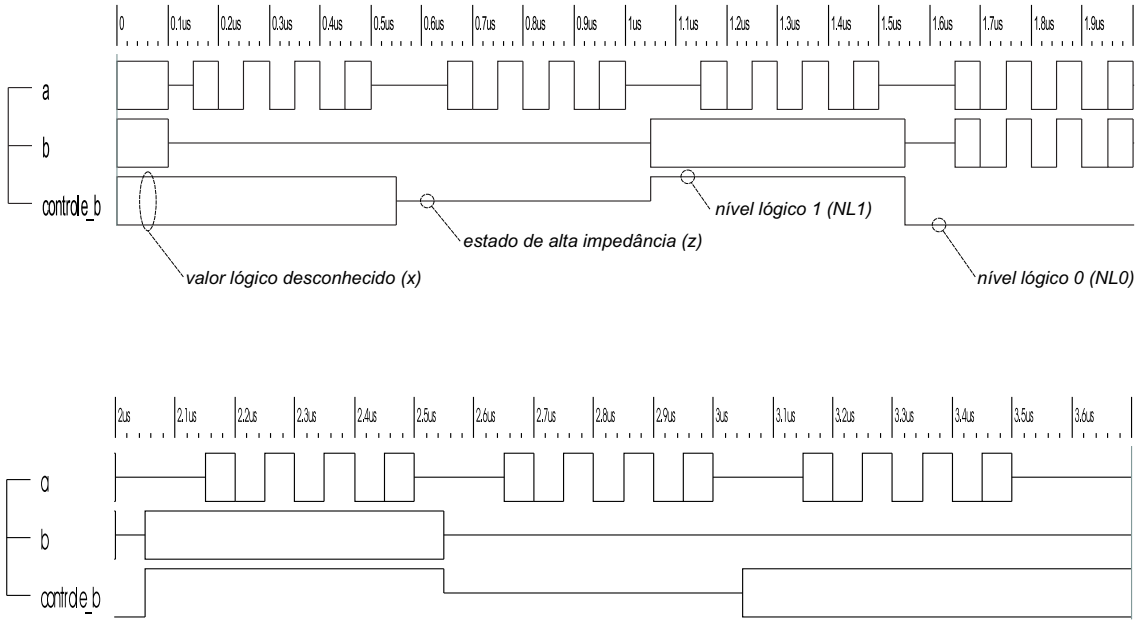

Figura 5.2: Simulação da chave de isolação

No caso da reconfiguração de um bloco B sobre um bloco A, mutuamente exclusivos, deve-se respeitar a seqüência de eventos indicada na figura 5.3. O estado de transição, após o estado ativo, tem a função de retirar (voltar os recursos do FPGA à uma condição operacional segura) a diferença entre o novo bloco a ser carregado e o antigo (figura

5.4), pois o mesmo poderá causar efeitos indesejáveis nos circuitos ativos e/ou dano ao FPGA.

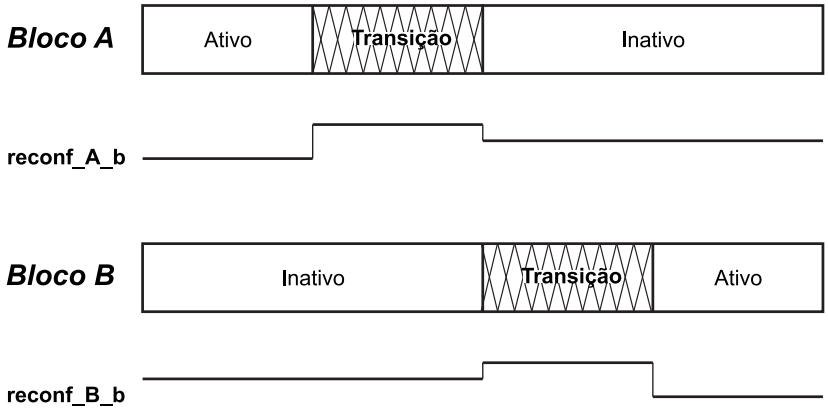

Figura 5.3: Seqüência de estados durante a reconfiguração de $B \rightarrow A$

Cabe ainda ao projetista verificar se existe a ativação simultânea dos dois blocos, isto é $reconf\_A\_b = 0$ e $reconf\_B\_b = 0$ (figura 5.3), o que pode acarretar a ativação de *ambas* as saídas dos blocos.

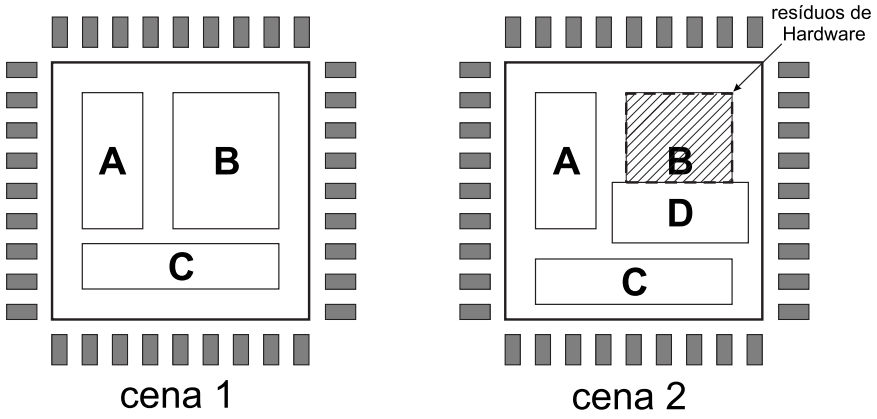

Figura 5.4: Reconfiguração de um DR-FPGA: *cena* 1 $\rightarrow$ *cena* 2 com superposição de modulo

Outro fator a ser considerado são os tempos de transição entre os estados ativo e inativo e vice-versa dos blocos, ele é função direta do *tipo* de arquivo de configuração do FPGA (*bitstream*). Basicamente existem quatro tipos de arquivos de configuração:

- *Completo*: O arquivo de *bitstream* além de possuir os dados de configuração dos blocos, contém dados que levam o restante dos recursos lógicos do FPGA a serem colocados em uma condição de operação segura[3];
- *Comprimido*: existem apenas os dados de configuração do bloco e admite-se a priori que todos os recursos do FPGA encontram-se em uma condição de operação segura;
- *Janela comprimida*: Apenas as diferenças de configuração, entre os blocos futuro e o atual estão presentes no arquivo de configuração. Admite-se que o FPGA encontra-se em uma condição de operação segura (responsabilidade do projetista);
- *Janela completa*: Existem as diferenças de configuração, entre os blocos futuro e o atual, sendo que o restante dos recursos lógicos é colocado em uma configuração de operação segura.

Se faz necessário minimizar o arquivo de configuração, escolhendo a estratégia de configuração mais adequada, visando diminuir os tempos de transição e conseqüentemente o de reconfiguração do FPGA.

[3]A mesma condição que se encontra o FPGA após o reset.

5.2 Exemplo de simulação

No exemplo foi simulada a reconfiguração de uma porta lógica E em OU[4]. Utilizou-se um DR-FPGA de mercado (Atmel AT40K40-2BGC[65]) e sua ferramenta (Figaro) de posicionamento, roteamento, extração de arquivos de pós-simulação[5] e geração de arquivos de configuração.

A família AT40K possui uma interface de configuração[66] semelhante a uma memória SRAM síncrona com 23 bits de endereço, 8/16 bits de via de dados (configurável via registrador) e que permite escrever uma palavra de configuração a cada 30 ns (t_{min}) ou lê-la a cada 1.000 ns (t_{min}). No simulador adotou-se: 16 bits de via de dados e $t_{clock\_reconf} = 30\ ns$ visando o tempo mínimo de reconfiguração do FPGA.

5.2.1 Diagrama de blocos E_OU

O diagrama em blocos da estrutura de simulação (figura 5.5) é constituído de 3 partes principais: a cena 00 que encapsula a porta lógica E, a cena 01 que abriga a porta lógica OU e a unidade de controle da reconfiguração, responsável pela coordenação do processo de simulação da reconfiguração dinâmica.

Na tabela 5.2 encontram-se descritos os sinais da unidade de controle (UC) bem como suas funções.

[4]Os arquivos fontes encontra-se no apêndice A.

[5]Entende-se, neste documento, por *pós-simulação* a simulação feita pós-leiaute em um FPGA. A ferramenta de posicionamento e roteamento extrai da base de dados a descrição estrutural HDL do projeto, bem como os atrasos presentes no elementos lógicos e recursos de interconexão.

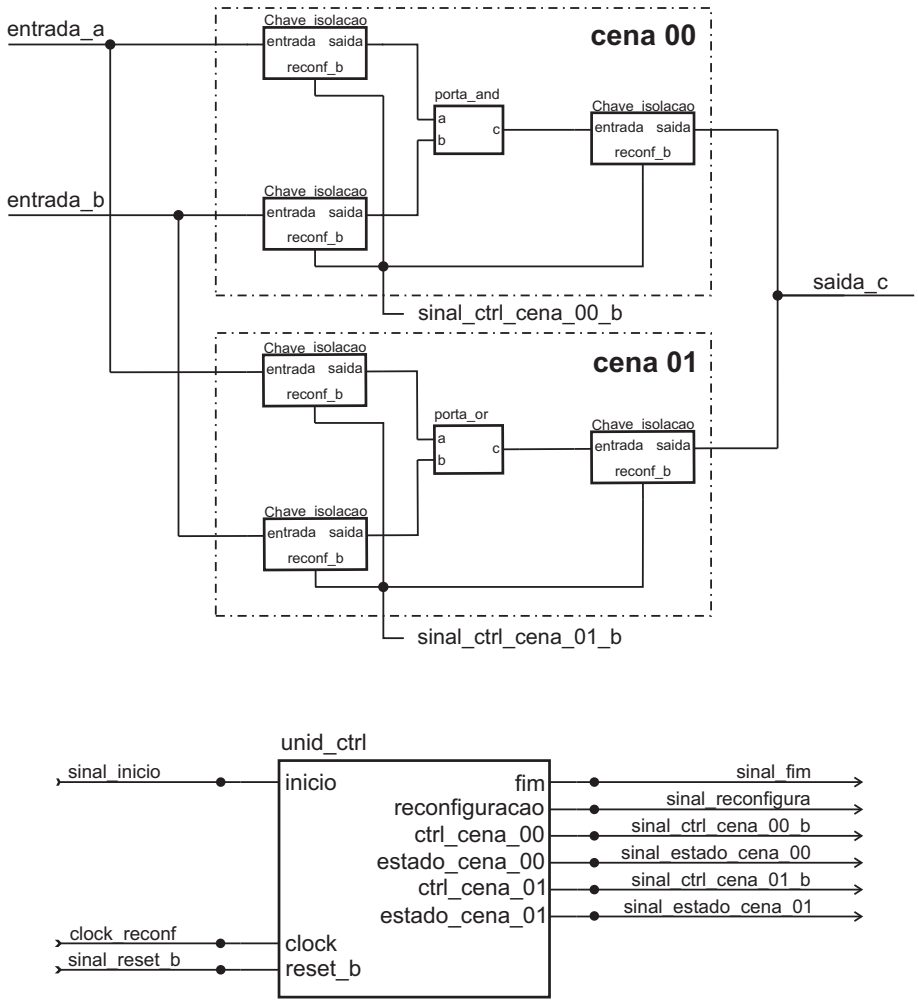

Figura 5.5: Diagrama de blocos do exemplo E_OU

5.2.2 Principio de funcionamento

Ao receber o sinal_reset_b a UC (vide a figura 5.5) gera um ciclo de configuração[6] do "FPGA", tendo como condição inicial a carga da cena 00 e ao final do processo o sinal_fim pulsa indicando o termino da configuração. O "FPGA" está pronto para

[6]Adota-se o termo configuração e não reconfiguração pois após o reset o FPGA encontra-se em um estado inerte e o mesmo deve ser configurado para entrar em operação.

Tabela 5.2: Sinais de controle da UC do E_OU

Sinal	Tipo	Descrição
sinal_inicio	E	Dispara o processo de reconfiguração do DR-FPGA
clock_reconf	E	Relógio de reconfiguração da UC
sinal_reset_b	E	Reset da UC
sinal_fim	S	Indica o fim da reconfiguração
sinal_reconfigura	S	NL1 = reconfiguração em processo
sinal_ctrl_cena_00_b	S	Controla as chaves de isolação da cena 00
sinal_estado_cena_00	S	Indica a condição do processo de reconfiguração
sinal_ctrl_cena_01_b	S	Controla as chaves de isolação da cena 01
sinal_estado_cena_01	S	Indica a condição do processo de reconfiguração

receber os sinais de entrada e comporta-se como uma porta E.

Ao ser ativado o sinal_inicio o "FPGA" será reprogramado dinamicamente, para uma porta OU e o sinal_fim apontará o termino da reconfiguração. Uma nova afirmação do sinal_inicio levará o "FPGA" para a condição de uma porta E, uma nova ativação para uma porta OU e assim sucessivamente.

Convém ressaltar que o sinal de reset_b, da UC não esta ligado ao pino de $\overline{reset}$ do FPGA e ele ao receber a tensão de alimentação ($VDD > 2,1V$) inicia um ciclo de limpeza de configuração[67] o que o coloca em um estado seguro. Estando o FPGA em estado seguro a UC irá configura-lo, o que é iniciado a partir da afirmação do sinal de sinal_reset_b.

5.2.3 Diagrama de tempo E_OU

A simulação da reconfiguração dinâmica tem como base os valores extraídos dos arquivos de configuração e do leiaute do FPGA. A extração forneceu os arquivos:

porta_and.v, *porta_and.sdf*, *porta_or.v* e *porta_or.sdf* . O Figaro gerou os arquivos de *bitstream* que permitiram a obtenção dos tempos de configuração e reconfiguração.

Os parâmetros da simulação são sete e estão localizados no arquivo *reconfiguracao.v* (vide apêndice A). Na tabela 5.3 estão os valores, que multiplicados pelo T_CLK_RECONF fornecem o tempo de reconfiguração para cada um dos módulos utilizados na simulação. O valores do T_REMOCAO_00 e 01 foram ajustados para o tempo mínimo pois ambos os módulos ocupam a mesma área do FPGA.

Tabela 5.3: Parâmetros da simulação do E_OU

Parâmetro	Valor	Descrição	Observação
T_CLK_RECONF_MARCA	15	Relógio de configuração	Forma de onda simétrica com 30 ns de período
T_CLK_OPERACAO_MARCA	50	Relógio da arquitetura	Forma de onda simétrica com 1 μs
T_LOAD_RESET_00	37	Carga do módulo 00	*Bitstream* comprimido
T_LOAD_00	9	Carga da diferença para módulo 01	*Bitstream* comprimido
T_REMOCAO_00	1	Remoção do resíduo de hardware	
T_LOAD_01	9	Carga da diferença para módulo 00	*Bitstream* comprimido
T_REMOCAO_01	1	Remoção do resíduo de hardware	

Obs.: Para maiores informações veja a listagem fonte no apêndice A.

Nas figuras 5.6, 5.7 e 5.8 estão os resultados da pós-simulação do exemplo E_OU e dos aspectos internos da estrutura de simulação (ex: funcionamento das chaves de isolação).

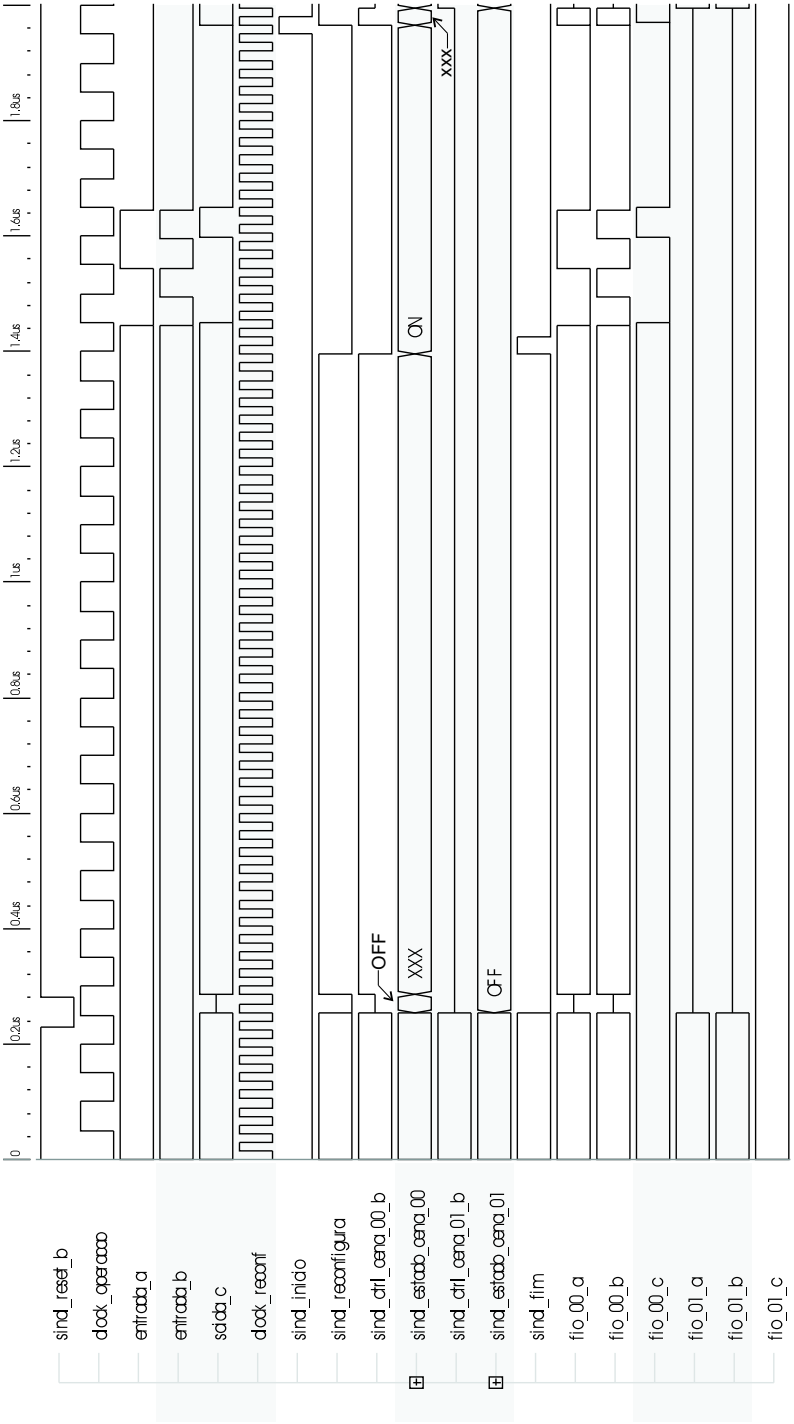

Figura 5.6: Diagrama de tempo E_OU - configuração porta E

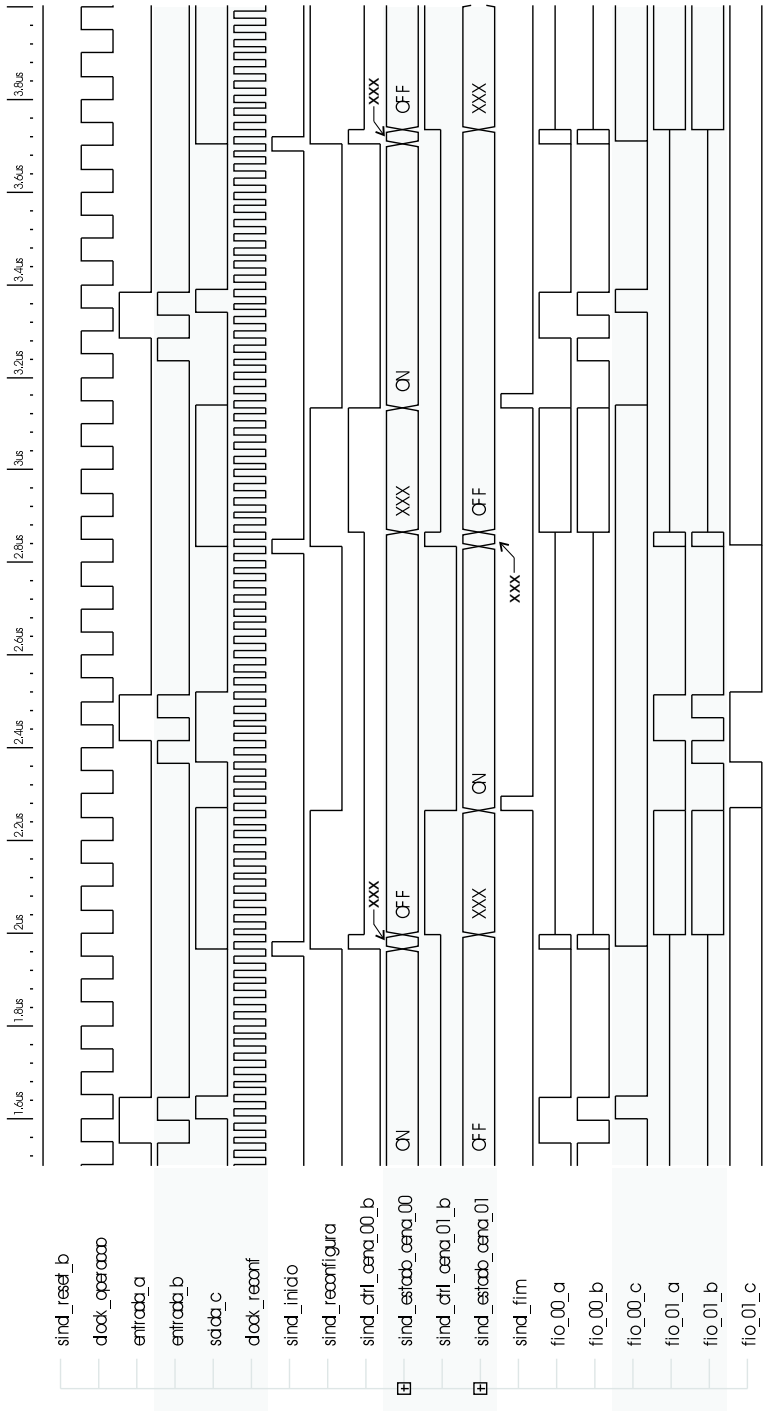

Figura 5.7: Diagrama de tempo E_OU - reconfiguração porta OU

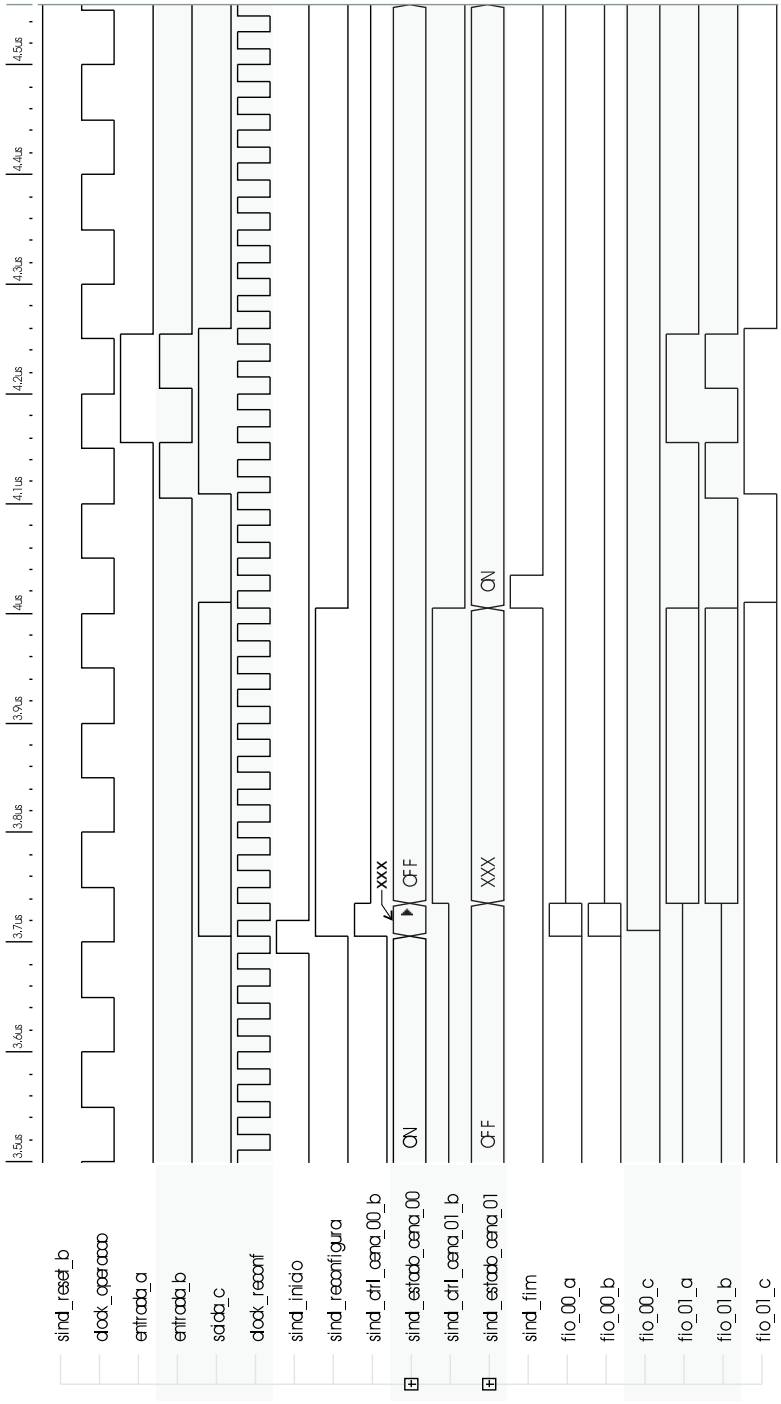

Figura 5.8: Diagrama de tempo E_OU - reconfiguração Porta E

5.3 Análise do impacto do tempo de reconfiguração

A avaliação quantitativa do *custo x beneficio* da reconfiguração dinâmica, em aplicações clássicas, pode ser feita utilizando as métricas sensíveis ao custo (equação 5.1).

$$Métrica\ sensivel\ ao\ custo = \frac{desempenho\ computacional}{medida\ de\ custo} \tag{5.1}$$

Wirthlin[35, 68] sugere que a métrica tenha como medida de custo a área (A) utilizada da pastilha e que em um FPGA ela corresponde ao número de células utilizadas. A métrica (equação 5.2) recebe o nome de densidade funcional (D), mede a vazão computacional (operações/s)[7] tem como fator de desempenho o inverso do tempo total de execução (T).

$$D = \frac{1}{AT} \tag{5.2}$$

A medida da densidade funcional nos FPGAs reconfiguráveis e parcialmente reconfiguráveis (equação 5.3) é função do tempo total execução e da área. O tempo de execução divide-se em duas parcelas: T_e que é o tempo de execução e o T_c que corresponde a soma dos tempos de reconfiguração entre as diversas cenas. Wirthlin assume que a área (medida em número de células) corresponde a maior área entre as diversas cenas, tendo em vista que as áreas livres não são aproveitáveis em FPGAs

[7] Na verdade o correto seria: operações/$(s \times m^2)$, mas Wirthlin[35, pág.248] em seu artigo original adotou operações/s. Acredita-se que a densidade funcional foi expressa em operações/s, pois a área utilizada em um FPGA pode ser medida pelo número de células utilizadas (número adimensional).

reconfiguráveis[68, Item 5.1.3, pág 51-52].

$$D = \frac{1}{A(T_e + T_c)} \tag{5.3}$$

Outro índice definido por Wirthlin é o de melhoria I e o $I_{máx}$ conforme definido nas equações 5.4 e 5.6 . Os parâmetros D_{estat} e D_{reconf} correspondem respectivamente a densidade funcional do circuito estático e do reconfigurável. O $I_{máx}$ indica o máximo benefício do uso da reconfiguração dinâmica sobre a estática, o que permite ao projetista uma medida indicativa do uso ou não da reconfiguração dinâmica[8].

$$I = \frac{\Delta D}{D_{estat}} = \frac{D_{reconf} - D_{estat}}{D_{estat}} = \frac{D_{reconf}}{D_{estat}} - 1 \tag{5.4}$$

Seja a taxa de configuração definida por $f = T_c/T_e$, então $T = T_e(1+f)$, aplicando na expressão da densidade funcional para o circuito de reconfiguração obtém-se a expressão:

$$D_{reconf} = \frac{1}{AT_e(1+f)} \tag{5.5}$$

pode-se calcular o valor de $D_{máx}$ da seguinte forma:

$$D_{máx} = \lim_{f \to 0} D_{reconf} = \frac{1}{AT_e}$$

[8]Convém lembrar que a medida de densidade funcional, utilizando $T = T_e + T_c$, não leva em conta a execução e reconfiguração em paralelo, isto é a sobreposição parcial ou total dos tempos[68, Item 7.2, pág 128] para uma dada aplicação.

e por fim obtém-se o índice de melhoria máxima que é dado pela expressão:

$$I_{máx} = \frac{D_{máx}}{D_{estat}} - 1 \tag{5.6}$$

Objetivando relacionar $I_{máx}$ com f pode-se escrever:

$$D_{reconf} > D_{estat}$$

substituindo as variáveis pelas suas expressões:

$$\frac{1}{A_{reconf}(T_{e\_reconf} + T_c)} > D_{estat}$$

$$\frac{1}{D_{estat} \cdot A_{reconf}} > T_{e\_reconf} + T_c$$

$$\frac{1}{D_{estat} \cdot A_{reconf}} > T_{e\_reconf} \cdot (1 + \frac{T_c}{T_{e\_reconf}})$$

$$\frac{1}{D_{estat}} \left(\frac{1}{A_{reconf} \cdot T_{e\_reconf}} \right) > 1 + \frac{T_c}{T_{e\_reconf}}$$

e identificando $D_{máx}$

$$\frac{D_{máx}}{D_{estat}} - 1 > \frac{T_c}{T_{e\_reconf}}$$

por fim obtém-se a relação:

$$f < I_{máx} \tag{5.7}$$

O procedimento adotado pode ser resumido nos seguintes passos:

1. Deve-se obter a area e o desempenho dos circuitos estático e reconfiguravel;
2. Calcular o valor do $I_{máx}$ e obter o valor da f;
3. Determinar o tempo de reconfiguração (T_c) do FPGA;
4. Caso $T_c < f \cdot T_{e\_reconf}$ então a abordagem de reconfiguração é uma boa candidata a resolução do problema.

Retomando o exemplo E_OU e tendo por base a análise qualitativa descrita acima fez-se necessário construir um equivalente estático do exemplo. O diagrama lógico pode ser visto na figura 5.9 e preferiu-se a captura do esquemático a uma HDL por questões de controle das otimizações já que o sintetizador consegue reduzir o circuito lógico a apenas uma célula[9]. O intuito desta abordagem é mostrar a medida da densidade funcional e não provar que a implementação utilizando reconfiguração dinâmica é melhor ou pior do que estática, haja visto o tamanho do circuito exemplo.

[9]Mesmo assim, foi necessário desabilitar o mapeamento tecnológico na ferramenta Figaro, pois a mesma fazia com que o circuito também ocupasse apenas uma célula.

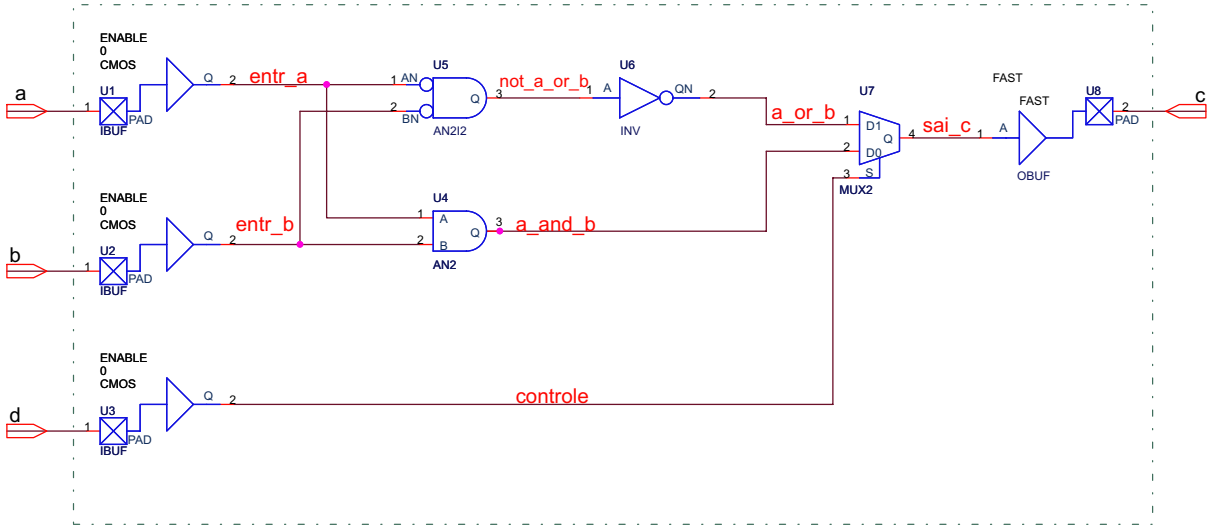

Figura 5.9: Esquema de uma porta E_OU estática

Nas figuras 5.10 e 5.11 encontra-se o resultados das pós-simulações dos projetos estático e dinâmico respectivamente.

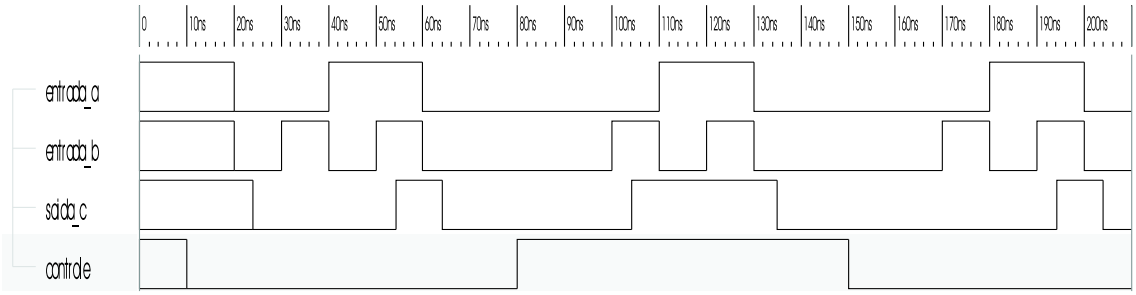

Figura 5.10: Pós-simulação da porta E_OU estática

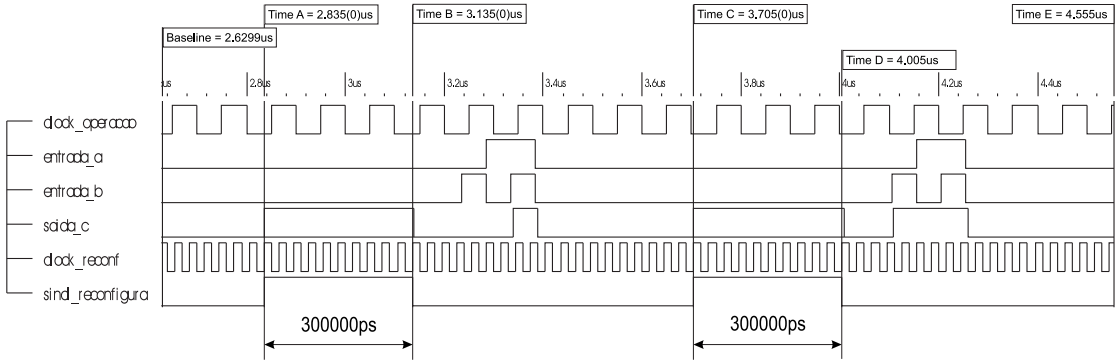

Figura 5.11: Pós-simulação do exemplo E_OU reconfigurável

Na tabela 5.4 estão consolidados os dados obtidos e os resultados obtidos.

Tabela 5.4: Resultados dos projetos do E_OU

Parâmetros e resultados	Estático	Reconfigurável	
		cena_00	cena_01
$T_e(ns)^*$	30,86	$6{,}32^\dagger$	5,74
$A(células)$	4	1	1
$D(operações/s)$	$8{,}10 \times 10^6$	$\frac{10^9}{12{,}06+Tc}$	
$D_{máx}(operações/s)$	$8{,}10 \times 10^6$	$82,92 \times 10^6$	
$I_{máx}$	0	$9{,}24 \rightarrow 924\%$	
$t_{CFG}(ns)$	2.100	1.110 $(reset \rightarrow 00)$	
$(t_{clock\_reconf} = 30\ ns)$		300 $(00 \rightarrow 01)$	300 $(01 \rightarrow 00)$

Obs.: No caso reconfigurável admite-se que o FPGA tenha sido "limpo" (*configuration clean cycle*) para receber um *bitstream compress*.

(∗) Para a execução de uma operação lógica E e outra OU.

(†) O dado é escrito diretamente na LUT.

Os tempos de atraso T_e, apresentados na tabela 5.4, foram obtidos dos relatórios gerados pela ferramenta Figaro. Eles são formados por duas partes:

- atrasos de roteamento: correspondente aos recursos de roteamento utilizados;
- atrasos lógicos: que incluem além das células lógicas também os introduzido pelos pinos de E/S,

e encontram-se desmembrados nas as equações 5.8, 5.9 e 5.10.

$$\begin{aligned} T_e\ estático &= T_p\ E + T_p\ Mux + T_p\ OU \\ &= [\ \underbrace{10,81}_{roteamento} + \underbrace{(5,23)}_{lógica}\ ] + [(3,33)] + [6,97 + (4,52)] \\ &= 30,86\ ns \end{aligned} \tag{5.8}$$

$$T_e\ cena\_00 = 1,21 + (5,11) = 6,32\ ns \tag{5.9}$$

$$T_e\ cena\_01 = 1,24 + (4,50) = 5,74\ ns \tag{5.10}$$

Pelo valor de $f < 9,24$ (equação 5.7) observa-se que potencialmente o E_OU reconfigurado pode alcançar um desempenho teórico de 924% ($f = 0$) sobre o projeto estático. O próximo passo é a obtenção da inequação do tempo de reconfiguração em funçao da taxa de configuração (f) que é dada por:

$$T_c < f \cdot T_{e\_reconf} = 9,24 \cdot (6,32 + 5,74) \times 10^{-9}$$

$$T_c < 111,43\ ns$$

Interpretando-se o resultado, conclui-se que para cada $12,06\ ns$ de tempo de execução do E_OU reconfigurável até $111,43\ ns$ ($D_{reconf} > D_{estat}$) pode ser gasto na

configuração do FPGA. Infelizmente o $T_{c_{AT40K}} = 600\ ns$ é $5,39$ vezes maior que o mínimo necessário para $D_{reconf} = D_{estat}$[10].

No gráfico 5.12 encontram-se alguns valores de referência para uma análise do impacto da f_{clock_reconf} sobre a taxa de reconfiguração (f) de um hipotético FPGA da família AT40K com freqüências maiores do que $33,33$ MHz .

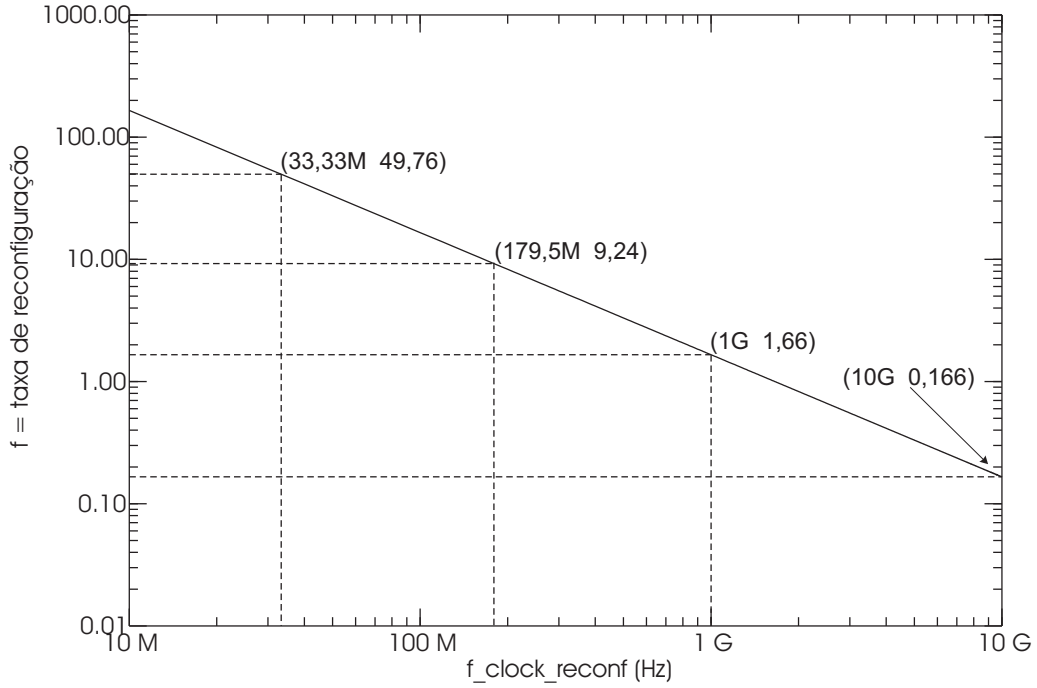

Figura 5.12: Taxa de reconfiguração x freqüência de reconfiguração

Pelos resultados obtidos verificou-se que existe uma grande vantagem (teórica) em se utilizar um DR-FPGA no experimento E_OU. Em contrapartida faz necessário que o mesmo suporte altas taxas de reconfiguração.

[10] $f_{clock\_reconf} = 179,5$ MHz

5.4 Experimento: MAC usando F.E.

Nesta seção é estudado o *problema das interconexões* entre os neurônios artificiais utilizando-se as F.E.

Uma RNA típica tem geralmente em seus neurônios uma estrutura de ponderação de pesos constituída por multiplicadores e acumuladores de n entradas (figura 1.1).

Suponha-se uma RNA *incompleta* de duas entradas, com uma camada de pesos e três unidades de processamento na camada escondida[11] (figura 5.13). Para o estudo do problema das interconexões pode-se desprezar a função de ativação (figura 1.1), pois ela representa um bloco em série com a estrutura de interconexão.

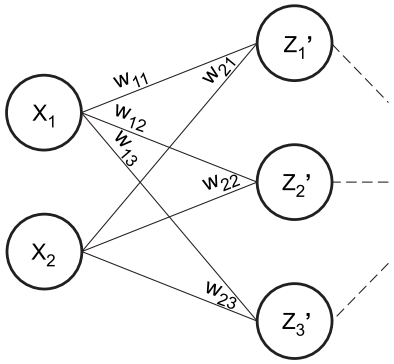

Figura 5.13: RNA com 2 entradas e 3 neurônios na camada intermediária

As equações de z_{j_in} serão descritas por z_j, como se segue:

$$\begin{cases} z_1 = x_1 \cdot w_{11} + x_2 \cdot w_{21} \\ z_2 = x_1 \cdot w_{12} + x_2 \cdot w_{22} \\ z_3 = x_1 \cdot w_{13} + x_2 \cdot w_{23} \end{cases} \tag{5.11}$$

[11] Camada de unidades de processamento que não fazem contato direto com o meio externo, isto é não ligadas a entradas e/ou saídas.

Uma das formas de se decompor em hardware a execução da RNA, utilizando as F.E., pode ser vista na figura 5.14. Nela a execução acontece da esquerda para direita e de cima para baixo. Inicialmente carregam-se os dados e as figuras de execução (multiplicações), a seguir elas são executadas resultando nos valores y_1 a y_6. Faz-se uma nova carga de figuras de execução[12] (somadores) e como resultado obtém-se os valores z_1 a z_3.

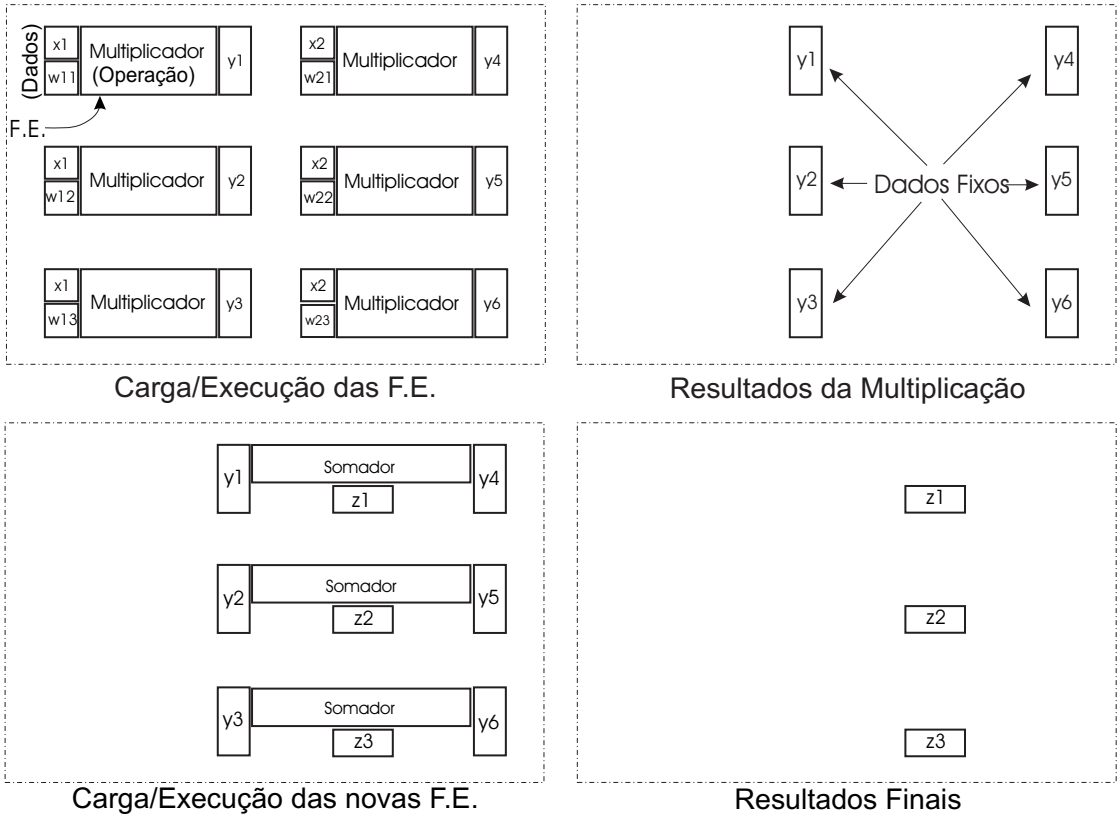

Figura 5.14: Seqüência dos atos do MAC utilizando F.E.

[12]Observe-se que os dados estão fixos e os operadores vão até ele pelo processo de reconfiguração do FPGA, resultando assim em uma computação sem transporte de dados.

5.4.1 Projeto e Simulação do MAC F.E.

Para facilitar o projeto e a simulação do MAC utilizando as F.E. (MAC F.E.), resolveu-se implementar apenas a equação 5.11 e estender a análise para as outras duas computações paralelas (z_2 e z_3) de forma teórica.

O projeto[13] do MAC simplificado (MAC1) foi feito em HDL[14] e decomposto em três partes principais (figura 5.15), denominadas *atos* e cada um deles foi sub-dividido em ações menores chamadas de *cenas* (figura 5.16).

Na decomposição do projeto procurou-se identificar:

- os *dados* que devem estar presentes nas células do FPGA, já que as F.E. não fazem uso de pinos para o transporte de dados;
- os *operadores* que darão origem as figuras de execução;
- a "passagem de dados" entre os módulos funcionais (intersecção das cena);
- o paralelismo dos operadores.

Durante o projeto verificou-se o correto chaveamento dos atos através da simulação da reconfiguração dinâmica (DCS). As chaves de isolação devem ser colocadas entre as entradas e saídas de dados de cada módulo funcional.

[13]No apêndice B descreve-se a metodologia básica adotada para um projeto RTR utilizando a ferramenta Figaro da Atmel.

[14]Os programas fontes encontram-se no apêndice C.

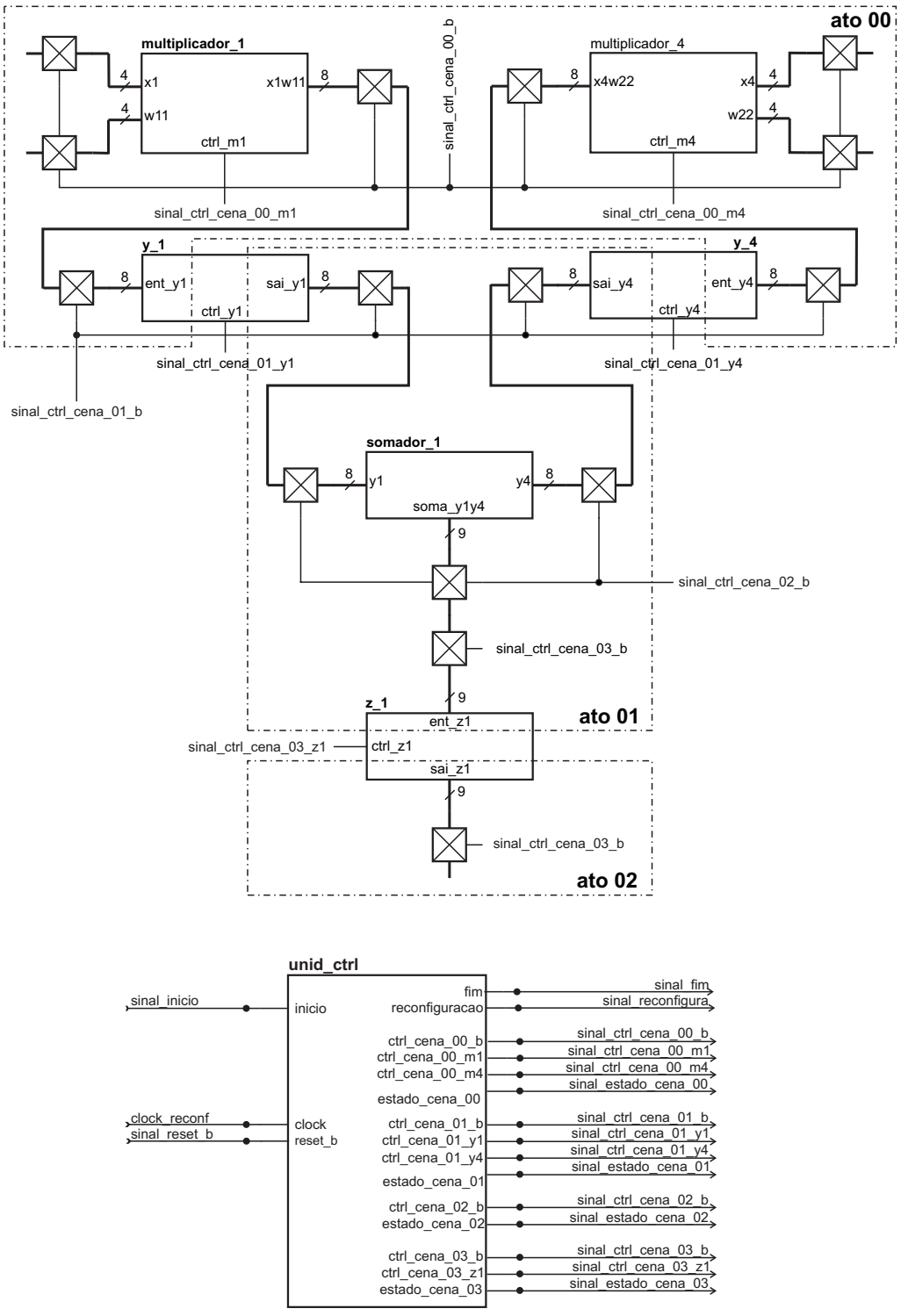

Figura 5.15: Diagrama em blocos dos atos do caso MAC1 utilizando F.E. (Veja a linha 13 do arquivo *reconfiguracao.v* no apêndice C).

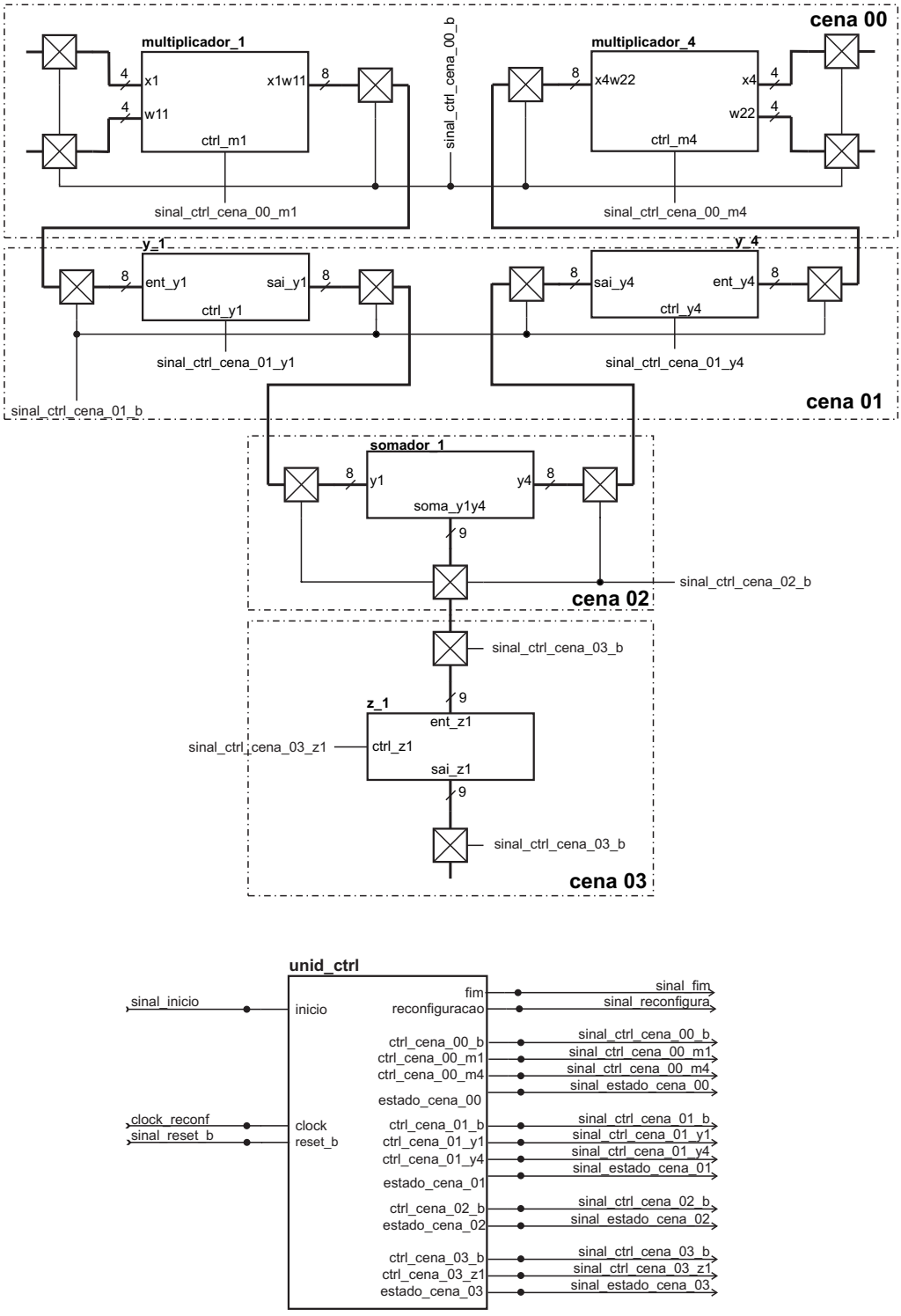

Figura 5.16: Diagrama em blocos das cenas do caso MAC1 utilizando F.E. (Veja a linha 13 do arquivo *reconfiguracao.v* no apêndice C).

Por exemplo na figura 5.15 pode-se ver uma chave de isolação conectada as entradas e saídas do multiplicador_1 e convém notar que as chaves de isolação de $x1$ e $w11$ não possuem entradas, sendo as mesmas utilizadas apenas durante a simulação.

Outro detalhe importante de projeto e o uso das barras de dados (*data bar*) y_1, y_4 e z_1 das cena 01 e 03 (figuras 5.16 e 5.15) em dois atos, ato 00 & 01 e 01 & 02. Isso deve-se ao fato que no uso da DCS tem-se como premissa conjuntos de tarefas mutuamente exclusivas (seção 5.1.1). Para contornar o problema adotou-se uma "granularidade" de cena, o que faz com que a escrita e leitura, nas barras de dados, sejam mutuamente exclusivas permitindo assim a simulação da reconfiguração dinâmica.

Os programa principal da simulação (figura 5.17) é o *reconfiguracao.v* que chama os módulos *chave_isolação{4, 8 e 9}.v, cena_{00, 01, 02 e 03}.v e unid_ctrl.v* . A unidade de controle é responsável por gerenciar toda a simulação, incluindo a *carga* dos atos e cenas através das chaves de isolação e pode ser configurada através de parâmetros (tabela 5.5). O valores dos parâmetros dos tempos de carga e remoção das cenas são múltiplos do T_CLK_RECONF e foram obtidos dos arquivos de *bitstream* gerados pela ferramenta Figaro.

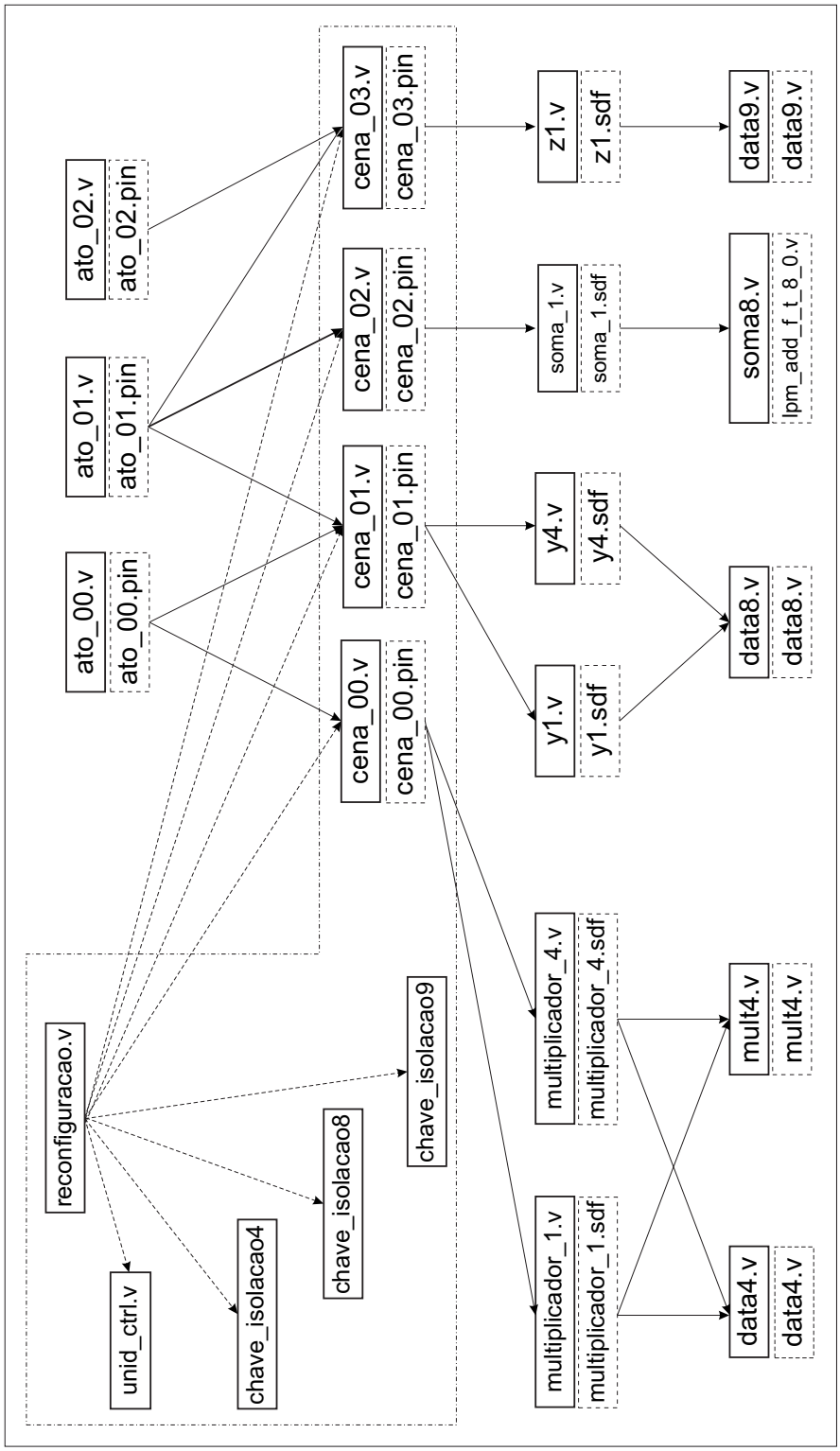

Figura 5.17: Árvore HDL do MAC1 utilizando F.E.

Tabela 5.5: Parâmetros de simulação do MAC1 F.E.

Parâmetro	Valor	Descrição
T_CLK_RECONF_MARCA	15	Relógio de configuração (período = 30 ns)
T_CLK_OPERACAO_MARCA	50	Relógio da arquitetura (período = 1 μs)
T_LOAD_RESET_00	428	Carga da cena 00 (reset).
T_LOAD_00	428	Carga da cena 00
T_REMOCAO_00	428	Remoção da cena 00
T_LOAD_01	84	Carga da cena 01
T_REMOCAO_01	84	Remoção da cena 01
T_LOAD_02	144	Carga da cena 02
T_REMOCAO_02	144	Remoção da cena 02
T_LOAD_03	49	Carga da cena 03
T_REMOCAO_03	49	Remoção da cena 03

Obs.: Admitiu-se por questões de limitação da ferramenta Figaro, que $T\_LOAD = T\_REMOCAO$ e *bitstream* sem opção de carga das diferenças (*windows bitstream*).

Um dos problemas encontrados, durante o projeto do MAC1 F.E., foi a determinação do tamanho dos arquivos de configuração das cenas e a sua otimização (vide tabela 5.6). Convém ressaltar que a ferramenta Figaro não é capaz de gerar *bitstream* de macros[15], somente de leiautes com pinos. Tal limitação faz com que os *bitstreams* incluam a programação dos pinos e os *eventuais* recursos de interconexão necessários para interligar os pinos as células do FPGA. Os *bitstreams* foram gerados com a opção *compress*[16] e sem redução de potência durante a (re)configuração, foram retiradas *manualmente* as linhas correspondentes a programação de pino de E/S e as de interconexão dos pinos E/S às células do FPGA.

Na (tabela 5.7) encontram-se descritos os sinais da unidade de controle que permitem o controle da arquitetura e também indicam o estado da reconfiguração das cenas.

[15]Macro é um *parte* de um projeto ou bloco funcional, previamente otimizado e que é armazenado em uma biblioteca (vide apêndice B).

[16]Admitiu-se o FPGA em estado limpo, isto é após um *configuration clean cycle*.

Tabela 5.6: Tamanho dos arquivos de reconfiguração do MAC1 F.E.

Cenas	BC	+ SRP	+ SPES + SR	+ FIOS	Final (linhas)
00	1.309	885	364	$4\ linhas \times (8+8)\ fios$	428
01	934	719	84	0	84
02	795	630	44	$4\ linhas \times (8+8+9)\ fios$	144
03	643	488	49	0	49

Legenda:
BC = *bitstream* comprimido,
SRP = sem a opção de redução de potência durante a reconfiguração,
SPES = sem pinos de E/S,
SR = sem repetidores horizontais, verticais, *route wire* e vias externos as F.E. ,
FIOS = número de fios que interligam o resultado de uma cena ao data_bar da outra.

Tabela 5.7: Sinais de controle da UC do MAC1 F.E.

Sinal	Tipo	Descrição
sinal_inicio	E	Dispara o processo de reconfiguração do DR-FPGA
clock_reconf	E	Relógio de reconfiguração da UC
sinal_reset_b	E	Reset da UC
sinal_fim	S	Indica o fim da reconfiguração
sinal_reconfigura	S	NL1 = reconfiguração em processo
sinal_ctrl_cena_00_b	S	Controla as chaves de isolação da cena 00
sinal_ctrl_cena_00_m1	S	NL0 = armazena x_1 e w_{11}
sinal_ctrl_cena_00_m4	S	NL0 = armazena x_2 e w_{21}
sinal_ctrl_cena_01_b	S	Controla as chaves de isolação da cena 01
sinal_ctrl_cena_01_y1	S	NL0 = armazena y_1
sinal_ctrl_cena_01_y1	S	NL0 = armazena y_4
sinal_ctrl_cena_02_b	S	Controla as chaves de isolação da cena 02
sinal_ctrl_cena_03_b	S	Controla as chaves de isolação da cena 03
sinal_ctrl_cena_03_z1	S	NL0 = armazena z_1
sinal_estado_cena_00	S	Estado da cena_00
sinal_estado_cena_01	S	Estado da cena_01
sinal_estado_cena_02	S	Estado da cena_02
sinal_estado_cena_03	S	Estado da cena_03

5.4.2 Resultados da simulação do MAC1 F.E.

Foram feitas várias simulações para validar e analisar o comportamento do MAC1, utilizando as F.E. As figuras 5.18 e 5.19 a 5.21 fornecem uma visão geral e detalhada da pos-simulação, respectivamente e com os tempos de atraso, das interconexões e das reconfigurações reais do *experimento*.

Nas figuras mencionadas é executada a seguinte equação: $z_1 = x_1 \cdot w_{11} + x_2 \cdot w_{21}$ para os seguintes casos:

1-) $x_1 = 2$, $w_{11} = 3$, $x_2 = 4$, $w_{21} = 5$

2-) $x_1 = 5$, $w_{11} = 7$, $x_2 = F$, $w_{21} = 3$

nas figuras detalhadas pode-se verificar o chaveamento correto dos atos e conseqüentemente das cenas.

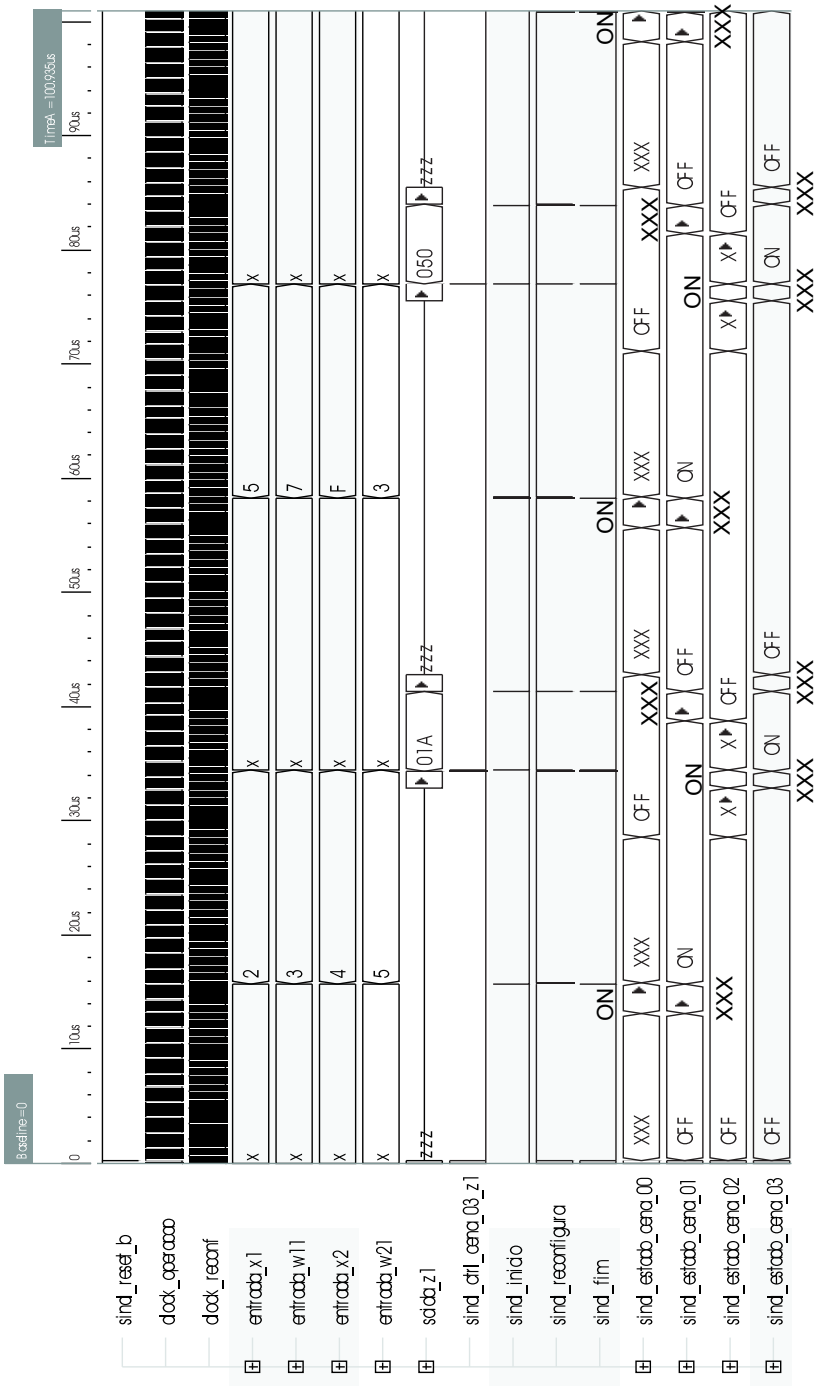

Figura 5.18: Visão geral da pós-simulação do MAC1 F.E.

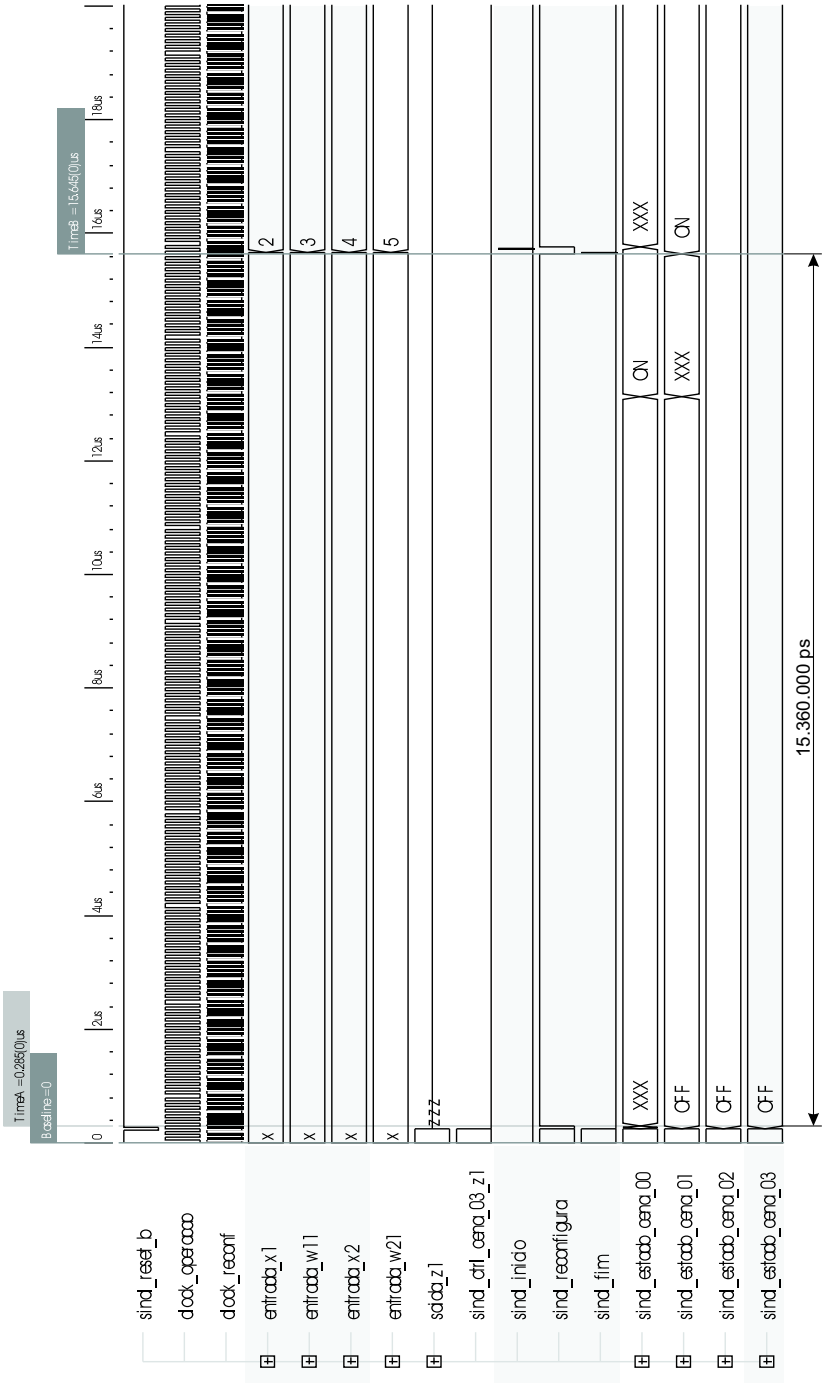

Figura 5.19: Detalhe da pós-simulação de 0 a 20 μs do MAC1 F.E.

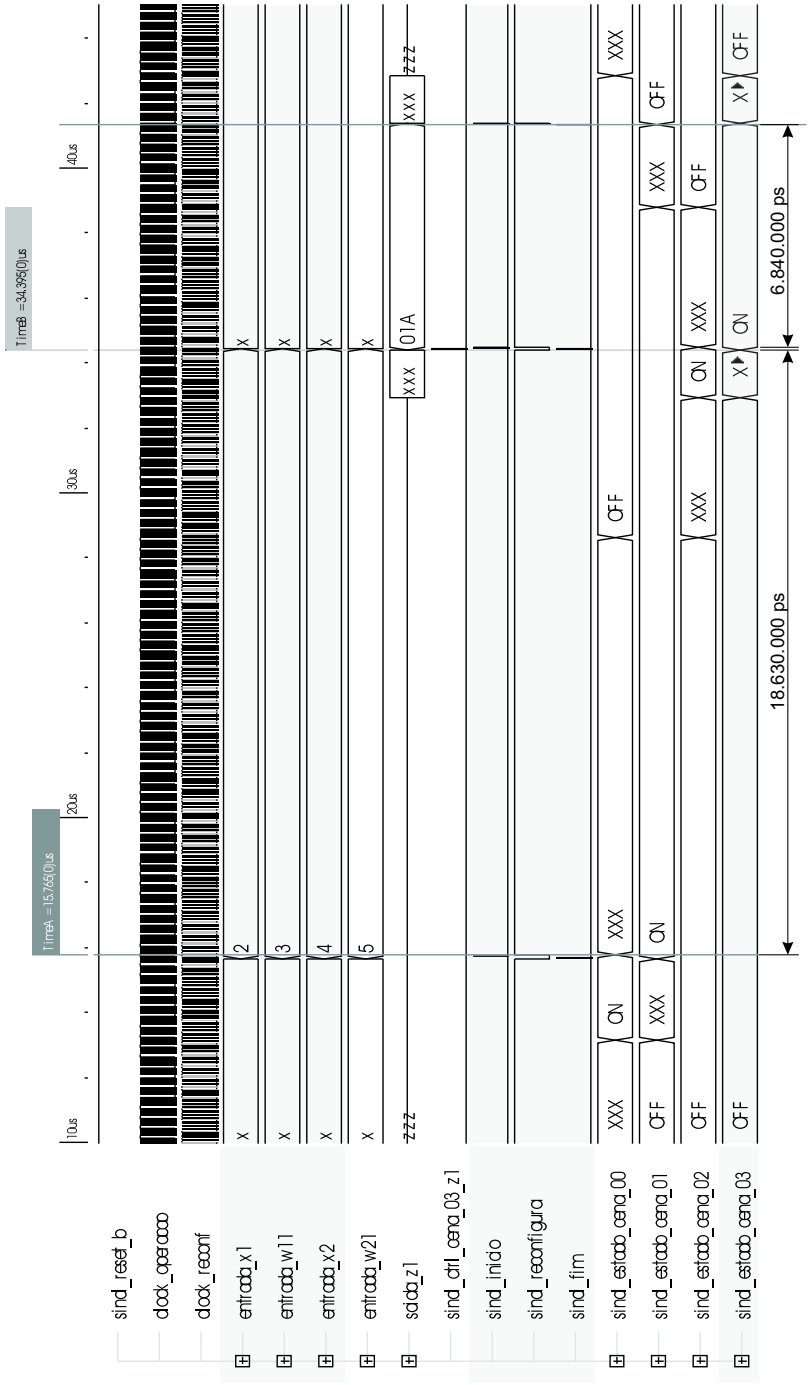

Figura 5.20: Detalhe da pós-simulação de 10 μs a 45 μs do MAC1 F.E.

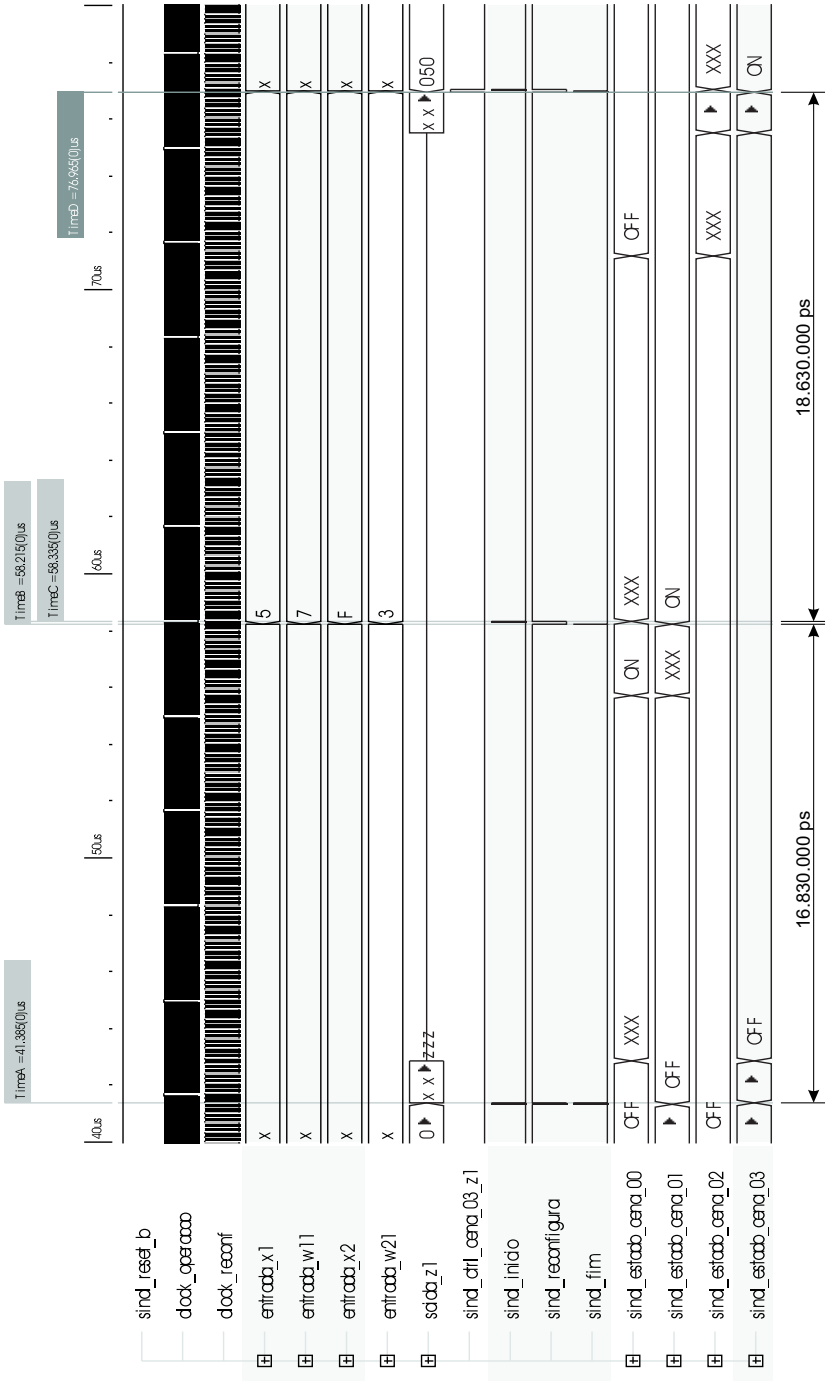

Figura 5.21: Detalhe da pós-simulação de 40 μs a 80 μs do MAC1 F.E.

5.4.3 *Implementação* do MAC

Foram feitos duas *implementações*[17] do MAC, uma utilizando o conceito de F.E. e outra com o método tradicional visando a comparação das mesmas.

A implementação do MAC utilizando F.E. segue a árvore HDL da figura 5.17 tendo como raízes os arquivos *ato_{00, 01 e 02}.v* . Na figura 5.22 encontra-se os leiautes do posicionamento e roteamento no FPGA, convém ressaltar que os arquivos *ato_{00, 01 e 02}.v* foram cada um deles instanciados 3 vezes para que fosse geradas as 3 equações necessárias (seção 5.4) e que os pinos de E/S e controle foram *unrouted*[18] de forma a mostrar o roteamento das barras de dados e das figuras de execução.

A implementação tradicional[19] pode ser vista na primeira metade superior da figura 5.23 e na parte inferior da figura 5.23 mostra-se o impacto das conexões de E/S retiradas do MAC utilizando as F.E. e o custo associado do MAC tradicional das ligações multiplicadores-somadores devido ao transporte de dados.

[17] Utilizou-se a palavra implementação, no sentido que o projeto foi implementado em um FPGA real e não apenas simulado em HDL. Convém ressaltar que o mesmo não foi montado em hardware.

[18] A ferramenta Figaro, não gera *bitstream* de projetos semi-roteados e sem pinos de E/S ou *netlists* sem conexões externas.

A ferramenta de síntese LeonardoSpectrum não foi capaz de sintetizar projetos com módulos isolados dentro de uma declaração de módulo Verilog que não tivesse ligações com os *ports* do modulo ou entre os seus elementos.

[19] Esta implementação não é única e pode ser otimizada. Ela deve ser vista como um modelo de estudo.

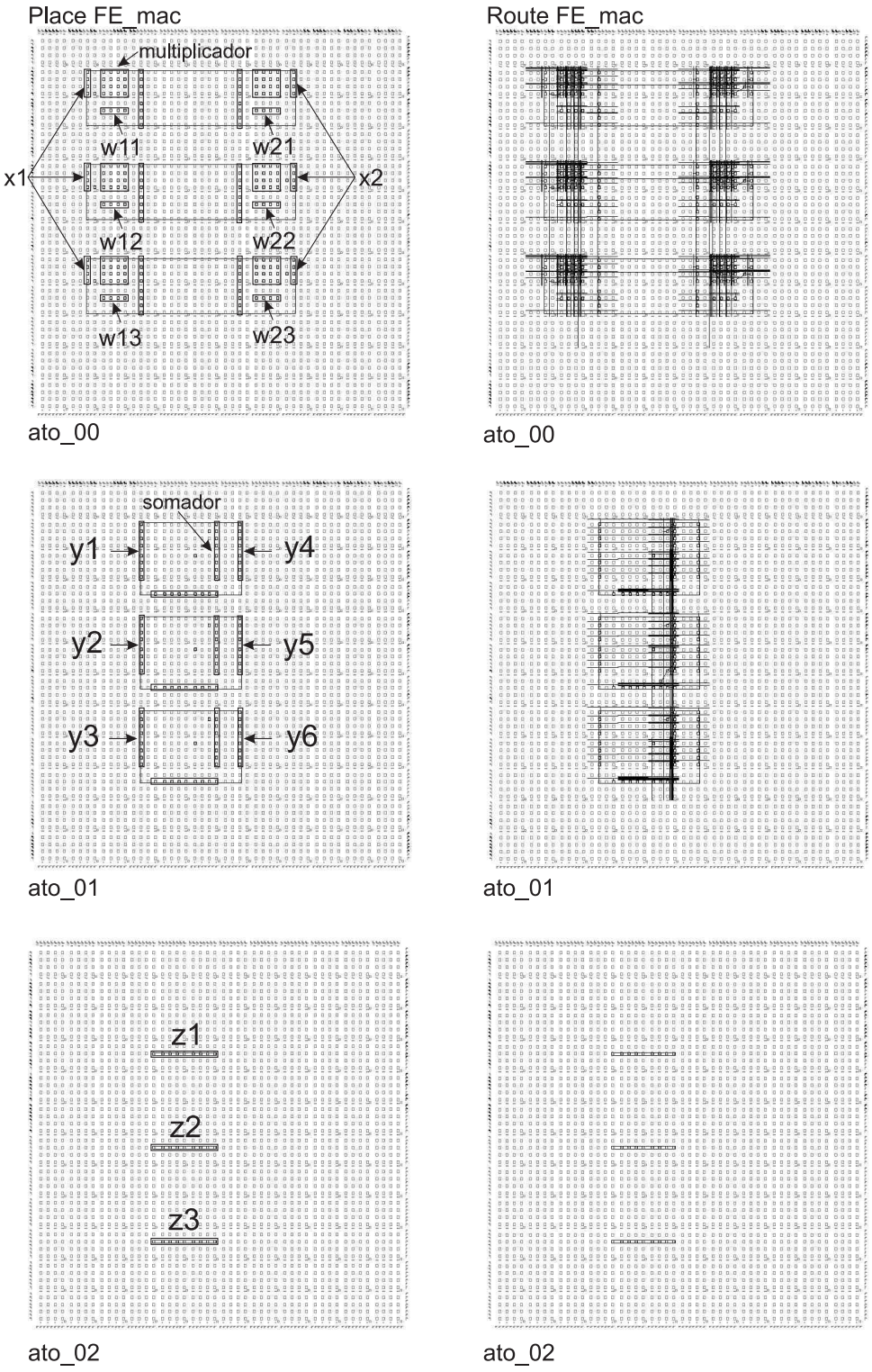

Figura 5.22: Leiaute do MAC utilizando F.E.

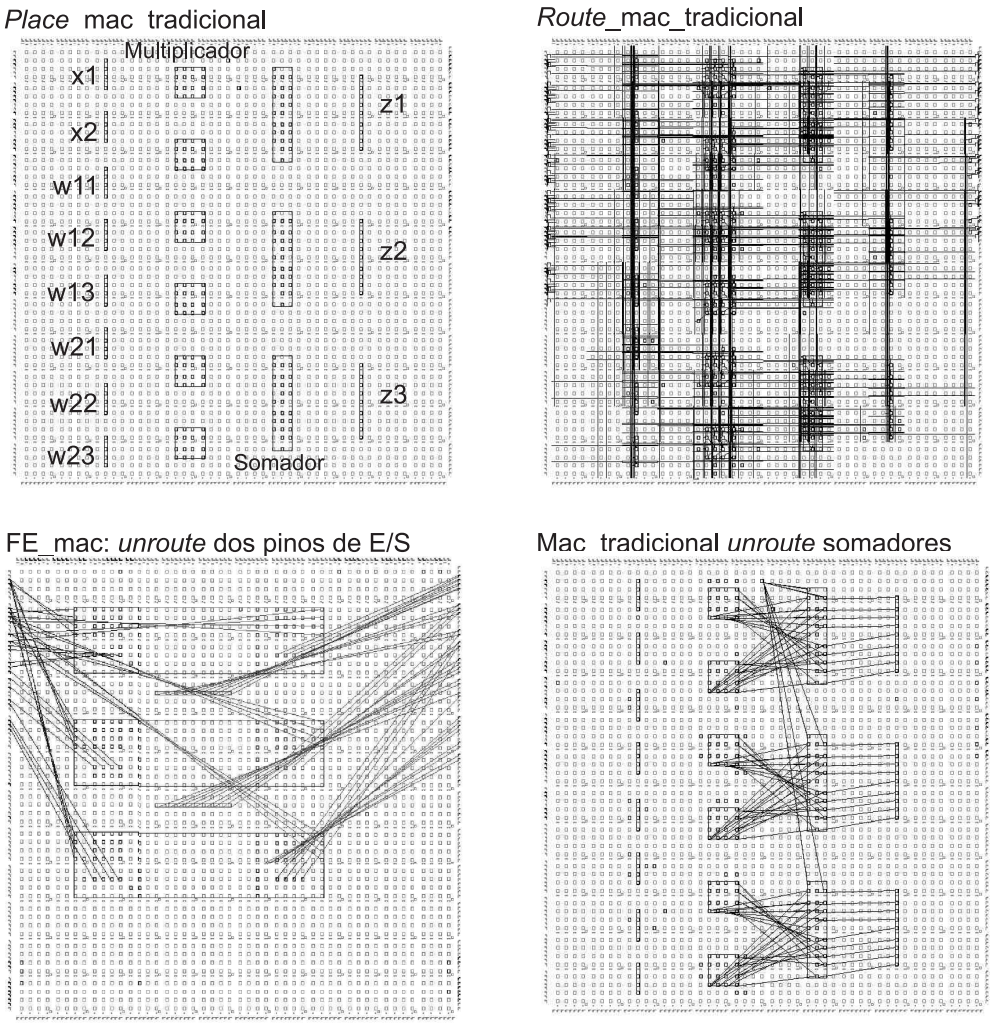

Figura 5.23: Leiaute do MAC tradicional e comparativo de roteamento

Nos exemplos a implementação MAC F.E.(figura 5.22) utilizou em seus 3 atos $126 + 73 + 0 = 199$ vias enquanto a implementação do MAC tradicional (figura 5.23) consumiu 1.125 vias e 59 pinos de E/S. As F.E. por não transportarem dados por vias, economizam, teoricamente, cerca de 82,31% dos recursos de interconexões do FPGA.

5.4.4 Análise da reconfiguração e CPS

Na tabela 5.8 foram consolidados os dados para a análise da reconfiguração dinâmica. Foram determinados os parâmetros de densidade funcional para ambos os casos, chegando-se a um $f < 0,4633$ (equação 5.7), o que indica que o projeto reconfigurável, pelo menos teoricamente, poderá chegar a um desempenho de $46,33\%$ maior do que o estático.

Tabela 5.8: Resultados do projeto MAC

Parâmetros e resultados	Estático	Reconfigurável		
		ato_00	ato_01	ato_02
$T_e(ns)$	76,08	35,92	31,3	2,1
$A(células)$	264	198	109	28
$D(operações/s)$	$49,79 \times 10^3$	$\frac{10^9}{198 \cdot (69,32 + Tc)}$		
$D_{máx}(operações/s)$	$49,79 \times 10^3$	$72,86 \times 10^3$		
$I_{máx}$	0	0,4633 $\rightarrow$ 46,33%		
$t_{CFG}(ns)$	126.870	46.080 ($reset \rightarrow 00$)		
($t_{clock\_reconf} = 30\ ns$)		50.490 ($00 \rightarrow 01$)	55.890 ($01 \rightarrow 02$)	20.520 ($02 \rightarrow 00$)
$Conexões\ por\ segundo\ (CPS)$	78,864 M	47,255 k ($t_{clock\_reconf} = 30\ ns$)		

Obs.: No caso reconfigurável admite-se que o FPGA tenha sido "limpo" (*configuration clean cycle*) para receber um *bitstream compress*.

Convém citar que os tempos de execução dos atos, do projeto reconfigurável, foram obtidos dos relatórios do Figaro (`palco.sts`) quando do projeto das macros ato_00, ato_01 e ato_02. Já os tempos de reconfiguração são os resultados do simulador (figuras 5.19 a 5.21) e posteriormente multiplicados pelo número de instâncias dos atos (no caso 3). Finalmente a área dos atos foi obtida através do relatório gerado pelo Figaro

do posicionamento e roteamento dos atos[20].

O proximo passo é a avaliação da expressão:

$$T_c < f \cdot T_{e\_reconf} = 0,4633 \cdot (35,92 + 31,3 + 2,1) \times 10^{-9}$$

$$T_c < 32,12\ ns$$

isto é, para cada $69,32\ ns$ de tempo de execução até $32,12\ ns$ pode ser gasto com a reconfiguração do FPGA, para que $D_{reconf} > D_{estat}$. O tempo total de reconfiguração do FPGA é de 126.900 ns o que é 3.951 vezes maior que o necessário para $D_{reconf} = D_{estat}$[21].

Pelos resultados obtidos nota-se que existe uma certa vantagem (46,33%) em se utilizar reconfiguração dinâmica neste experimento, mas torna-se inviável devido ao tempo de reconfiguração necessário do FPGA.

Os resultados invalidam o experimento MAC F.E.[22] para aplicações numéricas intensivas, visando aumento de densidade funcional (métrica utilizada), mas não invalidam as F.E.

[20] O arquivo gerado para instânciar os 3 atos encontra-se em C.1.6.
[21] $f_{clock\_reconf} = 131,69$ GHz.
[22] Utilizando os DR-FPGAs atualmente disponíveis no mercado.

Obtenção do CPS

Procurou-se estimar o desempenho de uma RNA de camada única, utilizando como base as unidades MAC[8]e obter o seu desempenho utilizando as implementações estática e as F.E.

O CPS do projeto MAC foram calculados utilizando a equação 5.12 e os resultados colocados na tabela 5.8. Abaixo são mostrados os cálculos realizados para cada um dos casos, bem como o cálculo do CPS máximo, para o caso utilizando as F.E.

$$CPS = \frac{número\ de\ conexões\ da\ RNA}{Tempo\ de\ execucão} \tag{5.12}$$

Cálculo do CPS para o projeto MAC estático:

$$CPS_{estático} = \frac{2\ entradas \times 3\ neuronios}{76,08 \times 10^{-9}} = 78,864\ MCPS$$

Cálculos do CPS para os MAC utilizando as F.E. e com $t_{clock\_reconf} = 30\ ns$.

$$CPS_{F.E.} = \frac{(2\ entradas \times 3\ neuronios) \times 10^{-9}}{35,92 + (50.490) + 31,3 + (55.890) + 2,1 + (20.520)} = 47,255\ kCPS$$

Cálculo do CPS máximo utilizando as F.E. ($t_{reconf} = 0\ ns$).

$$CPS_{F.E._{máx}} = \frac{(2\ entradas \times 3\ neuronios) \times 10^{-9}}{35,92 + 31,3 + 2,1 + (0)} = 86,555\ MCPS$$

Ao se comparar os resultados dos dois experimentos, verifica-se o que o tempo de reconfiguração possui um *grande* impacto sobre o desempenho.

Considerando-se o $CPS_{estático} \times CPS_{F.E._{máx}}$ obtém-se um ganho de apenas $9,75\%$ no desempenho, sobre o caso estático.

Convém destacar que enquanto o CPS aumenta de $9,75\%$ o $I_{máx}$ pode chegar a $46,33\%$, sendo que o primeiro índice está relacionado com a *velocidade* e o segundo ressaltou um melhor aproveitamento da área do FPGA.

Considerações gerais sobre a reconfiguração

As F.E. possibilitam a exploração do paralelismo e adjacências das figuras de execução, a semelhança das redes neuronais naturais, que possuem um tempo de propagação da ordem de alguns milisegundos e 10^{10} e 10^{12} de unidades funcionais (neurônios).

Suponha-se que as F.E. estejam limitadas a "planos de execução" no tempo (vide figura 5.24). Pode-se imaginar um pseudo-processador[23] com um tempo de busca/interpretação de instruções muito maior que a sua fase de execução. O tempo de ciclo da instrução será da ordem do tempo de um microprocessador com arquitetura dos finais dos anos 70 (figura 5.25), onde os mesmos não possuíam nenhuma técnica de redução da latência[24], do acoplamento processador-memória[69–71], tais como: memoria cache, pipeline de instruções e outras[25].

[23] Máquina SIMD reconfigurável.
[24] *Latency-hiding*
[25] Nesta análise não se incluem os *mainframes* da época.

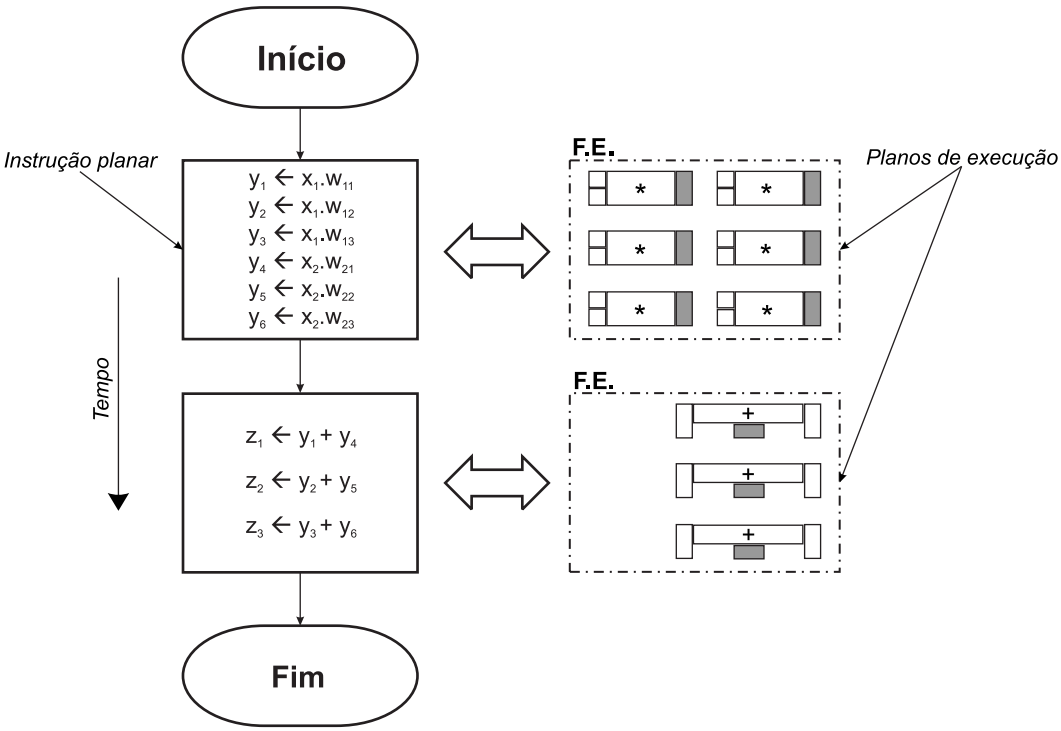

Figura 5.24: Fluxograma das F.E. interpretadas como instruções

- **AT40K @ 33,33 MHz : uma célula configurada como inversor.**

$t_{ciclo} = t_{execução} + t_{reconfiguração}$

$t_{ciclo} = 1{,}55 + \text{tamanho\_do\_vetor\_de\_configuração} \times t_{relógio\_reconfiguração}$ (ns)

$t_{ciclo} = 1{,}55 + 5 \times 30$ (ns)

$\boxed{t_{ciclo} = 151{,}55 \text{ ns}}$

- **Z80 @ 20 MHz : execução de uma instrução NOP**

$t_{ciclo} = T_{states} \times t_{relógio}$

$t_{ciclo} = 4 \times 50$ (ns)

$\boxed{t_{ciclo} = 200 \text{ ns}}$

Obs: a versão do Z80 @ 20 MHz foi anunciada em 1990.

Figura 5.25: Comparação entre F.E. em um AT40K e um microprocessador Z80

O problema *não está* nas F.E. e sim no manuseio e redução da latência do FPGA/circuito de reconfiguração[26]. Em certas aplicações por exemplo: processamento numérico intensivo, busca e/ou ordenação de tabelas, etc. certos trechos de programa pode ser acelerados utilizando algoritmos e/ou operações implementadas diretamente no hardware. Estas aplicações possuem um processador e um FPGA reconfigurável funcionando como co-processador ou unidade de processamento *standalone* . Neste contexto existem várias formas de redução do tempo de latência[72, 73] onde pode-se destacar:

- *Prefetching da configuração*: aproveita-se o tempo de execução do processador para reconfigurar o FPGA;
- *Compressão da configuração*: em algumas arquitetura existe o conceito de endereço imagem[27], o que permite programar varias partes do FPGA utilizando um único vetor de configuração (*endereço + dado*);
- *Cache da configuração*: Memória de programação com tempo de acesso bem menor do que a do processador o que permite um aumento da taxa de reconfiguração.

[26] Interno e externo do FPGA.
[27] Alguns bits do endereço são marcados como irrelevantes.

5.4.5 Estudo do impacto das interconexões

Nesta seção é analisado o impacto das interconexões, levando-se em consideração o crescimento das entradas e do número de neurônios de uma RNA. Para facilitar a análise estudo-se apenas a camada de entrada de uma RNA do tipo *Perceptron*.

Para o estudo do crescimento das interconexões, faz-se necessário obter a área de roteamento (A_{rot}), o tempo de atraso de roteamento (T_{rot}) e o número de interconexões(I_{nter}) utilizados do FPGA[28] dos projetos convencional e os baseados nas F.E. Com bases nestes dados calcula-se o fator de impacto das interconexões (FIC), que é dado pela seguinte métrica[29]:

$$FIC = \frac{1}{A_{rot} \cdot T_{rot} \cdot I_{nter}} \tag{5.13}$$

Sendo que o valor de Impacto Máximo das Interconexões (II_{max}) é dado por

$$II_{max} = \frac{FIC_{F.E.}}{FIC_{Est}} - 1 \tag{5.14}$$

Os projetos utilizados estão na figura 5.26 e servirão de base para a análise. Identificou-se os núcleos de cada um deles: o bloco multiplicador tradicional e bloco multiplicador utilizando as F.E., sendo que cada um deles será escalonado pelo número de entradas (n) e pelo número de neurônios da camada (m).

[28]No caso da ferramenta figaro obtém-se os dados dos arquivos: $*.sts$ (*number of route wire* = A_{rot} e *number of buses* = I_{nter}) e do $*.pdl$ (*route delay* = T_{rot})

[29]Proposta pelo autor.

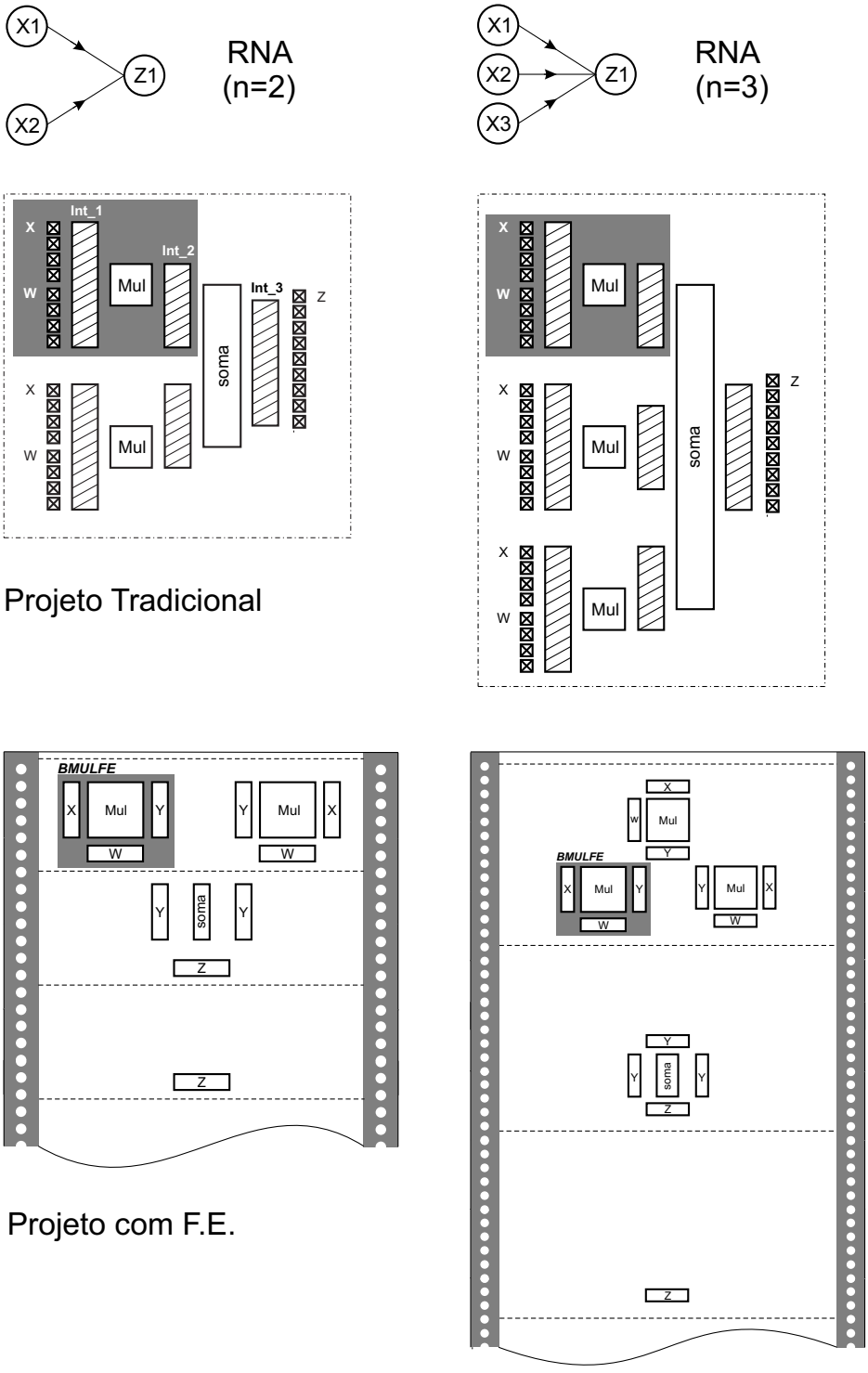

Figura 5.26: Implementação de RNAs utilizando metodologia tradicional e F.E.

Experimento utilizando a metodologia tradicional

Foram feitos vários experimentos, variando o valor de m e n, utilizando multiplicadores de 4 bits[30] ligados a pinos de entrada e saída do FPGA. Os resultados obtidos (vide exemplo no apêndice C.3.1) encontram-se na tabela 5.9, neles também foram incluídos os números de pinos utilizados do FPGA e o valor do FIC_{Est}.

A tabela 5.9 foi encerrada em $m = 3$ e $n = 7$, pois para $n = 8$ são necessário 320 pinos de E/S do FPGA. O maior FPGA disponível na ferramenta Figaro é o AT40K40-2BGC que possui 289 pinos sendo insuficientes para o experimento. Poder-se-ia particionar o projeto em múltiplos FPGAs (*multi-chip partitioning*), mas isso se mostrou desnecessário para a obtenção das familias de curvas.

Experimento utilizando F.E.

Considerando o projeto F.E. para um multiplicador de 4 bits, $m = 1$ e $n = 2$, os valores da área de roteamento, atraso e o número de interconexões (vide apêndice C.3.2) obtém-se a equação 5.16, a partir da equação 5.13 .

$$FIC_{F.E.} = \frac{1}{(\frac{5 \text{ células}}{2} \cdot mn) \cdot (19,94 \times 10^{-9} \text{ s}) \cdot (\frac{75 \text{ vias}}{2} \cdot mn)} \tag{5.15}$$

$$FIC_{F.E.} = \frac{10^6}{1,869375 \cdot m^2 n^2} \tag{5.16}$$

[30]Usado apenas como *exemplo*, pois o número de bits é insuficiente para os cálculos[19] da fase de propagação de uma RNA.

Os valor de $FIC_{F.E.}$ foram incluído na tabela 5.9, bem como o valor do II_{max} em porcentagem.

Tabela 5.9: Resultados do estudo da camada de entrada de uma RNA

m	n	A_rot(células)	T_rot(ns)	Inter.	Pinos	FIC_{Est}	$FIC_{F.E.}$	II_{max} (%)
1	2	6	21,9	99	32	76.662	133.734	74
	3	10	18,7	177	48	30.099	59.437	97
	4	15	16,4	160	64	25.375	33.433	**31**
	5	19	20,2	296	80	8.802	21.397	143
	6	24	22,2	349	96	5.360	14.859	177
	7	39	21,0	281	112	4.334	10.917	151
	8	34	19,0	362	128	4.262	8.358	96
	9	34	23,3	482	144	2.613	6.604	152
	10	40	17,4	488	160	2.937	5.349	82
2	2	16	26,1	204	56	11.720	33.433	185
	3	24	29,3	265	84	5.353	14.859	177
	4	31	24,3	406	112	3.262	8.358	156
	5	39	27,2	643	140	1.461	5.349	265
	6	47	22,5	560	168	1.687	3.714	120
	7	54	25,9	781	196	912	2.729	199
	8	48	27,9	911	224	817	2.089	155
	9	73	26,2	996	252	523	1.651	215
	10	73	25,8	1.043	280	507	1.337	163
3	2	22	25,4	340	80	5.246	14.859	183
	3	23	27,3	518	120	3.064	6.604	115
	4	47	30,2	689	160	1.020	3.714	263
	5	54	32,5	890	200	639	2.377	271
	6	83	28,6	981	240	429	1.651	**284**
	7	71	32,0	1.176	280	374	1.213	224

Observações:
- ◇ A coluna PINOS inclui os pinos criados pelos resultados das multiplicações;
- ◇ O somador não foi implementado, apenas a *aproximação* de suas conexões;
- ◇ Foi deixada à ferramenta FIGARO a escolha da posição física dos pinos de E/S.

Conforme a tabela 5.9 verifica-se que o índice $II_{max} > 0$ e encontram-se em uma faixa de 31,76% a 284,75% de ganho. Os índices FIC foram colocados em gráficos (figura 5.27, 5.28 e 5.29), para uma melhor visualização dos resultados obtidos.

Cada uma das figuras possuem 2 curvas e conjuntos de pontos. Os conjuntos de pontos são os pontos obtidos no experimento (tradicional), uma das curvas (vide a legenda de cada um dos gráficos) é uma curva média obtida usando o método dos mínimos quadrados e função do tipo potencial e a outra curva ajustada aos valores obtidos pela abordagem utilizando as F.E.

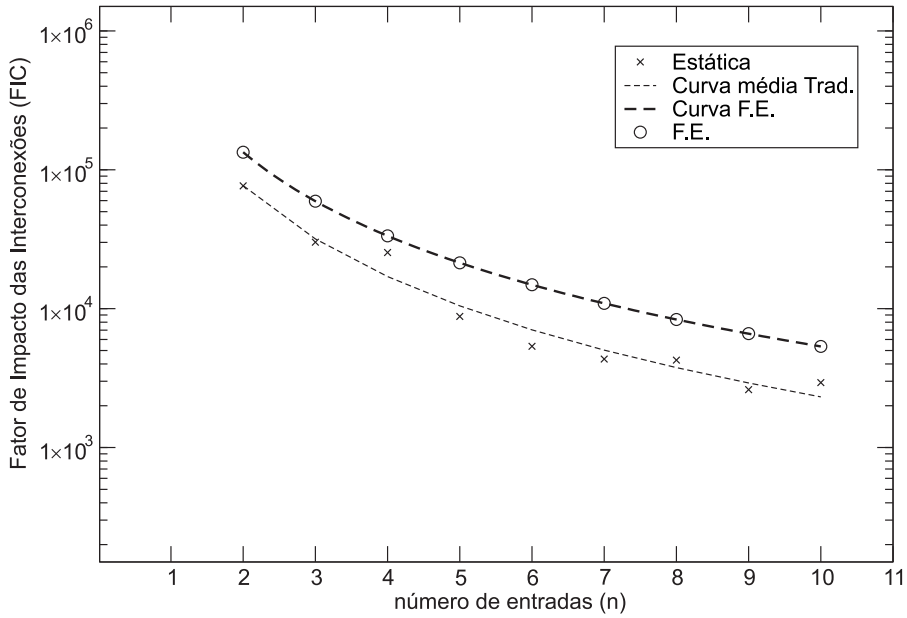

Figura 5.27: Camada de uma RNA com 1 neurônio e n entradas

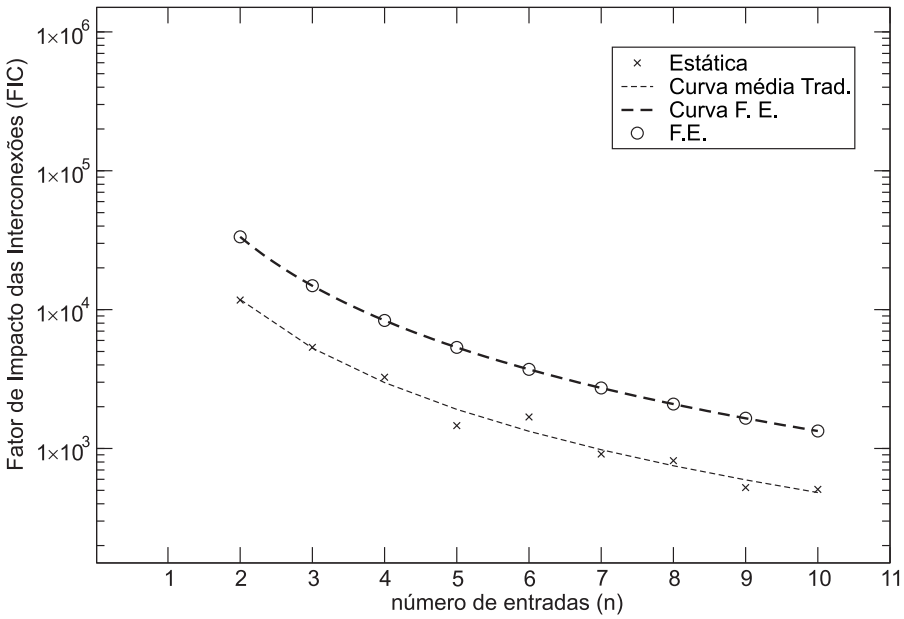

Figura 5.28: Camada de uma RNA com 2 neurônio e n entradas

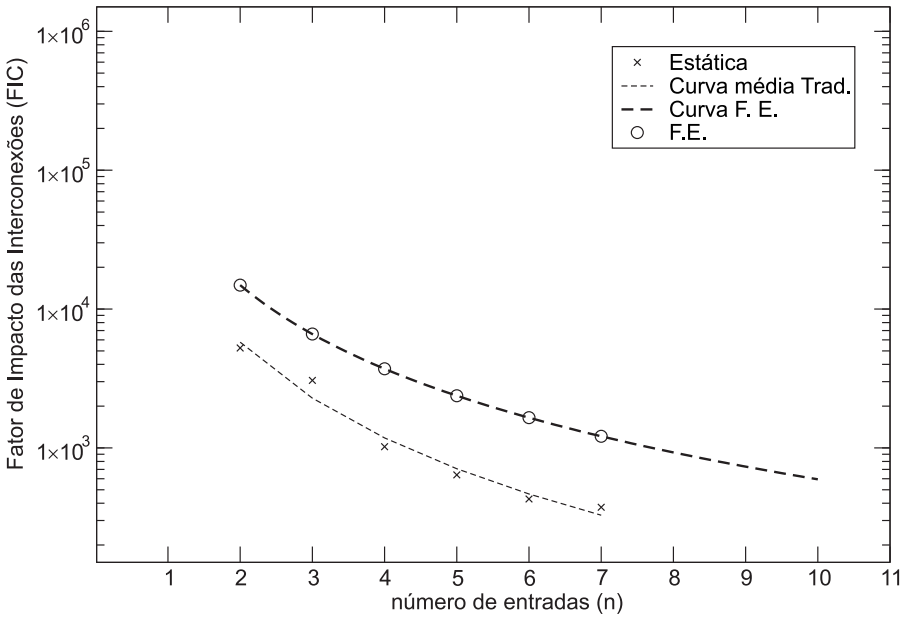

Figura 5.29: Camada de uma RNA com 3 neurônio e n entradas

Pela análise dos gráficos pode-se comprovar que as F.E têm uma capacidade de interconexão *superior* a metodologia tradicional.

Com isso demonstra-se que as F.E. reduzem o impacto das interconexões, pela eliminação do transporte de dados via barramento, entre os elementos de processamento de uma RNA e com isso reduzir o problema das interconexões (enunciado na seção 1.1) contribuindo assim para a implementação de RNAs em hardware.

oOo

Capítulo 6

Conclusão e perspectivas

Conclusão

Esta tese mostrou que é possível reduzir a complexidade das interconexões de uma rede neuronal artificial usando reconfiguração dinâmica. A técnica descoberta recebeu o nome de Figuras de Execução (F.E.).

As F.E. levam as operações aos dados e os mantêm fixos, eliminando assim a necessidade de transportá-los através de fios ou barramentos e com isso consegue-se diminuir a complexidade das interconexões[74].

Em simulações mediu-se o impacto das interconexões utilizando a metodologia tradicional e as F.E., para uma dada rede neuronal[1]. A seguir definiu-se uma medida relativa do impacto das interconexões e obtiveram-se *ganhos* de 31% a 284% das F.E sobre a metodologia tradicional.

Um dos grandes problemas foi o *baixo desempenho*, encontrado na simulação de

[1]Variou-se o número de suas entradas e também o número dos neurônios da camada intermediária.

uma rede neuronal utilizando as F.E. Neste experimento comparativo obteve-se 78,86 MCPS (projeto estático) contra 47,25 kCPS (projeto reconfigurável dinamicamente). Este baixo desempenho deve-se ao tempo de reconfiguração do FPGA dinamicamente reconfigurável, se o mesmo for reduzido a 0 ns obtém-se um desempenho de 86,55 MCPS.

Limitações gerais

Como fatores limitantes ao uso das F.E. em problemas reais, pode-se destacar:

1. A interface de programação dos DR-FPGAs, que geralmente trabalham a baixa velocidade e de forma seqüencial;
2. A baixa capacidade de células lógicas dos DR-FPGAs do tipo *sea-of-gates* disponíveis no mercado;
3. Metodologias e ferramentas de simulação para projetos reconfigurados dinamicamente;
4. Ferramentas de posicionamento, roteamento e geração de *bitstream* de projetos sem pinos de E/S ou *netlists* sem conexões externas .

Perspectivas

Na linha de desenvolvimento de RNAs e Neurocomputadores, sugere-se:

1. Extensão da ferramenta Figaro da Atmel para a geração de bitstream de macros e ou projetos sem pinos de E/S;

2. Projeto e implementação de uma placa de hardware, padrão PCI, que suporte a execução das F.E;

3. Estudo, projeto e implementação de um ambiente integrado para projeto e execução de redes neuronais;

4. Estudo de um ASIC reconfigurável[75], FPGA ou *core* FPGA que possa executar as F.E. e que possua as seguintes características:

   - **Interface de programação**: deve permitir a programação em paralelo de vários DR-FPGAS eficientemente, visando reduzir o impacto do tempo de reconfiguração dinâmica e ainda permitir a leitura/escrita de grupos de flip-flops (barra de dados) de maneira eficiente;
   - **DR-FPGAs Virtual**: interconexão de vários DR-FPGAs físicos[2] e que são vistos como um único DR-FPGA lógico (*FPGA como memória*);
   - **Dados virtuais**: permite que um dado seja replicado em varias partes de um mesmo DR-FPGA (endereços imagens) e/ou em outros DR-FPGAs[3]. Sugere-se utilizar uma abordagem derivada do conceito de *Reflective Memory*[77] no espaço da memória de programação, para o ajuste do dados dos flip-flops;

[2] A nível de vias expressas e locais e não de pinos de E/S[76]

[3] Na figura 5.14 os dados x1 (barra de dados) aparecem em várias F.E., no caso em que os dados são gerados dinamicamente, faz-se necessário um mecanismo semelhante a endereços imagens, onde um endereço seleciona várias locações de E/S ou memórias ao mesmo tempo

5. Estudar novas formas de comunicação massivamente paralela entre os periféricos e as F.E.;

6. Estudo, projeto e implementação de um neurocomputador utilizando o CI desenvolvido para a execução das F.E.

Como as Figuras de Execução não estão limitadas apenas ao domínio das RNAs e como acredita-se que elas também podem contribuir em outros domínios, nos quais sejam necessário processamento paralelo e/ou o impacto das conexões entre os elementos de processamento seja grande, sugere-se, de maneira geral:

1. Desenvolvimento de um modelo teórico para as F.E.;

2. Definição de uma linguagem de programação para as F.E.;

3. Construção de um compilador (ou interpretador) e simulador para as F.E.;

4. Projeto e implementação de uma plataforma de hardware que suporte a execução das F.E.;

5. Estudo, projeto e implementação de um ambiente integrado edição/compilação/depuração e execução das F.E.

As duas linhas de desenvolvimento propostas são complementares e algumas das tarefas são extensões ou se encontram duplicadas.

oOo

Referências Bibliográficas

1 LYSAGHT, P.; DUNLOP, J. Dynamic reconfiguration of field programmable gate arrays. In: MOORE, W.; LUK, W. (Ed.). *More FPGAs: Proceedings of the 1993 International workshop on field-programmable logic and applications*. Oxford, England: Abingdon EE&CS Books, 1993. p. 82–94. Disponível em: <http://oak.eee.strath.ac.uk/publications.html>. Acesso em: 31 out. 2003.

2 BAILEY, J.; HAMMERSTROM, D. Why VLSI implementations of associative VLCNs require connection multiplexing. In: *IEEE International Conference On Neural Network 1998*. USA: [s.n.], 1998. v. 2, p. 173–180.

3 TORRESEN, J. *Parallelization of Backpropagation Training for Feed-Forward Neural Networks*. Tese (PhD Thesis) — Norwegian University of Science and Technology, 1996. Disponível em: <http://heim.ifi.uio.no/~jimtoer/papers.html>. Acesso em: 11 nov. 2003.

4 WIDROW, B.; RUMELHART, D. E.; LEHR, M. A. Neural networks: Applications in industry business and science. *Communications of the ACM*, v. 37, n. 3, p. 93–105, 1994.

5 FAUSET, F. *Fundamental of Neural Networks*. USA: Prentice Hall, 1994. ISBN 0-13-042250-9.

6 HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. 1. ed. USA: Prentice Hall, 1994. ISBN 0-13-895863-7.

7 HARVEY, R. L. *Neural Network Principles*. USA: Prentice-Hall, Inc., 1994. ISBN 0-13-112194-4.

8 LINDSEY, C. S.; LINDBLAD, T. Survey of neural network hardware. In: *SPIE 1995 Symposium on Aerospace/Defense Sensing and Control and Dual Use Photonics,Proceedings of applications and Science of Artificial Neural Networks Conference*. [s.n.], 1995. v. 2492, part two, p. 1194–1205. Disponível em: <http://msia02msi.se/~lindsey/activities.html>. Acesso em: set. 2000.

9 LINDBLAD, T. et al. Neural networks chips. In: *Proceedings of AI Applications in Solar-Terrestrial Physics, Esa Wpp Series*. [s.n.], 1997. WPP 148, p. 87–98. Disponível em: <http://msia02msi.se/~lindsey/activities.html>. Acesso em: set. 2000.

10 BEIU, V. Mathematics of neural networks. In: ELLACOT, S. W.; MASON, J. C.; ANDERSON, I. J. (Ed.). [S.l.]: Kluwer Academic Publishers, 1997. cap. 12 - Constant Fan-In Digital Neural Networks are VLSI-Optimal, p. 89–94. ISBN 0-792-39933-1.

11 NICOLETTI, G. M. An analysis of neural networks as simulators and emulators. *Cybernetics and Systems*, v. 31, n. 3, p. 253–282, Apr-May 2000.

12 SERBEDVZIJA, N. B. Simulating artificial neural networks on parallel architectures. *IEEE Computer*, p. 56–63, Mar 1996.

13 SCHÖNAUER, T. et al. Digital neurohardware: Principles and perspectives. In: *Proceedings of Neuronale Netze in der Anwendung NN'98*. Germany: [s.n.], 1998. p. 101–106. Disponível em: <citeseer.nj.nec.com/schonauer98digital.html>. Acesso em: 11 nov. 2003.

14 HECHT-NIELSEN, R. *Neurocomputing*. USA: Addison-Wesley Publishing Company, Inc., 1990. ISBN 0-21-09355-3.

15 AXELROD, T. S. Handbook of neural computation. In: FIESLER, E.; BEALE, R. (Ed.). IOP Publishing Ltda and Oxford University Press, 1997. cap. E1.1 - Introduction, p. E1.1. Disponível em: <http://www.iop.org/Books/CIL/HNC/index.html>. Acesso em: 05 jul. 2003.

16 MOERLAND, P. D.; FIESLER, E. Handbook of neural computation. In: FIESLER, E.; BEALE, R. (Ed.). IOP Publishing Ltda and Oxford University Press, 1997. cap. E1.2 - Neural Network adaptations to hardware implementation, p. E1.2:1–E1.4:13. Disponível em: <http://www.iop.org/Books/CIL/HNC/index.html>. Acesso em: 05 jul. 2003.

17 ELDREDGE, J. G. *FPGA Density Enhancement of a Neural Network Through Run-Time Reconfiguration*. Dissertação (Mestrado) — Brigham Young University, Utah, USA, May 1994. Disponível em: <http://splish.ee.byu.edu/docs/rrann-ths.ps>. Acesso em: 30 set. 2003.

18 HAMMERSTROM, D. A VLSI architecture for high-performance, low-cost, on-chip learning. In: *1990 IJCNN International Joint Conference on Neural Networks*. USA: [s.n.], 1990. v. 2, p. 537–544.

19 BRUNELLI, L. *Estudo, Projeto e Síntese em HDL de uma Rede Neuronal*. Campina Grande, PB, Brasil, Junho 2002. Disponível em: <http://www.dee.ufcg.edu.br/~lubru>. Acesso em: 05 jul. 2003.

20 Mc GINNITY, T. M. et al. Novel architecture and synapse design for hardware implementations of neural networks. *Compute & Electrical Engineering*, v. 24, p. 75–87, 1998.

21 ROCHE, B. et al. Signalling techniques and their effect on neural network implementation sizes. *Information Sciences*, v. 132, n. 1-4, p. 67–82, February 2001.

22 TOSCANI, L. V.; VELOSO, P. A. S. *Complexidade de Algoritmos*. 1. ed. Brasil: Editora Sagra Luzzatto, 2001. ISBN 85-241-0649-2.

23 HAMMERSTROM, D. The connectivity requirements of simple association, or how many connections do you need? In: ANDERSON, D. Z. (Ed.). *Neural Information Processing System*. New York, USA: American Institute of Physics, 1988. p. 338–347. Disponível em: <http://www.cse.ogi.edu/~strom/papers/index.html>. Acesso em: 30 set. 2003.

24 BEIU, V. How to build VLSI - efficient neural chips. In: ALPAYDIN, E. (Ed.). *Proceedings of the International ICSC Symp. on Engineering of Intelligent Systems EIS'98*. Tenerife, Spain: ICSC Academic Press, 1998. v. 2, p. 66–75.

25 CRAVEN, M. P.; CURTIS, K. M.; HAYES-GILL, B. R. Consideration of multiplexing in neural network hardware. *IEE Proceedings Circuit Devices System*, v. 141, n. 3, p. 237–240, Jun 1994.

26 BEIU, V. A survey of perceptron circuit complexity results. In: *Proceedings of the International Joint Conference on Neural Networks 2003*. USA: [s.n.], 2003. v. 2, p. 989–994.

27 VUILLEMIN, J. E. et al. Programmable active memories: reconfigurable systems come of age. *IEEE Transactions on Very Large Scale Integration VLSI Systems*, v. 4, n. 1, p. 56–69, Mar 1996.

28 MANGIONE-SMITH, W. H. et al. Seeking solutions in configurable computing. *IEEE Computer*, v. 30, n. 12, p. 38–43, Dec 1997.

29 DEHON, A.; WAVRZYNEK, J. Reconfigurable computing: What, why and implications for design automation. In: *Proceedings of the 36th ACM/IEEE conference on Design automation conference*. USA: ACM PRESS, 1999. p. 610 – 615.

30 COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, v. 34, n. 2, p. 171–210, Jun 2002.

31 OLDFIELD, J. V.; DORF, R. C. *Field-Programmable Gate Arrays*. USA: John Wiley & Sons, Inc., 1995. ISBN 0-47155-665-1.

32 ATMEL. *Application Note: Implementing Cache Logic with FPGAs*. USA, Sep 1999. Disponível em: <http://www.atmel.com/dyn/resources/prod_documents/DOC0461.PDF>. Acesso em: 11 nov. 2003.

33 LYSAGHT, P. et al. Artificial neural network implementation on a fine-grained FPGA. In: HARTENSTEIN, R. W.; SERVI, M. Z. (Ed.). *Field-Programmable*

Logic. Prague, Czech Republic: Springer Verlag, 1994. p. 421–431. Disponível em: <http://oak.eee.strath.ac.uk/publications.html>. Acesso em: 30 set. 2003.

34 HADLEY, J. D. *The Performance Enhancement of a Run-Time Reconfigurable FPGA System Through Partial Reconfiguration*. Dissertação (Mestrado) — Brigham Young University, Utah, USA, November 1995. Disponível em: <http://splish.ee.byu.edu/docs/rrann2-ths.ps>. Acesso em: 04 nov. 2003.

35 WIRTHLIN, M. J.; HUTCHINGS, B. L. Improving functional density using run-time circuit reconfiguration. *IEEE Transactions on Very Large Scale Integration System*, v. 6, n. 2, p. 247–256, Jun 1998.

36 VITTOZ, E. A. Handbook of neural computation. In: FIESLER, E.; BEALE, R. (Ed.). IOP Publishing Ltda and Oxford University Press, 1997. cap. E1.3 - Analog VLSI implementation of neural networks, p. E1.3:1–E1.3:17. Disponível em: <http://www.iop.org/Books/CIL/HNC/index.html>. Acesso em: 5 jul. 2003.

37 HEEMSKERK, J. N. H. *Overview of Neural Hardware*. 1995. Disponível em: <http://citeseer.nj.nec.com/heemskerk95overview.html>. Acesso em: 11 nov. 2003.

38 SCHÜFFNY, R.; GRAUPNER, A.; SCHREITER, J. Hardware for neural networks. In: *4th International Workshop Neural Networks in Applications NN'99*. Magdeburg, Germany: [s.n.], 1999. Disponível em: <http://www.iee.et.tu-dresden.de/~hpsnweb/hdown//nn99.ps>. Acesso em: 11 out. 2003.

39 SAXENA, I.; HORAN, P. G. Handbook of neural computation. In: FIESLER, E.; BEALE, R. (Ed.). IOP Publishing Ltda and Oxford University Press, 1997. cap. E1.5 - Optical implementations, p. E1.5:1–E1.5:20. Disponível em: <http://www.iop.org/Books/CIL/HNC/index.html>. Acesso em: 05 jul. 2003.

40 SHAMS, S.; GAUDIOT, J. Parallel architectures for artificial neural networks. In: SUNDARARAJAN, N.; SARATCHANDRAN, P. (Ed.). [S.l.]: IEEE Computer Society, 1998. cap. 9 - Implementing Regularly Structured Neural Networks on the DREAM Machine, p. 271–302. ISBN 0-8186-8399-6.

41 NORDSTRÖM, T.; SVENSSON, B. Using and designing massively parallel computer for artificial neural networks. *Journal of Parallel and Distributed Computing*, v. 14, n. 3, p. 260–285, 1992. Disponível em: <http://www.sm.luth.se/~tono/papers/B.jpdc.92.ps.gz>. Acesso em: 11 out. 2003.

42 TORRESEN, J.; LANDSVERK, O. Parallel architectures for artificial neural networks. In: SUNDARARAJAN, N.; SARATCHANDRAN, P. (Ed.). [S.l.]: IEEE Computer Society, 1998. cap. 2 - A Review of Parallel Implementations of Backpropagation Neural Networks, p. 25–63. ISBN 0-8186-8399-6.

43 PORRMANN, M. et al. A dynamically reconfigurable hardware accelerator for self-organizing feature maps. In: *Proc. of the 5th World Multi-Conference on Systems, Cybernetics and Informatics, SCI 2001*. Orlando, Florida, USA: [s.n.], 2001. p. 242–247. Disponível em: <http://wwwhni.uni-paderborn.de/~mario/publications/index_e.php>. Acesso em: 16 out. 2003.

44 KOLINUMMI, P. et al. PARNEU: general-purpose partial tree computer. *Microprocessor and Microsystem*, v. 24, n. 1, p. 23–42, Mar 2000.

45 PORRMANN, M.; WITKOWSKI, U.; RüCKERT, U. A massively parallel architecture for self-organizing feature maps. *IEEE Transactions on Neural Networks*, v. 14, n. 5, p. 1110–1121, Sep 2003.

46 BARATTA, D. et al. Microelectronic implementation of artificial neural network. In: *Proceedings of 5th Electronic Devices and Systems International Conference*. Brno, CZ: [s.n.], 1998. p. VI–IX. Disponível em: <http://www.micro.dibe.unige.it/diotalevi/publications.htm>. Acesso em: 17 out. 2003.

47 BEIU, V. Handbook of neural computation. In: FIESLER, E.; BEALE, R. (Ed.). IOP Publishing Ltda and Oxford University Press, 1997. cap. E1.4 - Digital integrated circuit implementations, p. E1.4:1–E1.4:34. Disponível em: <http://www.iop.org/Books/CIL/HNC/index.html>. Acesso em: 05 jul. 2003.

48 CANER, E. S. *Um ambiente de desenvolvimento de aplicações de redes neuronais com facilidade de multiprocessamento*. Tese (PhD) — COPPE/UFRJ, RJ, Brasil, Fev 2001.

49 DIOTALEVI, F. *Analog Microelectronic Supervised Learning Systems*. Tese (PhD Thesis) — University of Genova, Italy, Oct 2000. Disponível em: <http://www.micro.dibe.unige.it/diotalevi/downloads/Diotalevi_PhD_thesis.pdf>. Acesso em: 11 nov. 2003.

50 LINDSEY, C. S. *Neural Networks in Hardware - Architectures, Products and Applications*. Stockholm, Sweden: [s.n.], Mar 1988. On line lecture. Atualizado em 26 de Agosto de 2002. Disponível em: <http://www.particle.kth.se/~lindsey/HardwareNNWCourse/HardwareNNWCourse.zip>. Acesso em: 11 nov. 2003.

51 TANENBAUM, A. S. *Structured Computer Organization*. 4. ed. USA: Prentice Hall, 1999. ISBN 0-13-095990-1.

52 BROWN, S.; ROSE, J. FPGA and CPLD architectures: A tutorial. *IEEE Design & Test of Computers*, v. 13, n. 2, p. 42–57, 1996.

53 BARROS, E. et al. *Hardware/software co-design: projetando hardware e software concorrentemente*. São Paulo, Brasil: [s.n.], 2000. Escola de Computação 2000.

54 ZHANG, X.; NG, K. W. A review of high-level synthesis for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems*, v. 24, n. 4, p. 199–211, August 2000.

55 HADLEY, J. D.; HUTCHINGS, B. L. Designing a partially reconfigured system. In: SCHEWEL, J. (Ed.). *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, Proc. SPIE 2607*. Bellingham, WA, USA: SPIE – The International Society for Optical Engineering, 1995. p. 210–220. Disponível em: <citeseer.nj.nec.com/hadley95designing.html>. Acesso em: 04 nov. 2003.

56 ELDREDGE, J. G.; HUTCHINGS, B. L. RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs. In: *1994 IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1994. v. 4, p. 2097–2102.

57 MORENO, J. M. et al. The role of dynamic reconfiguration for implementing artificial neural networks models in programmable hardware. In: MIRA, J.; SÁNCHEZ-ANDRÉS, J. V. (Ed.). *Engineering Applications of Bio-Inspired Artificial Neural Networks, International Work-Conference on Artificial and Natural Neural Networks, IWANN '99, Proceedings, Volume II*. Alicante, Spain: Springer-Verlag, 1999. (Lecture Notes in Computer Science, v. 1607), p. 85–94.

58 BACKUS, J. Can programming be liberated from the von Neumann style? a funcional style and its algebra of programs. *Communications of the ACM*, v. 21, n. 8, p. 613–641, August 1977. ACM Turing Award Lecture.

59 KWIAT, K. A.; JR., W. H. D. Modeling a versatile FPGA for prototyping adaptive systems. In: *Proceedings on Sixth IEEE International Workshop on Rapid System Prototyping*. Chapel Hill, NC, USA: [s.n.], 1995. p. 174–180.

60 VASILKO, M.; CABANIS, D. Improving simulation accuracy in design methodologies for dynamically reconfigruable logic system. In: *Proceedings of IEEE Symposium off FPGAs for Custom Computing Machines*. Napa, CA, USA: [s.n.], 1999. p. 20–23.

61 LUK, W.; SHIRAZI, N.; CHEUNG, P. Y. K. Modelling and optimising run-time reconfigurable system. In: ARNOLD, J.; POCEK, K. L. (Ed.). *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*. Napa, CA, USA: IEEE Computer Society Press, 1996. p. 197–176.

62 LYSAGHT, P.; STOCKWOOD, J. A simulation tool for dynamically reconfigurable field programmable gate arrays. *IEEE Transactions on Very Large Scale Integration System*, v. 4, n. 3, p. 381–390, September 1996.

63 ROBINSON, D.; LYSAGHT, P. Methods of exploiting simulation technology for simulating the timming of dynamically reconfigurable logic. *IEE Proceedings Computers an Digital Techniques*, v. 147, n. 3, p. 175–180, May 2000.

64 ROBERTSON, I.; IRVINE, J. A design flow for partially reconfigurable hardware. *ACM Transactions on Embedded Computing Systems*, v. 3, n. 2, p. 257–283, May 2004.

65 ATMEL. *Datasheet: AT40K/AT40KLV Series FPGA*. USA, Apr. 2002. 67 p. Rev. 0896C. Disponível em: <www.atmel.com/dyn/resources/prod_documents/DOC0896.PDF>. Acesso em: 22 set. 2004.

66 ATMEL. *Aplication note: AT40K Series Cache Logic (Mode 4) Configuration*. USA, Dec. 1998. 24 p. Rev. 1088AX.

67 ATMEL. *Aplication note: AT40K Series Configuration*. USA, Mar. 2002. 41 p. Rev. 1009B. Disponível em: <www.atmel.com/dyn/resources/prod_documents/DOC1009.PDF>. Acesso em: 22 set. 2004.

68 WIRTHLIN, M. J. *Improving Functional Density Through Run-Time Circuit Reconfiguration*. Tese (Doutorado) — Brigham Young University, Utah, USA, November 1997. Disponível em: <http://splish.ee.byu.edu/docs/mjw_ths.ps>. Acesso em: 24 set. 2004.

69 BAILEY, D. H. Onwoard to Petaflops Computing. *Communications of the ACM*, v. 40, n. 6, p. 90–92, June 1997.

70 DONGARA, J. J.; WALKER, D. W. The Quest for Petascale Computing. *IEEE Computing in Science & Engineering*, v. 3, n. 3, p. 32–39, May/June 2001.

71 WARREN, P. The future of computing - new architectures and new technologies. Part 1 Biology versus silicon. *IEE Computing & Control Engineering Journal*, v. 13, n. 2, p. 61–65, April 2002.

72 COMPTON, K. *Programming Architecture for Run-Time Reconfigurable Systems*. Dissertação (Mestrado) — University of Washington, Seatte, WA USA, December 1999. Disponível em: <http://www.ece.wisc.edu/~kati/Publications/Compton_MSThesis.pdf>. Acesso em: 14 dez. 2004.

73 LI, Z.; COMPTON, K.; HAUCK, S. Configuration caching management techniques for reconfigurable computing. In: *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa Valley, CA USA: [s.n.], 2000. p. 22–36.

74 BRUNELLI, L. et al. A novel Approach to reduce Interconnect Complexity in ANN Hardware Implementation. In: *IEEE - International Joint Conference on Neural Networks*. Canada: [s.n.], 2005. Aceito para publicação.

75 COMPTON, K. L. *Architecture Generation of Customized Reconfigurable Hardware*. Tese (PhD Thesis) — Northwestern University, USA, Dec 2003. Disponível em: <http://www.ece.wisc.edu/~kati/Publications/Compton_Dissertation.pdf>. Acesso em: 7 may 2005.

76 BABB, J.; TESSIER, R.; AGARWAL, A. Virtual wires: overcoming pin limitations in FPGA-based logic emulators. In: *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*. Napa, CA, USA: [s.n.], 1993. p. 142 – 151.

77 JOVANOVIC, M.; MILUTINOVIC, V. An overview of reflective memory systems. *IEEE Concurrency*, v. 7, n. 2, p. 56–64, Apr-Jun 1999.

Apêndice A

Simulação da RTR

A.1 Chave de isolação

A.1.1 chave_isolacao.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : chave_isolacao.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      14-09-2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES : DCS, DRL
// ------------------------------------------------------------------------------
// PROPOSITO       : Esta chave de isolacao permite a simulacao de circuitos de
//                   reconfiguracao dinamica.
//
//                   +-----------+----------+--------------------+
//                   I reconfig_b I   saida   I ESTADO do Conjunto I
//                   +-----------+----------+--------------------+
//                   I     0     I  entrada  I  Ativo             I
//                   +-----------+----------+--------------------+
//                   I     1     I     x     I  Transicao         I
//                   +-----------+----------+--------------------+
//                   I     z     I     z     I  Inativo           I
//                   +-----------+----------+--------------------+
//                   I     x     I     z     I  Inativo (adotado) I
//                   +-----------+----------+--------------------+
//                   obs: conjunto = blocos em operacao
//
```

```
// ------------------------------------------------------------------------------
// BILIOGRAFIA     : Lysaght P., Robinson D. "Methods of exploiting simulating
//                   the timming of dynamically reconfigurable logic", IEE Proc.
//                   Comput. Digit. Tech, Vol. 147, No. 3, May 2000.
// ------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO   FAIXA     : DESCRICAO            : DEFAULT   : UNIDADES
//   T_LOAD                []      : tempo de carga       :    0      :   ns
//   T_REMOVE              []      : tempo de remocao     :    0      :   ns
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N
//   Dominio do Clock         : N
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : N/A
//   Sintetizavel (s/n)       : N
//   Outras                   : N
// -FHDR------------------------------------------------------------------------

‘timescale 1 ns / 100 ps // adotado como referencia inicial.

module chave_isolacao   (
                        entrada,
                        saida,
                        reconfig_b
                        );

parameter T_LOAD    =  0;     // 0 ns
parameter T_REMOVE  =  0;     // 0 ns (igual a t_load no pior caso).

input    entrada;             // Entrada do sinal.

output   saida;               // Valores validos:
                              // X (durante a o periodo de transicao)
                              // Z (durante a o periodo de inatividade).

input    reconfig_b;          // Valores validos:
                              // 0 = chave ativa,
                              // 1 = periodo de transicao.
                              // Z = periodo de inatividade.
reg saida;

// Modulo combinatorio
always @(entrada or reconfig_b) begin
   case(reconfig_b)
      1’b0: #T_LOAD   saida = entrada;
      1’b1: #T_REMOVE saida = 1’bx;
      1’bx: saida = 1’bz; //LB
      1’bz: saida = 1’bz;
   endcase
end

endmodule

// ------------------------------------------------------------------------------
```

A.1.2 tb_chave_isolacao.v

```
// +FHDR-----------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ----------------------------------------------------------------------------
// NOME DO ARQUIVO : tb_chave_isolacao.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ----------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      14-09-2004  Luiz Brunelli  Gerada versao inicial.
// ----------------------------------------------------------------------------
// PALAVRAS CHAVES : DCS, DRL
// ----------------------------------------------------------------------------
// PROPOSITO       : Ambiente de Teste da chave de isolacao para simulacao
//                   de circuitos de reconfiguracao dinamica.
// ----------------------------------------------------------------------------
// BILIOGRAFIA     : Lysaght P., Robinson D. "Methods of exploiting simulating
//                   the timming of dynamically reconfigurable logic", IEE Proc.
//                   Comput. Digit. Tech, Vol. 147, No. 3, May 2000.
// ----------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO   FAIXA     : DESCRICAO            : DEFAULT   : UNIDADES
//   Nenhum.
// ----------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N
//   Dominio do Clock         : N
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : N/A
//   Sintetizavel (s/n)       : S
//   Outras                   : N
// -FHDR-----------------------------------------------------------------------

‘timescale 1 ns / 100 ps // adotado como referencia inicial.

module test0 ();

// Variaveis locais
reg   a;             // Estimulo de entrada
wire  b;             // Estimulo de saida
reg  controle_b;    // Controle da reconfiguracao.

// Modulo a ser testado
chave_isolacao  U1 (
                   .entrada(a),
                   .saida(b),
                   .reconfig_b(controle_b)
                   );

// Aplicacao dos Estimulos
initial
begin
   //Reconfig_b = x e entrada = z,x,0,1,0,1,0,x,z
   #50 controle_b = 1'bx;
   #50 a = 1'bz;
   #50 a = 1'bx;
   #50 a = 1'b0;
   #50 a = 1'b1;
```

```
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;

//Reconfig_b = z e entrada = z,x,0,1,0,1,0,x,z
#50 controle_b = 1'bz;
#50 a = 1'bz;
#50 a = 1'bx;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;

//Reconfig_b = 1 e entrada = z,x,0,1,0,1,0,x,z
#50 controle_b = 1'b1;
#50 a = 1'bz;
#50 a = 1'bx;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;

//Reconfig_b = 0 e entrada = z,x,0,1,0,1,0,x,z
#50 controle_b = 1'b0;
#50 a = 1'bz;
#50 a = 1'bx;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;

//Reconfig_b = 1 e entrada = z,x,0,1,0,1,0,x,z
#50 controle_b = 1'b1;
#50 a = 1'bz;
#50 a = 1'bx;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;

//Reconfig_b = z e entrada = z,x,0,1,0,1,0,x,z
#50 controle_b = 1'bz;
#50 a = 1'bz;
#50 a = 1'bx;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'b1;
#50 a = 1'b0;
#50 a = 1'bx;
#50 a = 1'bz;
```

```
   //Reconfig_b = x e entrada = z,x,0,1,0,1,0,x,z
   #50 controle_b = 1'bx;
   #50 a = 1'bz;
   #50 a = 1'bx;
   #50 a = 1'b0;
   #50 a = 1'b1;
   #50 a = 1'b0;
   #50 a = 1'b1;
   #50 a = 1'b0;
   #50 a = 1'bx;
   #50 a = 1'bz;

   #200;
end

endmodule
```

A.2 Exemplo E_OU

A.2.1 reconfiguracao.v

```
// +FHDR-----------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ----------------------------------------------------------------------------
// NOME DO ARQUIVO : reconfiguracao.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ----------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      10-09-2004  Luiz Brunelli  Gerada versao inicial.
// 0.2      12-09-2004  Luiz Brunelli  Incluida reconfiguracao durante reset_b.
// ----------------------------------------------------------------------------
// PALAVRAS CHAVES : DCS, DRL, RTR
// ----------------------------------------------------------------------------
// PROPOSITO       : Modulo principal responsavel pelo controle da
//                   reconfiguracao dinamica.
// ----------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//   --- Relogios ---
//   T_CLK_RECONF_MARCA   [N/D]  : Clock de Reconfig. :   30      :     ns
//   T_CLK_OPERACAO_MARCA [N/D]  : Clock de Operacao  :   100     :     ns
//
// ----------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clk, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : chave_isolacao, porta_and,
//                              porta_or, unid_ctrl.
//   Sintetizavel (S/N)       : N
//   Outras                   : N
```

```
// ------------------------------------------------------------------------------
// OBSERVACOES
//   Sugere-se automatizar a construcao deste modulo.
// ------------------------------------------------------------------------------
// LEGENDA
//   DCS = "Dynamic circuit switching".
//   DRL = "Dynamic reconfiguration logic".
//   RTR = "Run-time reconfiguration".
//   N/D = nao disponivel.
//   N/A = nao aplicavel.
// -FHDR----------------------------------------------------------------------

`timescale 1 ns / 100 ps // adotado como referencia inicial.

module reconfiguracao();

// ---------- Declaracao dos parametros -------------------------------------
parameter T_CLK_RECONF_MARCA   = 15;   // --- Clock de Reconfiguracao ---
                                       // T_CLK = 30ns (33.33 MHz)
                                       // T Marca = T Espaco

parameter T_CLK_OPERACAO_MARCA = 50;   // --- Clock de Operacao ---
                                       // T_CLK = 100ns ( 10.00 MHz)
                                       // T Marca = T Espaco
// --- Em outros modulos
defparam  UCR.T_LOAD_RESET_00 =     37; // --- Modulo 00 ---
defparam  UCR.T_LOAD_00       =      9; // Multiplos T_CLK_RECONF (min = 1)
defparam  UCR.T_REMOCAO_00    =      1;

defparam  UCR.T_LOAD_01       =      9; // --- Modulo 01 ---
defparam  UCR.T_REMOCAO_01    =      1; // Multiplos T_CLK_RECONF (min = 1)
// ------------------------------------------------------------------------------
// Definicao das variaveis locais.
// --- Entradas e saida dos modulos reconfiguraveis (mutuamente exclusivos)
reg  entrada_a;
reg entrada_b; //
wire saida_c;

// --- Unidade de controle da reconfiguracao.
reg            sinal_inicio;           // Chaveia a cena sequencialmente.
wire           sinal_fim;              // Termino da reconfiguracao da cena.
wire           sinal_reconfigura;      // Reconfiguracao em processo.
wire           sinal_ctrl_cena_00_b;   // NL0 = fecha chave de isolacao.
wire [8*11:1]  sinal_estado_cena_00;   // Transicao, Ativa, Inativa.
wire           sinal_ctrl_cena_01_b;   // NL0 = fecha chave de isolacao.
wire [8*11:1]  sinal_estado_cena_01;   // Transicao, Ativa, Inativa.
reg            clock_reconf;           // Sensivel a borda de subida.
reg            clock_operacao;         // Sensivel a borda de subida.
reg            sinal_reset_b;          // Ativo em NL0 e assincrono.

// ---- Fios entre modulos e chave de isolacao.
wire
   fio_entrada_a, fio_entrada_b, fio_saida_c,   // ent. e saida dos modulos.
   fio_00_a, fio_00_b, fio_00_c,                // chave_isolacao e modulos
   fio_01_a, fio_01_b, fio_01_c;

// Instanciacoes dos modulos
// --- Modulos do primeira cena.
porta_and   M00   (             // modulo 00
                  .a(fio_00_a),
                  .b(fio_00_b),
                  .c(fio_00_c)
```

```
                );

// --- Modulos do segunda cena.
porta_or    M01   (              // modulo 01
                  .a(fio_01_a),
                  .b(fio_01_b),
                  .c(fio_01_c)
                  );
// --- Unidade de Controle de Reconfiguracao.
unid_ctrl UCR    (
                 .inicio(sinal_inicio),
                 .fim(sinal_fim),
                 .reconfigura(sinal_reconfigura),
                 .ctrl_cena_00_b(sinal_ctrl_cena_00_b),
                 .estado_cena_00(sinal_estado_cena_00),
                 .ctrl_cena_01_b(sinal_ctrl_cena_01_b),
                 .estado_cena_01(sinal_estado_cena_01),
                 .clock(clock_reconf),
                 .reset_b(sinal_reset_b)
                );

// --- Chaves de isolacao para a simulacao da reconfiguracao dinamica
// --- Modulo 00 ---
chave_isolacao M00_a (                                  // entrada a.
                     .entrada(fio_entrada_a),
                     .saida(fio_00_a),
                     .reconfig_b(sinal_ctrl_cena_00_b)
                     );
chave_isolacao M00_b (                                  // entrada b.
                     .entrada(fio_entrada_b),
                     .saida(fio_00_b),
                     .reconfig_b(sinal_ctrl_cena_00_b)
                     );
chave_isolacao M00_c (                                  // saida c.
                     .entrada(fio_00_c),
                     .saida(fio_saida_c),
                     .reconfig_b(sinal_ctrl_cena_00_b)
                     );

// --- Modulo 01 ---
chave_isolacao M01_a (                                  // entrada a.
                     .entrada(fio_entrada_a),
                     .saida(fio_01_a),
                     .reconfig_b(sinal_ctrl_cena_01_b)
                     );
chave_isolacao M01_b (                                  // entrada b.
                     .entrada(fio_entrada_b),
                     .saida(fio_01_b),
                     .reconfig_b(sinal_ctrl_cena_01_b)
                     );
chave_isolacao M01_c (                                  // saida c.
                     .entrada(fio_01_c),
                     .saida(fio_saida_c),
                     .reconfig_b(sinal_ctrl_cena_01_b)
                     );

// --- Interligacoes gerais.
assign
   fio_entrada_a = entrada_a,
   fio_entrada_b = entrada_b,
   saida_c       = fio_saida_c;

// Aplicacao dos estimulos.
```

```
// --- Geracao do relogio da arquitetura
initial begin
   clock_operacao = 1'b0;
   forever begin
      #(T_CLK_OPERACAO_MARCA) clock_operacao = ~clock_operacao;
   end
end

// --- Geracao do relogio do circuito de reconfiguracao
initial begin
   clock_reconf = 1'b0;
   forever begin
      #(T_CLK_RECONF_MARCA) clock_reconf = ~clock_reconf;
   end
end

// --- Geracao do Reset
initial begin
        sinal_reset_b = 1'b1;
   #230 sinal_reset_b = 1'b0;
   #50  sinal_reset_b = 1'b1;
end

// --- Vetores de Teste
initial begin
   // Entradas Estaveis.
   sinal_inicio = 1'b0;

   //Fim da reconfiguracao apos reset_b
   wait ( sinal_fim);
   #50

   // Exercicio do Modulos da primeira cena
       entrada_a = 1'b0; entrada_b = 1'b0;
   #50 entrada_a = 1'b0; entrada_b = 1'b1;
   #50 entrada_a = 1'b1; entrada_b = 1'b0;
   #50 entrada_a = 1'b1; entrada_b = 1'b1;
   #50 entrada_a = 1'b0; entrada_b = 1'b0;
   #300;

   // Aplica-se o sinal de chaveamento de configuracao.
   @ (negedge clock_reconf) sinal_inicio = 1'b1;
   @ (negedge clock_reconf) sinal_inicio = 1'b0;

   // Termino da reconfiguracao
   wait ( sinal_fim);
   #50

   // Exercicio do Modulos da segunda cena
       entrada_a = 1'b0; entrada_b = 1'b0;
   #50 entrada_a = 1'b0; entrada_b = 1'b1;
   #50 entrada_a = 1'b1; entrada_b = 1'b0;
   #50 entrada_a = 1'b1; entrada_b = 1'b1;
   #50 entrada_a = 1'b0; entrada_b = 1'b0;
   #300;

   // Aplica-se o sinal de chaveamento de configuracao.
   @ (negedge clock_reconf) sinal_inicio = 1'b1;
   @ (negedge clock_reconf) sinal_inicio = 1'b0;

   // Termino da reconfiguracao
   wait ( sinal_fim);
   #50
```

```
   // Exercicio do Modulos da primeira cena
       entrada_a = 1'b0; entrada_b = 1'b0;
   #50 entrada_a = 1'b0; entrada_b = 1'b1;
   #50 entrada_a = 1'b1; entrada_b = 1'b0;
   #50 entrada_a = 1'b1; entrada_b = 1'b1;
   #50 entrada_a = 1'b0; entrada_b = 1'b0;
   #300;

   // Aplica-se o sinal de chaveamento de configuracao.
   @ (negedge clock_reconf) sinal_inicio = 1'b1;
   @ (negedge clock_reconf) sinal_inicio = 1'b0;

   // Termino da reconfiguracao
   wait ( sinal_fim);
   #50

   // Exercicio do Modulos da segunda cena
       entrada_a = 1'b0; entrada_b = 1'b0;
   #50 entrada_a = 1'b0; entrada_b = 1'b1;
   #50 entrada_a = 1'b1; entrada_b = 1'b0;
   #50 entrada_a = 1'b1; entrada_b = 1'b1;
   #50 entrada_a = 1'b0; entrada_b = 1'b0;
   #300

   // Fim do programa
   $stop(2);   // Encerra a simulacao
end

endmodule //Fim do Programa

// ------------------------------------------------------------------------------
```

A.2.2 unid_ctrl.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : unid_ctrl.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      10-09-2004  Luiz Brunelli  Gerada versao inicial.
// 0.1      11-09-2004  Luiz Brunelli  Chaveamento sequencial das cenas.
// 0.2      12-09-2004  Luiz Brunelli  Incluida a reconfiguracao apos reset_b.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES : UC, RTR
// ------------------------------------------------------------------------------
// PROPOSITO       : Unidade de Controle responsavel pela mudanca de cenas.
// ------------------------------------------------------------------------------
// PARAMETROS
//
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//
//   --- Modulo 00 ---
//   T_LOAD_RESET_00      [1-?]  : T carga do Modulo  :     1     : CLK_RECONF
//                                 apos o reset_b
//
//   T_LOAD_00            [1-?]  : T carga do Modulo  :     1     : CLK_RECONF
//   T_REMOCAO_00         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   --- Modulo 01 ---
//   T_LOAD_01            [1-?]  : T carga do Modulo  :     1     : CLK_RECONF
//   T_REMOCAO_01         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   OBS: De posse do arquivo modulo.md4, verifica-se o numero de linhas dele e
//        atualiza-se o valor do T_XXXXXXX. O valor minimo e de um  CLK_RECONF,
//        por causa da maquina sequencial.
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clock, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : N/A
//   Sintetizavel (S/N)       : N
//   Outras                   : N
// ------------------------------------------------------------------------------
// LEGENDA
//   RTR = "Run-Time Reconfiguration"
//   N/D = nao disponivel
//   N/A = nao aplicavel
// -FHDR-------------------------------------------------------------------------

module unid_ctrl  (
                  inicio, fim,
                  reconfigura,
                  ctrl_cena_00_b, estado_cena_00,
                  ctrl_cena_01_b, estado_cena_01,
                  clock, reset_b
                  );
```

```
// ---------- Declaracao dos parametros ---------------------------------
parameter T_LOAD_RESET_00  =  1;  // --- Modulo 00 ---
parameter T_LOAD_00        =  1;  // Multiplos do T_CLK_RECONF (min = 1).
parameter T_REMOCAO_00     =  1;

parameter T_LOAD_01        =  1;  // --- Modulo 01 ---
parameter T_REMOCAO_01     =  1;  // Multipols do T_CLK_RECONF (min = 1).
// ----------------------------------------------------------------------

// Definicao dos sinais
input   inicio;         // Chaveia a cena sequencialmente (nao ciclicamente).
output  fim;            // Termino da reconfiguracao da cena.

output  reconfigura;    // Reconfiguracao em processo
output  ctrl_cena_00_b; // NL0 = fecha chave de isolacao.
output  estado_cena_00; // Estado da reconfiguracao: ON, OFF, XXX.
output  ctrl_cena_01_b; // NL0 = fecha chave de isolacao.
output  estado_cena_01; // Estado da reconfiguracao: ON, OFF, XXX.

input   reset_b;        // Ativo em NL0 e assincrono.
input   clock;          // Clock de RECONFIGURACAO, sensivel a borda de subida.

reg             fim;
reg             reconfigura;
reg             ctrl_cena_00_b;
reg [8*11:1]    estado_cena_00; // Maximo 13 caracteres ASCII (b1234567890).
reg             ctrl_cena_01_b;
reg [8*11:1]    estado_cena_01; // Maximo 13 caracteres ASCII (b1234567890).

// Descrição do Hardware
always @(posedge clock) begin
   if(!reset_b) begin
      // Estado 0_0
      fim             <= 1'b0;
      reconfigura     <= 1'b0;
      ctrl_cena_00_b  <= 1'bz;
      estado_cena_00  <= " OFF";
      ctrl_cena_01_b  <= 1'bz;
      estado_cena_01  <= " OFF";

      // Estado 0_1
      @(posedge clock);
      fim             <= 1'b0;
      reconfigura     <= 1'b1;
      ctrl_cena_00_b  <= 1'b1;
      estado_cena_00  <= " XXX";
      ctrl_cena_01_b  <= 1'bz;
      estado_cena_01  <= " OFF";

      // Estado 0_2
      repeat (T_LOAD_RESET_00)   // Carga do Modulo 00
         @(posedge clock);       // Bistream Full|Compress.
      fim             <= 1'b1;   // <--
      reconfigura     <= 1'b0;
      ctrl_cena_00_b  <= 1'b0;
      estado_cena_00  <= " ON";
      ctrl_cena_01_b  <= 1'bz;
      estado_cena_01  <= " OFF";

      // Estado 0_3
      @(posedge clock);
      fim             <= 1'b0;   // <--
```

```
      reconfigura     <= 1'b0;
      ctrl_cena_00_b  <= 1'b0;
      estado_cena_00  <= " ON";
      ctrl_cena_01_b  <= 1'bz;
      estado_cena_01  <= " OFF";
   end
   else begin
      if(inicio) begin
         // Estado 1
         fim             <= 1'b0;
         reconfigura     <= 1'b1;
         ctrl_cena_00_b  <= 1'b1;
         estado_cena_00  <= " XXX";
         ctrl_cena_01_b  <= 1'bz;
         estado_cena_01  <= " OFF";

         // Estado 2 --- Remocao do Modulo 00.
         repeat (T_REMOCAO_00)      // Aguarda a remocao da interseccao
            @(posedge clock);       // dos bitstreams consecutivos.
         fim             <= 1'b0;
         reconfigura     <= 1'b1;
         ctrl_cena_00_b  <= 1'bz;
         estado_cena_00  <= " OFF";
         ctrl_cena_01_b  <= 1'b1;
         estado_cena_01  <= " XXX";

         // Estado 3 --- Carga do Modulo 01
         repeat (T_LOAD_01)         // Bitstreams do tipo window.
            @(posedge clock);       // e compress
         fim             <= 1'b1;   // <--
         reconfigura     <= 1'b0;
         ctrl_cena_00_b  <= 1'bz;
         estado_cena_00  <= " OFF";
         ctrl_cena_01_b  <= 1'b0;
         estado_cena_01  <= " ON";

         // Estado 4
         @(posedge clock);
         fim             <= 1'b0;   // <--
         reconfigura     <= 1'b0;
         ctrl_cena_00_b  <= 1'bz;
         estado_cena_00  <= " OFF";
         ctrl_cena_01_b  <= 1'b0;
         estado_cena_01  <= " ON";

         wait(inicio);  // Aguarda o pedido de reconfiguracao.

         // Estado 5
         @(posedge clock);
         fim             <= 1'b0;
         reconfigura     <= 1'b1;
         ctrl_cena_00_b  <= 1'bz;
         estado_cena_00  <= " OFF";
         ctrl_cena_01_b  <= 1'b1;
         estado_cena_01  <= " XXX";

         // Estado 6 --- Remocao do Modulo 01.
         repeat (T_REMOCAO_01)      // Aguarda a remocao da interseccao
            @(posedge clock);       // dos bitstreams consecutivos.
         fim             <= 1'b0;
         reconfigura     <= 1'b1;
         ctrl_cena_00_b  <= 1'b1;
         estado_cena_00  <= " XXX";
         ctrl_cena_01_b  <= 1'bz;
```

```
        estado_cena_01  <= " OFF";

        // Estado 7 --- Carga do Modulo 00
        repeat (T_LOAD_00)         // Bitstreams do tipo windows
        @(posedge clock);          // e compress.
        fim             <= 1'b1;
        reconfigura     <= 1'b0;
        ctrl_cena_00_b  <= 1'b0;
        estado_cena_00  <= " ON";
        ctrl_cena_01_b  <= 1'bz;
        estado_cena_01  <= " OFF";

        // Estado 8
        @(posedge clock);
        fim             <= 1'b0;
        reconfigura     <= 1'b0;
        ctrl_cena_00_b  <= 1'b0;
        estado_cena_00  <= " ON";
        ctrl_cena_01_b  <= 1'bz;
        estado_cena_01  <= " OFF";
      end
   end
end

endmodule  //Fim do programa
// ------------------------------------------------------------------------------
```

oOo

Apêndice B

Roteiro de Projeto RTR

B.1 Roteiro de implementação

```
================================================================================
Objeto:

   Roteiro para criacao de projetos reconfigurados dinamicamente para o  fabri-
   cante Atmel.

   Autor:  Luiz Brunelli
   Data:   13/9/2004     Versao: 0.1

================================================================================
Ferramentas utilizadas:

- Atmel Figaro - IDS7.6 com patch 9d versao Windows.
- Mentor Graphics - Model Sim Atmel 5.6e_p2.
- Mentor Graphics - LeonardoSpectrum Version: 2002c.37_OEM_Atmel.

Obs:
   O procedimento abaixo e valido para outras versoes dos programas  da  Mentor
   Graphics.

================================================================================
Descrição do exemplo:

   Dado um DR-FPGA Atmel AT40K40-2BGC (escolheu-se o maior por comodidade)  de-
   seja-se executar uma porta "E" e depois reconfigura-la para uma porta   "OU"
   preservando-se o restante do circuito.

Arquivos iniciais:
 fpga.pin       -> Descricao dos pinos utilizados no FPGA e
                   componente da familia AT40K.
 porta_and.v    -> Descricao em HDL da porta "E".
 porta_or.v     -> Descricao em HDL da porta "OU".
 ato_1.v        -> Descricao em HDL do circuito inicial.
 ato_2.v        -> Descricao em HDL do circuito reconfigurado.

Estrutura de diretorios:
  Projetos      -> Diretorio dos projetos.
```

```
     prj_01     -> Projeto 01.
        cena_1  -> Leiaute da porta "E".
        cena_2  -> Leiaute da porta "OU".
        doc     -> Documentacao do projeto.
        global  -> biblioteca de projeto e arquivos de configuracao do FPGA.
        palco   -> Leiaute das varias fases de configuracao do FPGA.
================================================================================
Filosofia de projeto [1].

   Um projeto reconfigurado dinamicamente constitui-se de varias  cenas,  (cada
cena possui seu sub-diretorio). Uma cena modela uma funcao desejada a  ser exe-
cutada carregada no FPGA e possui um determinado tempo de vida. No  exemplo sao
as funcoes "E" (cena_1) e "OU" (cena_2). Estas funcoes sao MACROS  (leiaute sem
pinos) que serao armazenadas na biblioteca bib.lib no diretorio global. As  Ma-
cros serão referenciadas nos arquivo ato_1.v e ato_2.v e que depois do  leiaute
darao origem aos bitstreams correspondentes. Cabera entao ao gerenciador de re-
configuracao dinamica a carga dos mesmos no tempo adequado. A geracao do leiau-
te do ato_2 e feita utilizando o recurso de ECO do Figaro. Este recurso permite
fazer pequenas mudancas automaticamente sem construir a nova cena desde o  ini-
cio.

REF:
1  J. Kadlec (2003 ), Handel-C Targeting ATMEL AT40K with Dynamic Re-configura-
   tion of modules based on CACHE LOGIC
   (www.celoxica.com/techlib/files/CEL-W0307171HUP-17.pdf), acesso: 18/05/2004.

================================================================================
Procedimentos:

1-) Crie uma pasta proj_01 no diretorio c:\proj.lb (exemplo).

2-) Dentro do sub-diretorio proj_01 crie as pastas: cena_1, cena_2, doc, global
    e palco.

3-) Copie os arquivos:
    porta_and.v -> C:\proj.lb\proj_01\cena_1
    porta_or.v  -> C:\proj.lb\proj_01\cena_2
    fpga.pin    -> C:\proj.lb\proj_01\global
    ato_1.v     -> C:\proj.lb\proj_01\palco\ato_1 (crie o sub-diretorio ato_1)
    ato_2.v     -> C:\proj.lb\proj_01\palco\ato_2 (crie o sub-diretorio ato_2)

    obs: crie tantas pastas ato_n quantas cenas tiver.

4-) Sintese dos arquivos HDL.
    - Crie um diretorio Leonardo, para abrigar o processo de sintese (ex:
      C:\proj.lb\leonardo).

    ==> Sintese das MACROS
    - Copie o arquivo C:\proj.lb\proj_01\cena_1\porta_and.v para
      C:\proj.lb\leonardo.
    - Invoque o programa LeonardoSpectrum e selecione Advanced Flow Tabs.
      > Na orelha Technology selecione AT40K e o botao Apply.
      > Na orelha Input e ajuste o diretorio de trabalho para
        C:\proj.lb\leonardo, no menu open files selecione o arquivo
        porta_and.v (faça um open) e "click" no botão Read.
        Verifique se seu arquivo fonte nao apresentou erros.
      > Na orelha Optimize deselecione Add I/O Pads
       (garante que as macros nao terao pads !!!) e "click" Apply.
      > Va a orelha Output e "click" Run Flow.
      > Salve o script de comando com o nome cmd
       (File -> Save Command File) para posterior uso.
      > Se for o caso depure o seu programa e ao sair salve o projeto
       (File -> Exit).
```

```
   - Compacte todos os arquivos com o nome leonardo.zip, teste-o e o mova para
     a pasta C:\proj.1b\proj_01\cena_1\ . A seguir expanda  os arquivos em uma
     pasta leonardo. Copie da pasta  leonardo o arquivo porta_and.edf  para  o
     diretorio C:\proj.1b\proj_01\cena_1\, a  seguir mova  leonardo.zip para a
     pasta leonardo e apague os arquivos do diretorio C:\proj.1b\leonardo.

   - Repita os passos anteriores para as demais cenas levando em conta  os no-
     mes dos arquivos.

   ==> Sintese dos atos.
   - Para cada ato:
     Siga o mesmo procedimento anterior e deixe a opcao  Add I/O pads SELECIO-
     NADA.
   - Ao final do processo copie os arquivos ato_1.edf e ato_2.edf para o dire-
     torio C:\proj.1b\proj_01\palco pois serao necessario para gerar a  confi-
     guracao inicial do FPGA e a(s) sua(s) reconfiguracoes (utilizando o  pro-
     cesso ECO do Figaro).

5-) Place&Route&Timming&[Bitstream].

   - Invoque o Figaro (IDS).
     > No Menu File -> Design Setup -> New Design
     > Aponte o diretorio C:\proj.1b\proj_01\cena_1
     > Design Name: porta_and.edf
     > Files Of Type: EDIF Netlist
     > Configuration: AT40K
     > Tools Flow: Exemplar-Verilog
     > "Click" OK.

   - No menu Design Directory Setup selecione c:\proj.1b\proj_01\cena1 e
     "click" OK. Faca o mesmo para a cena2 e para ato_1 (foram criados  3 pro-
     jetos).

   - Deselecione a opcao: Options -> Options -> Design Configuration: Automatic
     Create a User Lib.

   ==> Macros (não permite a geração de Bitstream).
   - Aponte o projeto cena_1 e open o projeto como uma macro (selecione a opcao
     EDIF Netlist) e aguarde.
   - No final do processo voce tera varias janelas. Organize as janelas  e  ar-
     raste a instancia do MAP Browser para G:(1,16).
   - Rode Tools -> Simulation -> Post Layout Simulation (nao esqueca  de fechar
     o ModelSim). No diretorio figba tem-se todo o timming da macro.
   - Rode Timming -> Show analyzed paths para gerar o porta_and.pdl.
   - Salve a macro na biblioteca bib.lib. Library -> Library Setup->Add Before
     com path  c:\proj.1b\proj_01\global  e com  nome bib.lib (crie e de OKs).
     De um "Check-in" na janela Compile e um ok. Saia do figaro.

   -Repita a operacao acima (não sera  necessario criar  novamente  a  bib.lib
   pois ela ja existe) para o projeto cena_2.

   ==> Atos.

   ### Ato 1
   - Aponte o projeto ato_1.
   - Ajuste o caminho da bib.lib
   - Invoque o gerador de macro (Tools -> Entry -> Generators), gere latch_1 e
     latch_2, salve na bib.lib e saia (cancel) do gerador.
   - Open como Design o projeto (Edif Netlist).
   - Import o arquivo fpga.pin (File -> Import Constraints -> selecione  *.pin
     e mude para o diretorio globlal para apontar o arquivo fpga.pin.
```

```
    - Organize as janelas.
    - Salve o Design (ato_1.fgd).
    - Abra a janela de compilacao (Windows -> New Compile Windows)
    - Organize as janelas e localize os 3 pinos a serem utilizados no projeto.
    - Posicione as instancias e as rotei.
    - Rode Post Layout Simulation e Analyzed Paths.
    - Gere os Bitstreams e os copie para a pasta bitstreams_1.0 (crie  a  pasta
      bitstream ato 1 e tipo "full").
    - Gere os Bitstreams Compress e os copie para a pasta bitstreams_1.1
    - Salve o projeto e saia.

    ### Ato 2 (usando ECO e as opcoes default)
    - Abra o projeto ato_1 como design (use o ato_1.fgd).
    - File -> Open as a ECO.
    - New Design, selecione EDIF netilist e aponte para ato_2.edf
    - De OK para prosseguir com o modo ECO.
    - Quando solicitado para rodar no modo ECO diga Yes.
    - O novo design mostra as partes nao modificadas em  Mangenta  e o restante
      que foi substituido (analizando-se as duas netlists) aparece  no caso nao
      roteado.
    - Salve o projeto como ato_2.
    - Rotei o projeto,  dispare o Post Layout Simulation,  Analyzed Paths  e os
      Bitstreams "full" (bitstreams_2.0) e compress (bitstreams-2.1).
    - Salve o projeto.
    - Gere o Bitstream do tipo Windows tendo como base o bistream ato_1  compri-
      mido.

6-) RESULTADOS.

    - Timming da porta "E"
    C:\proj.1b\proj_01\cena_1\figba

    - Timming da porta "OU"
    C:\proj.1b\proj_01\cena_2\figba

    - Timming do ato_1
    C:\proj.1b\proj_01\palco\figba

    - Timming do ato_2
    C:\proj.1b\proj_01\palco\figba

    - Biststreams:

    ### FPGA apos reset --> carga da porta "E".
    Ato_1 (compress)
    C:\proj.1b\proj_01\palco\bitstreams_1.1

    ### Porta "E" --> "OU"
    Ato_2 (Windows)
    C:\proj.1b\proj_01\palco\bitstrams_1_to_2

FIM
================================================================================
```

B.2 Programas fontes

B.2.1 fpga.pin

Descrição dos pinos utilizados do FPGA.

```
AddPartNamed ( 'AT40K40-2BGC' 'A'  )

Pinout ( 'ent_a' toPin 'A.D25' )
Pinout ( 'ent_b' toPin 'A.F23' )
Pinout ( 'sai_c' toPin 'A.F24' )
```

B.2.2 porta_and.v

Modulo HDL da porta "E".

```
// +FHDR-----------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ----------------------------------------------------------------------------
// NOME DO ARQUIVO : porta_and.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ----------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      13/05/2004  Luiz Brunelli  Gerada versao inicial.
// ----------------------------------------------------------------------------
// PALAVRAS CHAVES : AND
// ----------------------------------------------------------------------------
// PROPOSITO       : Porta "E, tipo macro e sem "pads" de E/S.
// -FHDR-----------------------------------------------------------------------

module porta_and (
                  Entrada_a,
                  Entrada_b,
                  Saida_c,
                  );

// Definicao dos sinais
input   Entrada_a;   // Entrada
input   Entrada_b;   // Entrada
output  Saida_c;     // Saida

// Descrição do Hardware
assign
   Saida_c = Entrada_a && Entrada_b;

endmodule // Fim da porta_and
```

B.2.3 porta_or.v

Modulo HDL da porta "OU".

```
// +FHDR----------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ---------------------------------------------------------------------------------
// NOME DO ARQUIVO : porta_or.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ---------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      13/05/2004  Luiz Brunelli  Gerada versao inicial.
// ---------------------------------------------------------------------------------
// PALAVRAS CHAVES : OR
// ---------------------------------------------------------------------------------
// PROPOSITO       : Porta "OR", tipo macro e sem "pads" de E/S.
// -FHDR----------------------------------------------------------------------------

module porta_or(
                Entrada_a,
                Entrada_b,
                Saida_c,
                );

// Definicao dos sinais
input   Entrada_a;   // Entrada
input   Entrada_b;   // Entrada
output  Saida_c;     // Saida

// Descrição do Hardware
assign
   Saida_c = Entrada_a || Entrada_b;

endmodule // Fim da porta_or
```

B.2.4 ato_1.v

Modulo estático com a porta "E"instanciado.

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : ato_1.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      13/05/2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES : RTR
// ------------------------------------------------------------------------------
// PROPOSITO       : Parte estatica e instancia da macro "AND".
// ------------------------------------------------------------------------------
// LEGENDA
//   RTR = "Run-time reconfiguration".
// -FHDR-------------------------------------------------------------------------

module ato_1(
             ent_a,
             ent_b,
             sai_c,
             );

// Definicao dos sinais
input   ent_a;   // Entrada
input   ent_b;   // Entrada
output  sai_c;   // Saida

// Variaveis internas.
wire  a, b, c;

// Descrição do Hardware (Estrutural)
latch_2 data_bar_entrada (                      // Atmel Macro.
                         .DATA({ent_a, ent_b}),
                         .ENABLE(1'b1),
                         .Q({a, b})
                          );

latch_1   data_bar_saida (                      // Atmel Macro.
                         .DATA(c),
                         .ENABLE(1'b1),
                         .Q(sai_c)
                          );

// --- Bloco a ser reconfigurado dinamicamente.
porta_and bloco_atual  (
                        .Entrada_a(a),
                        .Entrada_b(b),
                        .Saida_c(c)
                       );

endmodule   // Fim do programa
```

```
// === Declaracao dos Modulos Chamados ========================================

module  latch_2 (             // Atmel Macro.
                DATA,
                ENABLE,
                Q,
                );

input  [1:0] DATA;
input        ENABLE;       // 0 = latch.
output [1:0] Q;

endmodule // Fim do latch_2

// --------------------------------------------

module  latch_1 (             // Atmel Macro.
                DATA,
                ENABLE,
                Q,
                );

input  [0:0] DATA;
input        ENABLE;       // 0 = latch.
output [0:0] Q;

endmodule // Fim do latch_1

// --------------------------------------------

module porta_and (            // Macro do Usuario
                  Entrada_a,
                  Entrada_b,
                  Saida_c
                  );

input    Entrada_a, Entrada_b;
output   Saida_c;

endmodule // Fim da porta_and

// ============================================================================
```

B.2.5 ato_2.v

Modulo estático com a porta ”OU”instanciado.

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : ato_2.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      13/05/2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES : RTR
// ------------------------------------------------------------------------------
// PROPOSITO       : Parte estatica e instancia da marco "OR".
// ------------------------------------------------------------------------------
// LEGENDA
//   RTR = "Run-time reconfiguration".
// -FHDR-------------------------------------------------------------------------

module ato_2(
             ent_a,
             ent_b,
             sai_c,
             );

// Definicao dos sinais
input   ent_a;  // Entrada
input   ent_b;  // Entrada
output  sai_c;  // Saida

//  Variaveis internas.
wire  a, b, c;

// Descrição do Hardware (Estrutural)
latch_2 data_bar_entrada (                          // Atmel Macro.
                          .DATA({ent_a, ent_b}),
                          .ENABLE(1'b1),
                          .Q({a, b})
                          );

latch_1   data_bar_saida (                          // Atmel Macro.
                          .DATA(c),
                          .ENABLE(1'b1),
                          .Q(sai_c)
                          );

// --- Bloco a ser reconfigurado dinamicamente.
porta_or bloco_atual (
                      .Entrada_a(a),
                      .Entrada_b(b),
                      .Saida_c(c)
                      );

endmodule   // Fim do programa
```

```
// === Declaracao dos Modulos Chamados ========================================

module  latch_2 (             // Atmel Macro.
                DATA,
                ENABLE,
                Q,
                );

input  [1:0] DATA;
input        ENABLE;    // 0 = latch.
output [1:0] Q;

endmodule  // Fim do latch_2

// --------------------------------------------

module  latch_1 (             // Atmel Macro.
                DATA,
                ENABLE,
                Q,
                );

input  [0:0] DATA;
input        ENABLE;    // 0 = latch.
output [0:0] Q;

endmodule // Fim do latch_1

// --------------------------------------------

module porta_or (             // Macro do Usuario.
                 Entrada_a,
                 Entrada_b,
                 Saida_c
                 );

input    Entrada_a, Entrada_b;
output   Saida_c;

endmodule // Fim da porta_and

// ============================================================================
```

B.3 Resultados

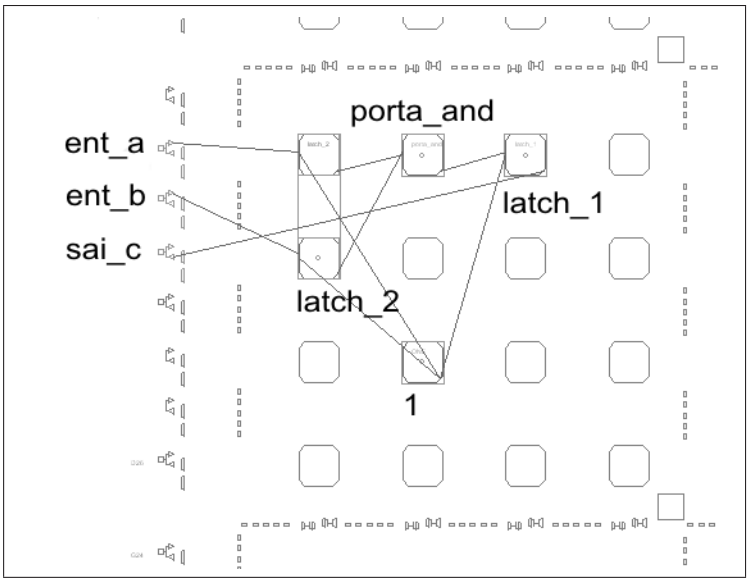

Figura B.1: Posicionamento do ATO_1

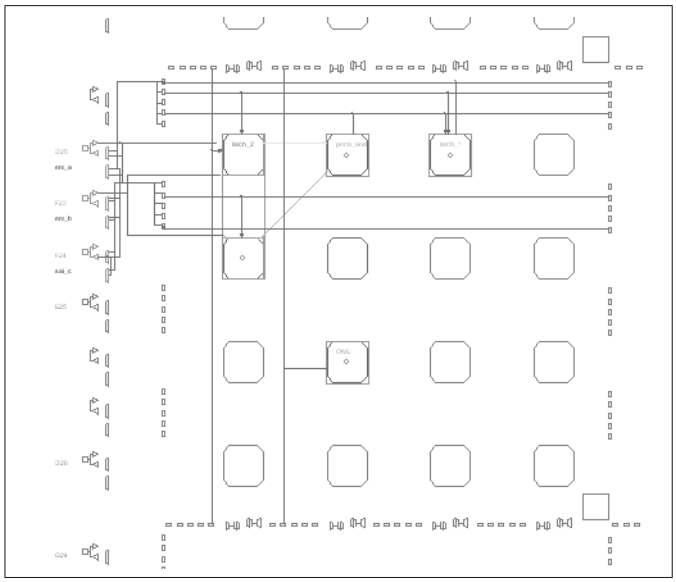

Figura B.2: Roteamento do ATO_1

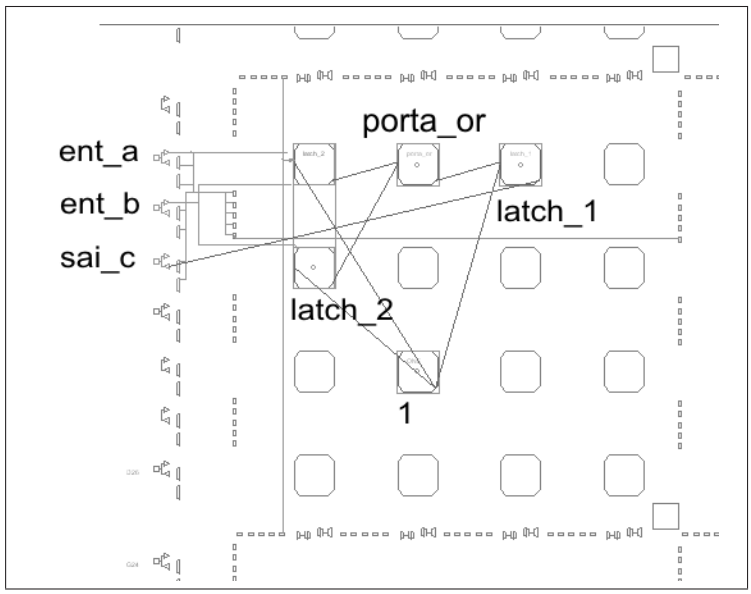

Figura B.3: ATO_2 ⟸ ECO sobre o ATO_1

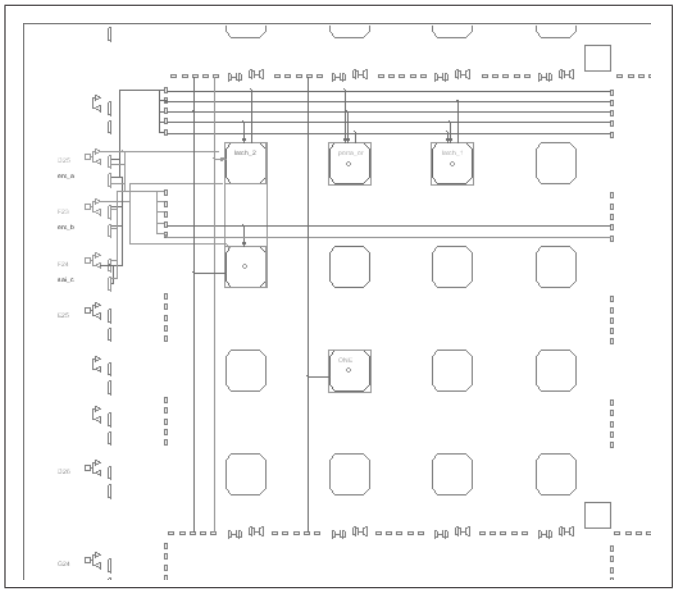

Figura B.4: Roteamento do ATO_2

oOo

Apêndice C

MAC - arquivos fontes

C.1 MAC - F.E. (principais)

C.1.1 reconfiguracao.v

```
// +FHDR----------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ---------------------------------------------------------------------------------
// NOME DO ARQUIVO : reconfiguracao.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ---------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      10-09-2004  Luiz Brunelli  Gerada versao inicial.
// 0.2      12-09-2004  Luiz Brunelli  Incluida reconfiguracao durante reset_b.
// 0.3      08-11-2004  Luiz Brunelli  Modificacoes para F.E. MAC.
// 0.4      20-11-2004  Luiz Brunelli  Correcao da entrada w22 para w21 para
//                                     z1 = x1w11 + x2w21, por problemas de coe-
//                                     rencia do texto da tese.
// ---------------------------------------------------------------------------------
// PALAVRAS CHAVES : DCS, DRL, RTR
// ---------------------------------------------------------------------------------
// PROPOSITO       : Modulo principal responsavel pelo controle da
//                   reconfiguracao dinamica.
// ---------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//   --- Relogios ---
//   T_CLK_RECONF_MARCA   [N/D]  : Clock de Reconfig. :   30      :     ns
//   T_CLK_OPERACAO_MARCA [N/D]  : Clock de Operacao  :   100     :     ns
//
//   --- Atos ---
```

```
//   T_EXECUCAO_XX        [N/D]  : Tempo de execucao do ato XX em ns.
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clk, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            :
//   Sintetizavel (S/N)       : N
//   Outras                   : N
// ------------------------------------------------------------------------------
// OBSERVACOES
//   Sugere-se automatizar a construcao deste modulo.
// ------------------------------------------------------------------------------
// LEGENDA
//   DCS = "Dynamic circuit switching".
//   DRL = "Dynamic reconfiguration logic".
//   RTR = "Run-time reconfiguration".
//   N/D = nao disponivel.
//   N/A = nao aplicavel.
// -FHDR----------------------------------------------------------------------

`timescale 1 ns / 100 ps // adotado como referencia inicial.

module reconfiguracao();

// ---------- Declaracao dos parametros --------------------------------------
parameter T_CLK_RECONF_MARCA   = 15;     // --- Clock de Reconfiguracao ---
                                         // T_CLK = 30ns (33.33 MHz)
                                         // T Marca = T Espaco

parameter T_CLK_OPERACAO_MARCA = 50;     // --- Clock de Operacao ---
                                         // T_CLK = 100ns ( 10.00 MHz)
                                         // T Marca = T Espaco
// --- Em outros modulos
defparam  UCR.T_LOAD_RESET_00 = 428;     // --- Cena 00 ---
defparam  UCR.T_LOAD_00       = 428;     // Multiplos T_CLK_RECONF (min = 1)
defparam  UCR.T_REMOCAO_00    = 428;

defparam  UCR.T_LOAD_01       =  84;     // --- Cena 01 ---
defparam  UCR.T_REMOCAO_01    =  84;     // Multiplos T_CLK_RECONF (min = 1)

defparam  UCR.T_LOAD_02       = 144;     // --- Cena 02 ---
defparam  UCR.T_REMOCAO_02    = 144;     // Multiplos T_CLK_RECONF (min = 1)

defparam  UCR.T_LOAD_03       =  49;     // --- Cena 03 ---
defparam  UCR.T_REMOCAO_03    =  49;     // Multiplos T_CLK_RECONF (min = 1)
//
// OBS: Os bitstream sao do tipo  "compress" , sem minimizacao de potencia,
//      sem pads E/S, {sem repetidores H e V, "route-wire", vias externas
//      externas as F.E.}.

// --- Tempo de Execucao dos ATOS
parameter  T_EXECUCAO_ATO_00   =      35.92 ;  // (default = 10)
parameter  T_EXECUCAO_ATO_01   =      31.3  ;  // (default = 10)
parameter  T_EXECUCAO_ATO_02   =       2.1  ;  // (default = 10)
//
// OBS: obtidos do arquivo C:\proj.lb\mac_fe_01\global\palco\palco.sts e
//      estimativa dos atrasos das interligacoes com os data_bar.

// ------------------------------------------------------------------------------
```

```
// Definicao das variaveis locais.
// --- Entradas e saida dos modulos reconfiguraveis
reg   [3:0] entrada_x1;    // apenas para simulacao.
reg   [3:0] entrada_w11;   // apenas para simulacao.
reg   [3:0] entrada_x2;    // apenas para simulacao.
reg   [3:0] entrada_w21;   // apenas para simulacao.
wire  [8:0] saida_z1;      // apenas para simulacao.

// --- Unidade de controle da reconfiguracao.
reg           sinal_inicio;           // Chaveia a cena sequencialmente.
wire          sinal_fim;              // Termino da reconfiguracao da cena.
wire          sinal_reconfigura;      // Reconfiguracao em processo.

wire          sinal_ctrl_cena_00_b;   // NL0 = fecha chave de isolacao.
wire          sinal_ctrl_cena_00_m1;  // NL0 = armazena x1 e w11.
wire          sinal_ctrl_cena_00_m4;  // NL0 = armazena x4 (x2) e w21.
wire [8*11:1] sinal_estado_cena_00;   // Transicao, Ativa, Inativa.

wire          sinal_ctrl_cena_01_b;   // NL0 = fecha chave de isolacao.
wire          sinal_ctrl_cena_01_y1;  // NL0 = armazena y1.
wire          sinal_ctrl_cena_01_y4;  // NL0 = armazena y4.
wire [8*11:1] sinal_estado_cena_01;   // Transicao, Ativa, Inativa.

wire          sinal_ctrl_cena_02_b;   // NL0 = fecha chave de isolacao.
wire [8*11:1] sinal_estado_cena_02;   // Transicao, Ativa, Inativa.

wire          sinal_ctrl_cena_03_b;   // NL0 = fecha chave de isolacao.
wire          sinal_ctrl_cena_03_z1;  // NL0 = armazena z1.
wire [8*11:1] sinal_estado_cena_03;   // Transicao, Ativa, Inativa.

reg           clock_reconf;           // Sensivel a borda de subida.
reg           clock_operacao;         // Sensivel a borda de subida.
reg           sinal_reset_b;          // Ativo em NL0 e assincrono.

// ---- Outras e Depuracao ---
reg           executa;                // Indica a execucao dos atos.

// ---- Fios entre modulos e chave de isolacao.

// Modulo 00
wire  [3:0] fio_00_cena_00_x1;
wire  [3:0] fio_00_cena_00_w11;
wire  [3:0] fio_00_cena_00_x4;
wire  [3:0] fio_00_cena_00_w21;
wire  [7:0] fio_00_cena_00_x1w11;
wire  [7:0] fio_00_cena_00_x4w21;

// Modulo 01
wire  [7:0]  fio_01_cena_01_ent_y1;
wire  [7:0]  fio_01_cena_01_sai_y1;
wire  [7:0]  fio_01_cena_01_ent_y4;
wire  [7:0]  fio_01_cena_01_sai_y4;

// Modulo 02
wire  [7:0] fio_02_cena_02_y1;
wire  [7:0] fio_02_cena_02_y4;
wire  [8:0] fio_02_cena_02_soma_y1y4;

// Modulo 03
wire  [8:0] fio_03_cena_03_ent_z1;
wire  [8:0] fio_03_cena_03_sai_z1;

// Interligacao entre os modulos.
```

```
// Modulo 00 e 01
wire  [7:0]  fio_M00_M01_x1w11;
wire  [7:0]  fio_M00_M01_x4w21;

// Modulo 01 e 02
wire [7:0]  fio_M01_M02_sai_y1;
wire [7:0]  fio_M01_M02_sai_y4;

// Modulo 02 e 03
wire  [8:0]  fio_M02_M03_soma_y1y4;

// Instanciacoes dos modulos
// --- Modulos da cena 00.
cena_00   M00   (                                          // modulo 00
                .cena_00_x1(fio_00_cena_00_x1[3:0]),
                .cena_00_w11(fio_00_cena_00_w11[3:0]),
                .cena_00_x4(fio_00_cena_00_x4[3:0]),
                .cena_00_w22(fio_00_cena_00_w21[3:0]),
                .cena_00_x1w11(fio_00_cena_00_x1w11[7:0]),
                .cena_00_x4w22(fio_00_cena_00_x4w21[7:0]),
                .cena_00_ctrl_m1(sinal_ctrl_cena_00_m1),
                .cena_00_ctrl_m4(sinal_ctrl_cena_00_m4)
                );

// --- Modulos da cena 01.
cena_01    M01  (                                          // modulo 01
                .cena_01_ent_y1(fio_01_cena_01_ent_y1[7:0]),
                .cena_01_sai_y1(fio_01_cena_01_sai_y1[7:0]),
                .cena_01_ent_y4(fio_01_cena_01_ent_y4[7:0]),
                .cena_01_sai_y4(fio_01_cena_01_sai_y4[7:0]),
                .cena_01_ctrl_y1(sinal_ctrl_cena_01_y1),
                .cena_01_ctrl_y4(sinal_ctrl_cena_01_y4)
                );

// --- Modulos da cena 02.
cena_02    M02 (                                          // modulo 02
               .cena_02_y1(fio_02_cena_02_y1[7:0]),
               .cena_02_y4(fio_02_cena_02_y4[7:0]),
               .cena_02_soma_y1y4(fio_02_cena_02_soma_y1y4[8:0])
               );

// --- Modulos da cena 03.
cena_03    M03  (                                          // modulo 03
                .cena_03_ent_z1(fio_03_cena_03_ent_z1[8:0]),
                .cena_03_sai_z1(fio_03_cena_03_sai_z1[8:0]),
                .cena_03_ctrl_z1(sinal_ctrl_cena_03_z1)
                );

// --- Unidade de Controle da Reconfiguracao.
unid_ctrl UCR   (
                .inicio(sinal_inicio),
                .fim(sinal_fim),
                .reconfigura(sinal_reconfigura),

                .ctrl_cena_00_b (sinal_ctrl_cena_00_b),
                .ctrl_cena_00_m1(sinal_ctrl_cena_00_m1),
                .ctrl_cena_00_m4(sinal_ctrl_cena_00_m4),
                .estado_cena_00 (sinal_estado_cena_00),

                .ctrl_cena_01_b (sinal_ctrl_cena_01_b),
                .ctrl_cena_01_y1(sinal_ctrl_cena_01_y1),
                .ctrl_cena_01_y4(sinal_ctrl_cena_01_y4),
                .estado_cena_01 (sinal_estado_cena_01),
```

```
                .ctrl_cena_02_b(sinal_ctrl_cena_02_b),
                .estado_cena_02(sinal_estado_cena_02),

                .ctrl_cena_03_b (sinal_ctrl_cena_03_b),
                .ctrl_cena_03_z1(sinal_ctrl_cena_03_z1),
                .estado_cena_03 (sinal_estado_cena_03),

                .clock(clock_reconf),
                .reset_b(sinal_reset_b)
                );

// --- Chaves de isolacao para a simulacao da reconfiguracao dinamica
// --- Modulo 00 ---
chave_isolacao4 M00_cena_00_x1 (                    // entrada cena_00_x1
                    .entrada4(entrada_x1[3:0]),
                    .saida4(fio_00_cena_00_x1[3:0]),
                    .reconfig4_b(sinal_ctrl_cena_00_b)
                    );
chave_isolacao4 M00_cena_00_w11 (                   // entrada cena_00_w11
                    .entrada4(entrada_w11[3:0]),
                    .saida4(fio_00_cena_00_w11[3:0]),
                    .reconfig4_b(sinal_ctrl_cena_00_b)
                    );
chave_isolacao4 M00_cena_00_x4 (                    // entrada cena_00_x1
                    .entrada4(entrada_x2[3:0]),
                    .saida4(fio_00_cena_00_x4[3:0]),
                    .reconfig4_b(sinal_ctrl_cena_00_b)
                    );
chave_isolacao4 M00_cena_00_w21 (                   // entrada cena_00_w11
                    .entrada4(entrada_w21[3:0]),
                    .saida4(fio_00_cena_00_w21[3:0]),
                    .reconfig4_b(sinal_ctrl_cena_00_b)
                    );
chave_isolacao8 M00_cena_00_x1w11 (                // saida cena_00_x1w11
                    .entrada8(fio_00_cena_00_x1w11[7:0]),
                    .saida8(fio_M00_M01_x1w11[7:0]),
                    .reconfig8_b(sinal_ctrl_cena_00_b)
                    );
chave_isolacao8 M00_cena_00_x4w21 (                 // saida cena_00_x4w21
                    .entrada8(fio_00_cena_00_x4w21[7:0]),
                    .saida8(fio_M00_M01_x4w21[7:0]),
                    .reconfig8_b(sinal_ctrl_cena_00_b)
                    );
// --- Modulo 01 ---
chave_isolacao8 M01_cena_01_ent_y1 (                // entrada
                     .entrada8(fio_M00_M01_x1w11[7:0]),
                     .saida8(fio_01_cena_01_ent_y1[7:0]),
                     .reconfig8_b(sinal_ctrl_cena_01_b)
                     );
chave_isolacao8 M01_cena_01_sai_y1 (                // entrada
                     .entrada8(fio_01_cena_01_sai_y1[7:0]),
                     .saida8(fio_M01_M02_sai_y1[7:0]),
                     .reconfig8_b(sinal_ctrl_cena_01_b)
                     );
chave_isolacao8 M01_cena_01_ent_y4 (                // saida
                     .entrada8(fio_M00_M01_x4w21[7:0]),
                     .saida8(fio_01_cena_01_ent_y4[7:0]),
                     .reconfig8_b(sinal_ctrl_cena_01_b)
                     );
chave_isolacao8 M01_cena_01_sai_y4 (                // saida
                     .entrada8(fio_01_cena_01_sai_y4[7:0]),
```

```
                      .saida8(fio_M01_M02_sai_y4[7:0]),
                      .reconfig8_b(sinal_ctrl_cena_01_b)
                      );
// --- Modulo 02 ---
chave_isolacao8 M02_cena_02_y1 (                    // entrada
                     .entrada8(fio_M01_M02_sai_y1[7:0]),
                     .saida8(fio_02_cena_02_y1[7:0]),
                     .reconfig8_b(sinal_ctrl_cena_02_b)
                     );
chave_isolacao8 M02_cena_02_y4 (                    // entrada
                     .entrada8(fio_M01_M02_sai_y4[7:0]),
                     .saida8(fio_02_cena_02_y4[7:0]),
                     .reconfig8_b(sinal_ctrl_cena_02_b)
                     );
chave_isolacao9 M02_cena_02_soma_y1y4 (             // saida
                     .entrada9(fio_02_cena_02_soma_y1y4[8:0]),
                     .saida9(fio_M02_M03_soma_y1y4[8:0]),
                     .reconfig9_b(sinal_ctrl_cena_02_b)
                     );
// --- Modulo 03 ---
chave_isolacao9 M03_cena_03_ent_z1 (                // entrada
                     .entrada9(fio_M02_M03_soma_y1y4[8:0]),
                     .saida9(fio_03_cena_03_ent_z1[8:0]),
                     .reconfig9_b(sinal_ctrl_cena_03_b)
                     );
chave_isolacao9 M03_cena_03_sai_z1 (                // entrada
                     .entrada9(fio_03_cena_03_sai_z1[8:0]),
                     .saida9(saida_z1[8:0]),
                     .reconfig9_b(sinal_ctrl_cena_03_b)
                     );

// Aplicacao dos estimulos.
// --- Geracao do relogio da arquitetura
initial begin
   clock_operacao = 1'b0;
   forever begin
      #(T_CLK_OPERACAO_MARCA) clock_operacao = ~clock_operacao;
   end
end

// --- Geracao do relogio do circuito de reconfiguracao
initial begin
   clock_reconf = 1'b0;
   forever begin
      #(T_CLK_RECONF_MARCA) clock_reconf = ~clock_reconf;
   end
end

// --- Geracao do Reset
initial begin
        sinal_reset_b = 1'b1;
   #230 sinal_reset_b = 1'b0;
   #50  sinal_reset_b = 1'b1;
end

// --- Vetores de Teste
initial begin
   // Valores iniciais daas entradas.
   sinal_inicio = 1'b0;
   executa = 1'b0;

   // Fim da reconfiguracao apos reset_b
```

```
wait ( sinal_fim);

// Apresentacao do ato 00.
@(negedge clock_reconf);
entrada_x1[3:0] = 4'h2; entrada_w11[3:0] = 4'h3;
entrada_x2[3:0] = 4'h4; entrada_w21[3:0] = 4'h5;

// Execucao do ato.
executa = 1'b1;
#(T_EXECUCAO_ATO_00);
executa = 1'b0;

// Apresentacao do ato 01.
@(negedge clock_reconf) sinal_inicio = 1'b1;
@(negedge clock_reconf) sinal_inicio = 1'b0;

// Aguarda o termino da reconfiguracao.
wait ( sinal_fim);

// Execucao do ato.
executa = 1'b1;
#(T_EXECUCAO_ATO_01);
executa = 1'b0;

// Retirada dos dados (ilustrativa)
entrada_x1[3:0] = 4'hx; entrada_w11[3:0] = 4'hx;
entrada_x2[3:0] = 4'hx; entrada_w21[3:0] = 4'hx;

// Apresentacao do ato 02.
@(negedge clock_reconf) sinal_inicio = 1'b1;
@(negedge clock_reconf) sinal_inicio = 1'b0;

// Aguarda o termino da reconfiguracao
wait ( sinal_fim);

// Execucao do ato
executa = 1'b1;
#(T_EXECUCAO_ATO_02);
executa = 1'b0;

// ---------------- Nova Corrida ------------------

// Apresentacao do ato 00
@(negedge clock_reconf) sinal_inicio = 1'b1;
@(negedge clock_reconf) sinal_inicio = 1'b0;

// Aguarda o termino da reconfiguracao
wait ( sinal_fim);

// Dados de entrada.
@(negedge clock_reconf);
entrada_x1[3:0] = 4'h5; entrada_w11[3:0] = 4'h7;
entrada_x2[3:0] = 4'hF; entrada_w21[3:0] = 4'h3;

// Execucao do ato.
#(T_EXECUCAO_ATO_00);

// Apresentacao do ato 01.
@ (negedge clock_reconf) sinal_inicio = 1'b1;
@ (negedge clock_reconf) sinal_inicio = 1'b0;

// Aguarda o termino da reconfiguracao.
wait ( sinal_fim);
```

```
   // Execucao do ato.
   #(T_EXECUCAO_ATO_01);

   // Retirada dos dados (ilustrativa)
   entrada_x1[3:0] = 4'hx; entrada_w11[3:0] = 4'hx;
   entrada_x2[3:0] = 4'hx; entrada_w21[3:0] = 4'hx;

   // Apresentacao do ato 02.
   @(negedge clock_reconf) sinal_inicio = 1'b1;
   @(negedge clock_reconf) sinal_inicio = 1'b0;

   // Aguarda o termino da reconfiguracao
   wait ( sinal_fim);

   // Execucao do ato
   #(T_EXECUCAO_ATO_02);

   // -------------- Nova Corrida ---------------------

   // Apresentacao do ato 00.
   @(negedge clock_reconf) sinal_inicio = 1'b1;
   @(negedge clock_reconf) sinal_inicio = 1'b0;

   // Aguarda o termino da reconfiguracao
   wait ( sinal_fim);
   #50;

   #100;  // Estetica da simulacao.

   // Fim do programa
   $stop(2);   // Encerra a simulacao.
end

endmodule   // Fim do modulo.

// --------------------------------------------------------------------------------
```

C.1.2 unid_ctrl.v

```
// +FHDR----------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ---------------------------------------------------------------------------------
// NOME DO ARQUIVO : unid_ctrl.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ---------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      10-09-2004  Luiz Brunelli  Gerada versao inicial.
// 0.1      11-09-2004  Luiz Brunelli  Chaveamento sequencial das cenas.
// 0.2      12-09-2004  Luiz Brunelli  Incluida a reconfiguracao apos reset_b.
// 0.3      07-11-2004  Luiz Brunelli  Modificacao para o caso F.E. MAC.
// ---------------------------------------------------------------------------------
// PALAVRAS CHAVES : UC, RTR
// ---------------------------------------------------------------------------------
// PROPOSITO       : Unidade de Controle responsavel pela mudanca de cenas.
// ---------------------------------------------------------------------------------
// PARAMETROS
//
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//
//   --- Cena 00 ---
//   T_LOAD_RESET_00      [1-?]  : T carga da Cena    :     1     : CLK_RECONF
//                                 apos o reset_b
//   T_LOAD_00            [1-?]  : T carga da Cena    :     1     : CLK_RECONF
//   T_REMOCAO_00         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   --- Cena 01 ---
//   T_LOAD_01            [1-?]  : T carga da Cena    :     1     : CLK_RECONF
//   T_REMOCAO_01         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   --- Cena 02 ---
//   T_LOAD_02            [1-?]  : T carga da Cena    :     1     : CLK_RECONF
//   T_REMOCAO_02         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   --- Cena 03 ---
//   T_LOAD_03            [1-?]  : T carga da Cena    :     1     : CLK_RECONF
//   T_REMOCAO_03         [1-?]  : T retirada do M.   :     1     : CLK_RECONF
//
//   OBS: De posse do arquivo modulo.md4, verifica-se o numero de linhas dele e
//        atualiza-se o valor do T_XXXXXXX. O valor minimo e de um  CLK_RECONF,
//        por causa da maquina sequencial.
//
//   OBS: Tempo de setup dos dados = 1 CLK_RECONF (adotado por simplicidade).
//
//   OBS: Como os atuais DR-FPGA nao permitem programacao em paralelo de recur-
//        sos logicos faz-se necessario programar modulos paralelizaveis
//        sequencialmente.
// ---------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clock, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : N/A
//   Sintetizavel (S/N)       : N
//   Outras                   : N
```

```
// ------------------------------------------------------------------------------
// LEGENDA
//   RTR = "Run-Time Reconfiguration"
//   DR-FPGA = "Dynamic Reconfiguration Field Programmable Gate Array"
//   N/D = nao disponivel
//   N/A = nao aplicavel
// -FHDR---------------------------------------------------------------------------

`timescale 1 ns / 100 ps // adotado como referencia inicial.

module unid_ctrl  (
                  inicio, fim,
                  reconfigura,
                  ctrl_cena_00_b,  ctrl_cena_00_m1,
                  ctrl_cena_00_m4, estado_cena_00,
                  ctrl_cena_01_b,  ctrl_cena_01_y1,
                  ctrl_cena_01_y4, estado_cena_01,
                  ctrl_cena_02_b,  estado_cena_02,
                  ctrl_cena_03_b,  ctrl_cena_03_z1,
                  estado_cena_03,
                  clock, reset_b
                  );

// ---------- Declaracao dos parametros ----------------------------------
parameter T_LOAD_RESET_00  =  1;  // --- Cena 00 ---
parameter T_LOAD_00        =  1;  // Multiplos do T_CLK_RECONF (min = 1).
parameter T_REMOCAO_00     =  1;

parameter T_LOAD_01        =  1;  // --- Cena 01 ---
parameter T_REMOCAO_01     =  1;  // Multipols do T_CLK_RECONF (min = 1).

parameter T_LOAD_02        =  1;  // --- Cena 02 ---
parameter T_REMOCAO_02     =  1;  // Multipols do T_CLK_RECONF (min = 1).

parameter T_LOAD_03        =  1;  // --- Cena 01 ---
parameter T_REMOCAO_03     =  1;  // Multipols do T_CLK_RECONF (min = 1).
// ------------------------------------------------------------------------

// Definicao dos sinais
input    inicio;         // Chaveia a cena sequencialmente (nao ciclicamente).
output   fim;            // Termino da reconfiguracao da cena.

output   reconfigura;      // Reconfiguracao em processo

output   ctrl_cena_00_b;   // NL0 = fecha chave de isolacao.
output   ctrl_cena_00_m1;  // NL0 = armazena x1 e w11.
output   ctrl_cena_00_m4;  // NL0 = armazena x4 e w22.
output   estado_cena_00;   // Estado da reconfiguracao: ON, OFF, XXX.

output   ctrl_cena_01_b;   // NL0 = fecha chave de isolacao.
output   ctrl_cena_01_y1;  // NL0 = armazena y1.
output   ctrl_cena_01_y4;  // NL0 = armazena y4.
output   estado_cena_01;   // Estado da reconfiguracao: ON, OFF, XXX.

output   ctrl_cena_02_b;   // NL0 = fecha chave de isolacao.
output   estado_cena_02;   // Estado da reconfiguracao: ON, OFF, XXX.

output   ctrl_cena_03_b;   // NL0 = fecha chave de isolacao.
output   ctrl_cena_03_z1;  // NL0 = armazena z1.
output   estado_cena_03;   // Estado da reconfiguracao: ON, OFF, XXX.

input    reset_b;        // Ativo em NL0 e assincrono.
```

```
input    clock;          // Clock de RECONFIGURACAO,sensivel a borda de subida.

reg            fim;
reg            reconfigura;

reg            ctrl_cena_00_b;
reg [8*11:1]   estado_cena_00; // Maximo 13 caracteres ASCII (b1234567890).
reg            ctrl_cena_00_m1;
reg            ctrl_cena_00_m4;

reg            ctrl_cena_01_b;
reg [8*11:1]   estado_cena_01; // Maximo 13 caracteres ASCII (b1234567890).
reg            ctrl_cena_01_y1;
reg            ctrl_cena_01_y4;

reg            ctrl_cena_02_b;
reg [8*11:1]   estado_cena_02; // Maximo 13 caracteres ASCII (b1234567890).

reg            ctrl_cena_03_b;
reg [8*11:1]   estado_cena_03; // Maximo 13 caracteres ASCII (b1234567890).
reg            ctrl_cena_03_z1;

// Descricao do Hardware
always @(posedge clock) begin
   if(!reset_b) begin
      // Posiciona cena +00 e +01
      // Estado 0 --- Condicao inicial (default)
      fim             <= 1'b0;
      reconfigura     <= 1'b0;

      ctrl_cena_00_b  <= 1'bz;   // cena inativa
      ctrl_cena_00_m1 <= 1'b0;   // dado armazenado (default)
      ctrl_cena_00_m4 <= 1'b0;
      estado_cena_00  <= " OFF";

      ctrl_cena_01_b  <= 1'bz;   // cena inativa
      ctrl_cena_01_y1 <= 1'b0;
      ctrl_cena_01_y4 <= 1'b0;
      estado_cena_01  <= " OFF";

      ctrl_cena_02_b  <= 1'bz;   // cena inativa
      estado_cena_02  <= " OFF";

      ctrl_cena_03_b  <= 1'bz;   // cena inativa
      ctrl_cena_03_z1 <= 1'b0;
      estado_cena_03  <= " OFF";

      // Estado 1 --- Preparacao da carga da cena_00
      @(posedge clock);
      fim             <= 1'b0;
      reconfigura     <= 1'b1;

      ctrl_cena_00_b  <= 1'b1;   // cena em transicao
      ctrl_cena_00_m1 <= 1'b0;
      ctrl_cena_00_m4 <= 1'b0;
      estado_cena_00  <= " XXX";

      ctrl_cena_01_b  <= 1'bz;   // cena inativa
      ctrl_cena_01_y1 <= 1'b0;
      ctrl_cena_01_y4 <= 1'b0;
      estado_cena_01  <= " OFF";

      ctrl_cena_02_b  <= 1'bz;   // cena intativa
      estado_cena_02  <= " OFF";
```

```
ctrl_cena_03_b  <= 1'bz;    // cena inativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " OFF";

// Estado 2
repeat (T_LOAD_RESET_00)   // Carga da Cena 00 e preparacao da 01.
   @(posedge clock);       // Bistream Full.
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'b0;   // cena ativa
ctrl_cena_00_m1 <= 1'b1;   // dados "fluindo"
ctrl_cena_00_m4 <= 1'b1;
estado_cena_00  <= " ON";

ctrl_cena_01_b  <= 1'b1;   // cena transicao
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " XXX";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'bz;   // cena inativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " OFF";

// Estado 3
repeat (T_LOAD_01)         // Carga da Cena 01
   @(posedge clock);       // Bistream windows e compress.
fim             <= 1'b1;
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'b0;   // cena ativa
ctrl_cena_00_m1 <= 1'b1;   // dados "fluindo
ctrl_cena_00_m4 <= 1'b1;
estado_cena_00  <= " ON";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b1;
ctrl_cena_01_y4 <= 1'b1;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'bz;   // cena inativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " OFF";

// Estado 4
// Obs: identico ao anterior a menos do sinal fim.
@(posedge clock);
fim             <= 1'b0;   // cena(s) em execucao
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'b0;
ctrl_cena_00_m1 <= 1'b1;
ctrl_cena_00_m4 <= 1'b1;
estado_cena_00  <= " ON";

ctrl_cena_01_b  <= 1'b0;
ctrl_cena_01_y1 <= 1'b1;
ctrl_cena_01_y4 <= 1'b1;
```

```
      estado_cena_01  <= " ON";

      ctrl_cena_02_b  <= 1'bz;
      estado_cena_02  <= " OFF";

      ctrl_cena_03_b  <= 1'bz;
      ctrl_cena_03_z1 <= 1'b0;
      estado_cena_03  <= " OFF";
   end   // fim do Reset

   else begin
      if(inicio) begin              // Mudanca de cena(s)
         // Posiciona cena -00, +01, +02 e +03
         // Estado 5 --- Armazena o resultado do processamento.
         @(posedge clock);
         fim             <= 1'b0;   // salva os dados das cena(s) em execucao.
         reconfigura     <= 1'b0;   // t_setup = 1 clock de reconfiguracao.

         ctrl_cena_00_b  <= 1'b0;
         ctrl_cena_00_m1 <= 1'b0;   // condicao default.
         ctrl_cena_00_m4 <= 1'b0;
         estado_cena_00  <= " ON";

         ctrl_cena_01_b  <= 1'b0;
         ctrl_cena_01_y1 <= 1'b0;   // armazena o valor de y1.
         ctrl_cena_01_y4 <= 1'b0;   // armazena o valor de y4.
         estado_cena_01  <= " ON";

         ctrl_cena_02_b  <= 1'bz;
         estado_cena_02  <= " OFF";

         ctrl_cena_03_b  <= 1'bz;
         ctrl_cena_03_z1 <= 1'b0;
         estado_cena_03  <= " OFF";

         // Estado 6 --- Inicio da transicao de cenas
         fim             <= 1'b0;
         reconfigura     <= 1'b1;

         ctrl_cena_00_b  <= 1'b1;   //cena em transicao
         ctrl_cena_00_m1 <= 1'b0;
         ctrl_cena_00_m4 <= 1'b0;
         estado_cena_00  <= " XXX";

         ctrl_cena_01_b  <= 1'b0;   // cena ativa
         ctrl_cena_01_y1 <= 1'b0;
         ctrl_cena_01_y4 <= 1'b0;
         estado_cena_01  <= " ON";

         ctrl_cena_02_b  <= 1'bz;
         estado_cena_02  <= " OFF";

         ctrl_cena_03_b  <= 1'bz;
         ctrl_cena_03_z1 <= 1'b0;
         estado_cena_03  <= " OFF";

         // Estado 6 --- Remocao da Cena 00 e preparacao Cena 02.
         repeat (T_REMOCAO_00)      // Aguarda a remocao da interseccao
            @(posedge clock);       // dos bitstreams consecutivos.
         fim             <= 1'b0;
         reconfigura     <= 1'b1;

         ctrl_cena_00_b  <= 1'bz;   // cena inativa
         ctrl_cena_00_m1 <= 1'b0;
```

```
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b1;   // cena em transicao
estado_cena_02  <= " XXX";

ctrl_cena_03_b  <= 1'bz;   // cena inativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " OFF";

// Estado 7 --- Carga da Cena 02, preparacao Cena 03.
repeat (T_LOAD_02 )        // Bitstreams window e compress.
   @(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b0;   // cena ativa
estado_cena_02  <= " ON";

ctrl_cena_03_b  <= 1'b1;   // cena em transicao
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " XXX";

// Estado 8 --- Carga da Cena 03
repeat (T_LOAD_03 )        // Bitstreams window e compress.
   @(posedge clock);
fim             <= 1'b1;
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b0;   // cena ativa
estado_cena_02  <= " ON";

ctrl_cena_03_b  <= 1'b0;   // cena ativa
ctrl_cena_03_z1 <= 1'b1;
estado_cena_03  <= " ON";

// Estado 9
// Obs: identico ao anterior a menos do sinal fim.
@(posedge clock);
```

```
fim             <= 1'b0;
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b0;   // cena ativa
estado_cena_02  <= " ON";

ctrl_cena_03_b  <= 1'b0;   // cena ativa
ctrl_cena_03_z1 <= 1'b1;
estado_cena_03  <= " ON";

wait(inicio);  // Aguarda o pedido de reconfiguracao.
// Posiciona cena: -01, -02 e + 03
// Estado 10 --- Armazena o resultado do processamento.
@(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b0;   // cena ativa
estado_cena_02  <= " ON";

ctrl_cena_03_b  <= 1'b0;   // cena ativa
ctrl_cena_03_z1 <= 1'b0;   // <-- armazena o valor de z1.
estado_cena_03  <= " ON";

// Estado 11 --- Inicio da transicao da cena 02.
@(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b0;   // cena ativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " ON";

ctrl_cena_02_b  <= 1'b1;   // cena em transicao
estado_cena_02  <= " XXX";

ctrl_cena_03_b  <= 1'b0;   // cena ativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " ON";
```

```
// Estado 12 --- Remocao da cena 02 e preparacao da 01.
repeat (T_REMOCAO_02 )     // Aguarda a remocao da
   @(posedge clock);       // interseccao dos bitstreams
fim             <= 1'b0;   // consecutivos.
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'b1;   // cena em transicao
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " XXX";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'b0;  // Resultado final do processamento.
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " ON";

// Estado 13 --- Remocao 01.
repeat (T_REMOCAO_01 )     // Aguarda a remocao da
   @(posedge clock);       // interseccao dos bitstreams
fim             <= 1'b1;   // consecutivos.
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'bz;   // cena inativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " OFF";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'b0;  // Resultado final do processamento.
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " ON";

// Estado 14
// Obs: identico ao anterior a menos do sinal fim.
@(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b0;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'bz;   // cena inativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " OFF";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
```

```
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'b0;  // Resultado final do processamento.
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " ON";

wait(inicio);  // Aguarda o pedido de reconfiguracao - novo inicio.
// Novo ciclo -03, +00 e +01
// Estado 15 --- Inicio da transicao de cenas.
@(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'bz;   // cena inativa
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " OFF";

ctrl_cena_01_b  <= 1'bz;   // cena inativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " OFF";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'b1;   // cena em transicao
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " XXX";

// Estado 16 --- Remocao da cena 03 e preparacao da cena 00.
repeat (T_REMOCAO_03)   // Aguarda a remocao da interseccao dos
   @(posedge clock);    // bitstreams consecutivos.
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'b1;   // cena em transicao
ctrl_cena_00_m1 <= 1'b0;
ctrl_cena_00_m4 <= 1'b0;
estado_cena_00  <= " XXX";

ctrl_cena_01_b  <= 1'bz;   // cena inativa
ctrl_cena_01_y1 <= 1'b0;
ctrl_cena_01_y4 <= 1'b0;
estado_cena_01  <= " OFF";

ctrl_cena_02_b  <= 1'bz;   // cena inativa
estado_cena_02  <= " OFF";

ctrl_cena_03_b  <= 1'bz;   // cena inativa
ctrl_cena_03_z1 <= 1'b0;
estado_cena_03  <= " OFF";

// Estado 17 --- Carga da cena 00 e preparacao da 01
repeat (T_LOAD_00)   // Bitstreams windows e compress.
   @(posedge clock);
fim             <= 1'b0;
reconfigura     <= 1'b1;

ctrl_cena_00_b  <= 1'b0;   // cena ativa
ctrl_cena_00_m1 <= 1'b1;
ctrl_cena_00_m4 <= 1'b1;
estado_cena_00  <= " ON";
```

```
        ctrl_cena_01_b  <= 1'b1;   // cena em transicao
        ctrl_cena_01_y1 <= 1'b0;
        ctrl_cena_01_y4 <= 1'b0;
        estado_cena_01  <= " XXX";

        ctrl_cena_02_b  <= 1'bz;   // cena inativa
        estado_cena_02  <= " OFF";

        ctrl_cena_03_b  <= 1'bz;  // cena inativa
        ctrl_cena_03_z1 <= 1'b0;
        estado_cena_03  <= " OFF";

        // Estado 18 --- Carga da cena 01
        repeat (T_LOAD_01)   // Bitstreams windows e compress.
           @(posedge clock);
        fim             <= 1'b1;
        reconfigura     <= 1'b0;

        ctrl_cena_00_b  <= 1'b0;   // cena ativa
        ctrl_cena_00_m1 <= 1'b1;
        ctrl_cena_00_m4 <= 1'b1;
        estado_cena_00  <= " ON";

        ctrl_cena_01_b  <= 1'b0;   // cena ativa
        ctrl_cena_01_y1 <= 1'b1;
        ctrl_cena_01_y4 <= 1'b1;
        estado_cena_01  <= " ON";

        ctrl_cena_02_b  <= 1'bz;   // cena inativa
        estado_cena_02  <= " OFF";

        ctrl_cena_03_b  <= 1'bz;   // cena inativa
        ctrl_cena_03_z1 <= 1'b0;
        estado_cena_03  <= " OFF";

        // Estado 19
        // Obs: identico ao anterior a menos do sinal fim.
        @(posedge clock);
        fim             <= 1'b0;
        reconfigura     <= 1'b0;

        ctrl_cena_00_b  <= 1'b0;   // cena ativa
        ctrl_cena_00_m1 <= 1'b1;
        ctrl_cena_00_m4 <= 1'b1;
        estado_cena_00  <= " ON";

        ctrl_cena_01_b  <= 1'b0;   // cena ativa
        ctrl_cena_01_y1 <= 1'b1;
        ctrl_cena_01_y4 <= 1'b1;
        estado_cena_01  <= " ON";

        ctrl_cena_02_b  <= 1'bz;   // cena inativa
        estado_cena_02  <= " OFF";

        ctrl_cena_03_b  <= 1'bz;   // cena inativa
        ctrl_cena_03_z1 <= 1'b0;
        estado_cena_03  <= " OFF";
      end
   end
end

endmodule  //Fim do programa
// ------------------------------------------------------------------------------
```

C.1.3 chave_isolacao4.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : chave_isolacao4.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      15-09-2004  Luiz Brunelli  Gerada versao inicial.
// 0.1      11-11-2004  Luiz Brunelli  Versao modificada para 4 chaves.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES : DCS, DRL
// ------------------------------------------------------------------------------
// PROPOSITO       : Criacao de 4 chaves de isolacao.
// ------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO   FAIXA     : DESCRICAO            : DEFAULT   : UNIDADES
//
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N
//   Dominio do Clock         : N
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : chave_isolacao
//   Sintetizavel (s/n)       : N
//   Outras                   : N
// -FHDR-------------------------------------------------------------------------

‘timescale 1 ns / 100 ps // adotado como referencia inicial.

module chave_isolacao4 (
                        entrada4,
                        saida4,
                        reconfig4_b
                        );

input    [3:0] entrada4;   // Entrada do sinal.
output   [3:0] saida4;     // Valores validos:
                           // X (durante a o periodo de transicao)
                           // Z (durante a o periodo de inatividade).

input    reconfig4_b;      // Valores validos:
                           // 0 = chave ativa,
                           // 1 = periodo de transicao.
                           // Z = periodo de inatividade.

// Instanciacoes dos modulos

chave_isolacao CH00_3  (
                        .entrada(entrada4[3]),
                        .saida(saida4[3]),
                        .reconfig_b(reconfig4_b)
                        );

chave_isolacao CH00_2  (
                        .entrada(entrada4[2]),
                        .saida(saida4[2]),
```

```
                        .reconfig_b(reconfig4_b)
                        );

chave_isolacao CH00_1   (
                        .entrada(entrada4[1]),
                        .saida(saida4[1]),
                        .reconfig_b(reconfig4_b)
                        );

chave_isolacao CH00_0   (
                        .entrada(entrada4[0]),
                        .saida(saida4[0]),
                        .reconfig_b(reconfig4_b)
                        );

endmodule

// ------------------------------------------------------------------------
```

C.1.4 cena_00.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : cena_00.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      12-11-2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :
// ------------------------------------------------------------------------------
// PROPOSITO       :
// ------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clk, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : multiplicador_1, multiplicaodr_4.
//   Sintetizavel (S/N)       : N
//   Outras                   : N
//
// -FHDR-------------------------------------------------------------------------

‘timescale 1 ns / 100 ps // adotado como referencia inicial.

module   cena_00  (
                  cena_00_x1,
                  cena_00_w11,
                  cena_00_x4,
                  cena_00_w22,
                  cena_00_x1w11,
                  cena_00_x4w22,
                  cena_00_ctrl_m1,
                  cena_00_ctrl_m4
                  );

// Definicao dos sinais
input    [3:0] cena_00_x1;       // Entrada de dados.
input    [3:0] cena_00_w11;
input    [3:0] cena_00_x4;
input    [3:0] cena_00_w22;
output   [7:0] cena_00_x1w11;    // Saida de dados.
output   [7:0] cena_00_x4w22;
input    cena_00_ctrl_m1;        // NL0 = armazena x1 e w11.
input    cena_00_ctrl_m4;        // NL0 = armazena x4 e w22.

// Descricao de hardware

// Inclui data_bar_x1, data_bar_w11 e multiplicador.
multiplicador_1  Modulo_M1   (
                             .pino_x1(cena_00_x1[3:0]),
```

```
                              .pino_w11(cena_00_w11[3:0]),
                              .pino_x1w11(cena_00_x1w11[7:0]),
                              .pino_ctrl_m1(cena_00_ctrl_m1)
                              );

// Inclui data_bar_x4, data_bar_w22 e multiplicador.
multiplicador_4   Modulo_M4   (
                              .pino_x4(cena_00_x4[3:0]),
                              .pino_w22(cena_00_w22[3:0]),
                              .pino_x4w22(cena_00_x4w22[7:0]),
                              .pino_ctrl_m4(cena_00_ctrl_m4)
                              );

endmodule   // Fim do programa
// ------------------------------------------------------------------------------
```

C.1.5 ato_00.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : ato_00.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      15-11-2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :
// ------------------------------------------------------------------------------
// PROPOSITO       :
// ------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clk, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : multiplicador_1, multiplicaodr_4.
//   Sintetizavel (S/N)       : N
//   Outras                   : N
//
// -FHDR-------------------------------------------------------------------------

`timescale 1 ns / 100 ps // adotado como referencia inicial.

module   ato_00  (
                  cena_00_x1,
                  cena_00_w11,
                  cena_00_x4,
                  cena_00_w22,
                  cena_01_sai_y1,
                  cena_01_sai_y4,
                  cena_00_ctrl_m1,
                  cena_00_ctrl_m4,
                  cena_01_ctrl_y1,
                  cena_01_ctrl_y4
                  );

// Definicao dos sinais
input    [3:0] cena_00_x1;       // Entrada de dados.
input    [3:0] cena_00_w11;
input    [3:0] cena_00_x4;
input    [3:0] cena_00_w22;
output   [7:0] cena_01_sai_y1;   // saida de dados
output   [7:0] cena_01_sai_y4;
input    cena_00_ctrl_m1;        // NL0 = armazena x1 e w11.
input    cena_00_ctrl_m4;        // NL0 = armazena x4 e w22.
input    cena_01_ctrl_y1;        // NL0 = armazena y1.
input    cena_01_ctrl_y4;        // NL0 = armazena y4.

// Variaveis locais
```

```
wire   [7:0] cena_00_x1w11;
wire   [7:0] cena_00_x4w22;

// Descricao de hardware

// Inclui data_bar_x1, data_bar_w11 e multiplicador.
multiplicador_1  Modulo_M1   (
                              .pino_x1(cena_00_x1[3:0]),
                              .pino_w11(cena_00_w11[3:0]),
                              .pino_x1w11(cena_00_x1w11[7:0]),
                              .pino_ctrl_m1(cena_00_ctrl_m1)
                              );

// Inclui data_bar_x4, data_bar_w22 e multiplicador.
multiplicador_4   Modulo_M4   (
                              .pino_x4(cena_00_x4[3:0]),
                              .pino_w22(cena_00_w22[3:0]),
                              .pino_x4w22(cena_00_x4w22[7:0]),
                              .pino_ctrl_m4(cena_00_ctrl_m4)
                              );

// data_bar_y1
y1 Modulo_Y1   (
               .pino_ent_y1(cena_00_x1w11[7:0]),
               .pino_ctrl_y1(cena_01_ctrl_y1),
               .pino_sai_y1(cena_01_sai_y1[7:0])
               );


// data_bar_y4
y4 Modulo_Y4   (
               .pino_ent_y4(cena_00_x4w22[7:0]),
               .pino_ctrl_y4(cena_01_ctrl_y4),
               .pino_sai_y4(cena_01_sai_y4[7:0])
               );


endmodule   // Fim do programa

// ------------------------------------------------------------------

// Declarações dos modulos a serem utilizados

module multiplicador_1 (                  // MACRO
                        pino_x1, pino_w11, pino_x1w11,
                        pino_ctrl_m1
                        );

input  [3:0]   pino_x1, pino_w11;         // entradas
input          pino_ctrl_m1;
output [7:0]   pino_x1w11;           // saidas

endmodule


module multiplicador_4 (                  // MACRO
                        pino_x4, pino_w22, pino_x4w22,
                        pino_ctrl_m4
                        );

input  [3:0]   pino_x4, pino_w22;         // entradas
input          pino_ctrl_m4;
```

```
output [7:0]   pino_x4w22;           // saidas

endmodule

module y1 (                    // MACRO
          pino_ent_y1,
          pino_sai_y1,
          pino_ctrl_y1
          );

input  [7:0]   pino_ent_y1;   // entradas
output [7:0]   pino_sai_y1;   // saidas
input          pino_ctrl_y1;  // controle

endmodule


module y4 (                    // MACRO
          pino_ent_y4,
          pino_sai_y4,
          pino_ctrl_y4
          );

input  [7:0]   pino_ent_y4;   // entradas
output [7:0]   pino_sai_y4;   // saidas
input          pino_ctrl_y4;  // controle

endmodule

// ------------------------------------------------
```

C.1.6 mac_fe_00.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : mac_fe_00.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      15-11-2004  Luiz Brunelli  Gerada versao inicial.
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :
// ------------------------------------------------------------------------------
// PROPOSITO       :
// ------------------------------------------------------------------------------
// PARAMETROS
//   NOME DO PARAMETRO    FAIXA  : DESCRICAO          : DEFAULT   : UNIDADES
//
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : sincrono, ativo em NL0.
//   Dominio do Clock         : clk, borda de subida.
//   Timing Critico           : N/D
//   Caracteristicas de Teste : N/D
//   Interfaces Assincronas   : N
//   Instanciacoes            : multiplicador_1, multiplicaodr_4.
//   Sintetizavel (S/N)       : N
//   Outras                   : N
//
// -FHDR-------------------------------------------------------------------------

`timescale 1 ns / 100 ps // adotado como referencia inicial.

module mac_fe_00  (
                  pino_x1, pino_x2,
                  pino_w11, pino_w12, pino_w13,
                  pino_w21, pino_w22, pino_w23,
                  pino_y1,  pino_y2,  pino_y3,
                  pino_y4,  pino_y5,  pino_y6,
                  pino_ctrl_m1,
                  pino_ctrl_m4,
                  pino_ctrl_y1,
                  pino_ctrl_y4,
                  );

input  [3:0]   pino_x1, pino_x2,
               pino_w11, pino_w12, pino_w13,
               pino_w21, pino_w22, pino_w23;

output [7:0]   pino_y1,  pino_y2,  pino_y3,
               pino_y4,  pino_y5,  pino_y6;

input          pino_ctrl_m1,
               pino_ctrl_m4,
               pino_ctrl_y1,
               pino_ctrl_y4;


// ---- Descrição do Hardware --------------------------------------
```

```
ato_00  MATO_00_0 (
                  .cena_00_x1(pino_x1[3:0]),
                  .cena_00_w11(pino_w11[3:0]),
                  .cena_00_x4(pino_x2[3:0]),
                  .cena_00_w22(pino_w21[3:0]),
                  .cena_01_sai_y1(pino_y1[7:0]),
                  .cena_01_sai_y4(pino_y4[7:0]),
                  .cena_00_ctrl_m1(pino_ctrl_m1),
                  .cena_00_ctrl_m4(pino_ctrl_m4),
                  .cena_01_ctrl_y1(pino_ctrl_y1),
                  .cena_01_ctrl_y4(pino_ctrl_y4)
                  );

ato_00  MATO_00_1 (
                  .cena_00_x1(pino_x1[3:0]),
                  .cena_00_w11(pino_w12[3:0]),
                  .cena_00_x4(pino_x2[3:0]),
                  .cena_00_w22(pino_w22[3:0]),
                  .cena_01_sai_y1(pino_y2[7:0]),
                  .cena_01_sai_y4(pino_y5[7:0]),
                  .cena_00_ctrl_m1(pino_ctrl_m1),
                  .cena_00_ctrl_m4(pino_ctrl_m4),
                  .cena_01_ctrl_y1(pino_ctrl_y1),
                  .cena_01_ctrl_y4(pino_ctrl_y4)
                  );

ato_00  MATO_00_2 (
                  .cena_00_x1(pino_x1[3:0]),
                  .cena_00_w11(pino_w13[3:0]),
                  .cena_00_x4(pino_x2[3:0]),
                  .cena_00_w22(pino_w23[3:0]),
                  .cena_01_sai_y1(pino_y3[7:0]),
                  .cena_01_sai_y4(pino_y6[7:0]),
                  .cena_00_ctrl_m1(pino_ctrl_m1),
                  .cena_00_ctrl_m4(pino_ctrl_m4),
                  .cena_01_ctrl_y1(pino_ctrl_y1),
                  .cena_01_ctrl_y4(pino_ctrl_y4)
                  );

endmodule

// ---------------------------------------------------------------

// Declarações dos modulos a serem utilizados

module   ato_00  (
                  cena_00_x1,
                  cena_00_w11,
                  cena_00_x4,
                  cena_00_w22,
                  cena_01_sai_y1,
                  cena_01_sai_y4,
                  cena_00_ctrl_m1,
                  cena_00_ctrl_m4,
                  cena_01_ctrl_y1,
                  cena_01_ctrl_y4
                  );

// Definicao dos sinais
input    [3:0] cena_00_x1;       // Entrada de dados.
input    [3:0] cena_00_w11;
```

```
input    [3:0] cena_00_x4;
input    [3:0] cena_00_w22;
output   [7:0] cena_01_sai_y1;   // saida de dados
output   [7:0] cena_01_sai_y4;
input    cena_00_ctrl_m1;        // NL0 = armazena x1 e w11.
input    cena_00_ctrl_m4;        // NL0 = armazena x4 e w22.
input    cena_01_ctrl_y1;        // NL0 = armazena y1.
input    cena_01_ctrl_y4;        // NL0 = armazena y4.

endmodule

// ------------------------------------------------------------------
```

C.2 MAC - tradicional

C.2.1 tb_mac_tradicional.v

```
// +FHDR-----------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ----------------------------------------------------------------------------
// NOME DO ARQUIVO : tb_mac_tradicional.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ----------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.1      20/11/2004  Luiz Brunelli  Gerada versao inicial.
// 0.2      29/11/2004  Luiz Brunelli  Retirado pino_ctrl_latch.
//
// ----------------------------------------------------------------------------
// PALAVRAS CHAVES :  MAC
// ----------------------------------------------------------------------------
// PROPOSITO       :  Bancada de teste do MAC tradicional
// ----------------------------------------------------------------------------
// PARAMETROS
//     NOME DO PARAMETRO   :FAIXA     : DESCRICAO         : DEFAULT   : UNIDADES
//     T_EXECUCAO          [N/D]        Tempo de execucao do modulo     ns
//
// ----------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N/A.
//   Dominio do Clock         : N/A.
//   Timing Critico           : N/D.
//   Caracteristicas de Teste : N/D.
//   Interfaces Assincronas   : N/A.
//   Instanciacoes            : N/D.
//   Sintetizavel (s/n)       : S
//   Outras                   : N/A.
// -FHDR-----------------------------------------------------------------------

`timescale 1 ns / 100 ps    // adotado como referencia inicial.

module bancada_teste ();

parameter   T_EXECUCAO = 76.08;  // Tempo de propagacao do modulo.
                                 // C:\proj.lb\mac_01\figaro\mac_tradicional.pdl
// Variaveis locais
reg   [3:0]    sinal_pino_x1, sinal_pino_x2,                  // Entradas.
               sinal_pino_w11, sinal_pino_w12, sinal_pino_w13,
               sinal_pino_w21, sinal_pino_w22, sinal_pino_w23;
wire  [8:0]    sinal_pino_z1, sinal_pino_z2, sinal_pino_z3;    // Saidas.

reg   [8:0]    valor_z1, valor_z2, valor_z3;                  // Calculado.
reg            erro;

// Modulo a ser testado
mac_tradicional   U1 (
                    .pino_x1(sinal_pino_x1),
                    .pino_x2(sinal_pino_x2),
                    .pino_w11(sinal_pino_w11),
                    .pino_w12(sinal_pino_w12),
                    .pino_w13(sinal_pino_w13),
```

```
                    .pino_w21(sinal_pino_w21),
                    .pino_w22(sinal_pino_w22),
                    .pino_w23(sinal_pino_w23),
                    .pino_z1(sinal_pino_z1),
                    .pino_z2(sinal_pino_z2),
                    .pino_z3(sinal_pino_z3)
                    );

// Aplicacao dos Estimulos
initial
begin
   // Inicializacoes.
   erro =1'b0;
   #100;

   // Carga dos pesos
   sinal_pino_w11 = 4'h2; sinal_pino_w12 = 4'h4;  sinal_pino_w13 = 4'h6;
   sinal_pino_w21 = 4'h3; sinal_pino_w22 = 4'h5;  sinal_pino_w23 = 4'h7;

   // Variaveis de entrada.

   // ---- Estimulo ----
   sinal_pino_x1 = 4'h1; sinal_pino_x2 = 4'h2;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10;
      erro = 1'b0;
   end

   // ---- Estimulo ----
   sinal_pino_x1 = 4'h3; sinal_pino_x2 = 5'h4;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10;
      erro = 1'b0;
   end

   // ---- Estimulo ----
   sinal_pino_x1 = 'h5; sinal_pino_x2 = 4'h6;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
```

```
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10;
      erro = 1'b0;
   end

   // ---- Estimulo ----
   sinal_pino_x1 = 4'h7; sinal_pino_x2 = 4'h8;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10;
      erro = 1'b0;
   end

   // ---- Estimulo ----
   sinal_pino_x1 = 4'h3; sinal_pino_x2 = 4'h5;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10;
      erro = 1'b0;
   end

   #50;     // Efeito cosmetico.

   // ---- Estimulo ----
   sinal_pino_x1 = 4'h3; sinal_pino_x2 = 4'ha;
   // Calculo do valor teorico.
   valor_z1 = sinal_pino_x1 * sinal_pino_w11 + sinal_pino_x2 * sinal_pino_w21;
   valor_z2 = sinal_pino_x1 * sinal_pino_w12 + sinal_pino_x2 * sinal_pino_w22;
   valor_z3 = sinal_pino_x1 * sinal_pino_w13 + sinal_pino_x2 * sinal_pino_w23;

   #T_EXECUCAO;

   // Comparacao
   if (valor_z1 != sinal_pino_z1) begin
      erro = 1'b1;
      #10
      erro = 1'b0;
   end

   #200;    // Efeito cosmetico.
end

endmodule
```

C.2.2 mac_tradicional.v

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : mac_tradicional.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.1      19/02/2004  Luiz Brunelli  Gerada versao inicial.
// 0.2      24/03/2004  Luiz Brunelli  Padronização de nome de sinais e modulos.
// 0.3      16/11/2004  Luiz Bunelli   Mudanca no nome dos sinais.
// 0.4      19/11/2004  Luiz Brunelli  Acrescentado novos modulos.
// 0.5      29/11/2004  Luiz Brunelli  Retirado pino_ctrl_latch.
//
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :  MAC
// ------------------------------------------------------------------------------
// PROPOSITO       :  z1 = x1*w11 + x2*w21 (tradicional)
//                    z2 = x1*w12 + x2*w22
//                    z3 = x1*w13 + x2*w23
//
// OBS: Foram colocados "latchs" nas entradas e saidas para estudar o impacto
//      apenas das conexoes x as barras de dados das F.E.
//
// ------------------------------------------------------------------------------
// PARAMETROS
//     NOME DO PARAMETRO   FAIXA     : DESCRICAO          : DEFAULT   : UNIDADES
//     nenhum
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N/A.
//   Dominio do Clock         : N/A.
//   Timing Critico           : N/D.
//   Caracteristicas de Teste : N/D.
//   Interfaces Assincronas   : N/A.
//   Instanciacoes            : N/D.
//   Sintetizavel (s/n)       : S
//   Outras                   : N/A.
// -FHDR-------------------------------------------------------------------------

module mac_tradicional  (
                        pino_x1, pino_x2,
                        pino_w11, pino_w12, pino_w13,
                        pino_w21, pino_w22, pino_w23,
                        pino_z1, pino_z2, pino_z3
                        );

input   [3:0]  pino_x1, pino_x2,
               pino_w11, pino_w12, pino_w13,
               pino_w21, pino_w22, pino_w23;

output  [8:0]  pino_z1, pino_z2, pino_z3;

wire           pino_ctrl_latch;

//Variaveis internas.
wire [3:0]    x1, x2;
wire [3:0]    w11, w12, w13, w21, w22, w23;
wire [7:0]    x1w11, x2w21, x1w12, x2w22, x1w13, x2w23;
```

```
wire [8:0]    z1, z2, z3;

// Descricao do Hardware.

assign
   pino_ctrl_latch = 1'b1;    // latch em transparencia.

// ---------------------------------------------
latch_4 data_x1   (                          // x1
                  .DATA(pino_x1[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(x1[3:0])
                  );

latch_4 data_x2   (                           // x2
                  .DATA(pino_x2[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(x2[3:0])
                  );

// ---------------------------------------------
latch_4 data__w11 (                          // w11
                  .DATA(pino_w11[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w11[3:0])
                  );

latch_4 data__w12 (                          // w12
                  .DATA(pino_w12[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w12[3:0])
                  );

latch_4 data__w13 (                         // w13
                  .DATA(pino_w13[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w13[3:0])
                  );

latch_4 data_w21  (                        // w21
                  .DATA(pino_w21[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w21[3:0])
                  );

latch_4 data_w22  (                          // w22
                  .DATA(pino_w22[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w22[3:0])
                  );

latch_4 data_w23  (                          // w23
                  .DATA(pino_w23[3:0]),
                  .ENABLE(pino_ctrl_latch),
                  .Q(w23[3:0])
                  );

//--------------- z1 = x1w11 + x2w21 --------------
mult_4 multiplicador_1 (
                       .DataA( x1[3:0]),
                       .DataB( w11[3:0]),
                       .PRODUCT(x1w11[7:0])
                       );
```

```
mult_4 multiplicador_2 (
                        .DataA( x2[3:0]),
                        .DataB( w21[3:0]),
                        .PRODUCT(x2w21[7:0])
                        );

soma_9 somador_1       (
                        .DATAA({1'b0,x1w11[7:0]}),
                        .DATAB({1'b0,x2w21[7:0]}),
                        .SUM(z1[8:0])
                        );

latch_9 data_z1        (
                        .DATA(z1[8:0]),
                        .ENABLE(pino_ctrl_latch),
                        .Q(pino_z1[8:0])
                        );

//--------------- z2 = x1w12 + x2w22 --------------
mult_4 multiplicador_3 (
                        .DataA( x1[3:0]),
                        .DataB( w12[3:0]),
                        .PRODUCT(x1w12[7:0])
                        );

mult_4 multiplicador_4 (
                        .DataA( x2[3:0]),
                        .DataB( w22[3:0]),
                        .PRODUCT(x2w22[7:0])
                        );

soma_9 somador_2       (
                        .DATAA({1'b0,x1w12[7:0]}),
                        .DATAB({1'b0,x2w22[7:0]}),
                        .SUM(z2[8:0])
                        );

latch_9 data_z2        (
                        .DATA(z2[8:0]),
                        .ENABLE(pino_ctrl_latch),
                        .Q(pino_z2[8:0])
                        );

//--------------- z3 = x1w13 + x2w23 --------------
mult_4 multiplicador_5 (
                        .DataA( x1[3:0]),
                        .DataB( w13[3:0]),
                        .PRODUCT(x1w13[7:0])
                        );

mult_4 multiplicador_6 (
                        .DataA( x2[3:0]),
                        .DataB( w23[3:0]),
                        .PRODUCT(x2w23[7:0])
                        );

soma_9 somador_3       (
                        .DATAA({1'b0,x1w13[7:0]}),
                        .DATAB({1'b0,x2w23[7:0]}),
                        .SUM(z3[8:0])
                        );

latch_9 data_z3        (
```

```
                        .DATA(z3[8:0]),
                        .ENABLE(pino_ctrl_latch),
                        .Q(pino_z3[8:0])
                        );
endmodule

// ------------------------------------------------

// Declarações dos modulos a serem utilizados

// ------------------------------------------------
module  latch_4 (
                DATA,
                ENABLE,
                Q
                );

input  [3:0] DATA;
input        ENABLE;    // 0 = latch.
output [3:0] Q;

endmodule

// ------------------------------------------------
module  latch_9 (
                DATA,
                ENABLE,
                Q
                );

input  [8:0] DATA;
input        ENABLE;    // 0 = latch.
output [8:0] Q;

endmodule

// ------------------------------------------------
module mult_4 (
               DataA,
               DataB,
               PRODUCT
               );

input  [3:0] DataA, DataB;    // entradas do multiplicador
output [7:0] PRODUCT;         // produto

endmodule

// ------------------------------------------------

module soma_9 (
               DATAA,
               DATAB,
               SUM
               );

input  [8:0] DATAA, DATAB;
output [8:0] SUM;

endmodule

// ------------------------------------------------
```

C.3 Estudo das interconexões

C.3.1 Projeto tradicional (exemplo)

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : mac_t_21.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      21/12/2004  Luiz Brunelli  Gerada versao inicial.
//
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :  MAC (sintetizado com o EDS da Everest Design System)
// ------------------------------------------------------------------------------
// PROPOSITO       : Estudo das interconexoes
//
// ------------------------------------------------------------------------------
// PARAMETROS
//     NOME DO PARAMETRO   FAIXA     : DESCRICAO          : DEFAULT   : UNIDADES
//     nenhum
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N/A.
//   Dominio do Clock         : N/A.
//   Timing Critico           : N/D.
//   Caracteristicas de Teste : N/D.
//   Interfaces Assincronas   : N/A.
//   Instanciacoes            : N/D.
//   Sintetizavel (s/n)       : S
//   Outras                   : N/A.
// -FHDR-------------------------------------------------------------------------

module mac_t_21 (
                x1, x2,
                w1, w2,
                z1
                );

input   [3:0]   x1, x2,
                w1, w2;

output  [15:0]  z1;


// Descricao do Hardware.

assign
   z1[7: 0] = x1[3:0]*w1[3:0],
   z1[15:8] = x2[3:0]*w2[3:0];

endmodule

// ------------------------------------------------
```

```
// +FHDR-------------------------------------------------------------------------
// Luiz Brunelli Copyright (c)
// Universidade Federal da Paraiba
// ------------------------------------------------------------------------------
// NOME DO ARQUIVO : mac_t_32.v
// DEPARTAMENTO    : DEE/DSC
// AUTOR           : Luiz Brunelli
// EMAIL DO AUTOR  : lubru@dee.ufpb.br
// ------------------------------------------------------------------------------
// HISTORICO DAS REVISOES
// VERSAO   DATA        AUTOR          DESCRICAO
// 0.0      21/12/2004  Luiz Brunelli  Gerada versao inicial.
//
// ------------------------------------------------------------------------------
// PALAVRAS CHAVES :  MAC (sintetizado com o EDS da Everest Design System)
// ------------------------------------------------------------------------------
// PROPOSITO       : Estudo das interconexoes
//
// ------------------------------------------------------------------------------
// PARAMETROS
//     NOME DO PARAMETRO   FAIXA     : DESCRICAO          : DEFAULT   : UNIDADES
//     nenhum
// ------------------------------------------------------------------------------
// REUSO
//   Estrategia do Reset      : N/A.
//   Dominio do Clock         : N/A.
//   Timing Critico           : N/D.
//   Caracteristicas de Teste : N/D.
//   Interfaces Assincronas   : N/A.
//   Instanciacoes            : N/D.
//   Sintetizavel (s/n)       : S
//   Outras                   : N/A.
// -FHDR-------------------------------------------------------------------------

module mac_t_32 (
                x1, x2, x3,
                w11, w12, w21, w22, w31, w32,
                z1, z2
                );

input   [3:0]   x1, x2, x3,
                w11, w12, w21, w22, w31, w32;

output  [23:0]  z1, z2;


// Descricao do Hardware.

assign
   z1[ 7: 0] = x1[3:0]*w11[3:0],
   z1[15: 8] = x2[3:0]*w21[3:0],
   z1[23:16] = x3[3:0]*w31[3:0],

   z2[ 7: 0] = x1[3:0]*w12[3:0],
   z2[15: 8] = x2[3:0]*w22[3:0],
   z2[23:16] = x3[3:0]*w32[3:0];

endmodule

// ------------------------------------------------
```

```
>>>
FIGARO STATISTICS FILE [RESUMO]
====================
Date And Time  : December 21, 2004 at 4:21:24 pm
Device Type    : AT40K40-2BGC
Figaro Version : Atmel 7.5 (patch level 9 applied)

Design Statistics for " mac_t_21 "
Design Step : Initial Route
...
Number of Flip-Flops :                        0
Number of Gates :                             32
Number of Macro Wires :                       0
Number of Route Wires :                       6    <--- células usada no roteamento.
Number of Buses :                             99   <--- número de interligações.
     Local Buses :                            68
     Express Buses :                          31
...
-------------------------------------------
Arquivo: C:\proj.1b\mac_t_21\mac_t_21_a.sts
-------------------------------------------

>>>
Atmel Figaro version 7.5 (patch level 9 applied) [ RESUMO ]
Path Analysis December 21, 2004 at 4:21:47 pm
Design: mac_t_21
Part: A (AT40K40-2BGC)
Imported from: c:\proj.1b\mac_t_21\mac_t_21.edf
...
Delay Path Report Section
-------------------------
  Long Delay Paths
  ----------------
    Path #1
      Delay = 36.57ns
      U32 PAD -> U1 PAD
      MACRO U32: (IBUF) PORT: PAD                                        \_  0.00ns
      MACRO U32: (IBUF) ARC: PAD -> Q                          F 1.07ns \_  1.07ns
      NET   NET16: U32 Q -> H DATAB[0]                         F 8.05ns \_  9.12ns
      CORE  H/X2Y1/$E1 ARC: Y -> H                             F 2.15ns \_ 13.05ns
      NET   CARRY6: H/X2Y1/$E1 H -> H/X2Y2/$E1 Y               F 0.10ns \_ 13.15ns
      CORE  H/X2Y2/$E1 ARC: Y -> G                             F 2.10ns \_ 15.25ns
      NET   PPSO10: H/X2Y2/$E1 G -> H/X1Y3/$E1 X               F 0.09ns \_ 15.34ns
      CORE  H/X1Y3/$E1 ARC: X -> G                             F 2.15ns \_ 17.49ns
      NET   PPSO13: H/X1Y3/$E1 G -> H/Product3/$E1 X           F 2.72ns \_ 20.21ns
      CORE  H/Product3/$E1 ARC: X -> H                         F 2.36ns \_ 22.57ns
      NET   ADDCARRY3: H/Product3/$E1 H -> H/Product2/$E1 X    F 0.09ns \_ 22.66ns
      CORE  H/Product2/$E1 ARC: X -> H                         F 1.55ns \_ 24.21ns
      NET   ADDCARRY2: H/Product2/$E1 H -> H/Product1/$E1 X    F 0.10ns \_ 24.31ns
      CORE  H/Product1/$E1 ARC: X -> H                         F 1.55ns \_ 25.86ns
      NET   NET32: H PRODUCT[7] -> U1 A                        F 9.03ns \_ 34.89ns
      MACRO U1: (OBUF) ARC: A -> PAD                           F 1.68ns \_ 36.57ns

      logic delay = 14.61ns (40%)
      route delay = 21.96ns (60%)  <--- Tempo de atraso introduzido pelo roteamento
...
-------------------------------------------
Arquivo: C:\proj.1b\mac_t_21\mac_t_21_a.pdl
-------------------------------------------
```

C.3.2 Projeto utilizando F.E

```
>>>
FIGARO STATISTICS FILE [RESUMO]
====================
Date And Time  : December 16, 2004 at 2:25:37 pm
Device Type    : AT40K05-2RQC
Figaro Version : Atmel 7.5 (patch level 9 applied)

Design Statistics for " bmulfe "
Design Step : Optimize Route
...
Number of Flip-Flops :                          0
Number of Gates :                               33
Number of Macro Wires :                         0
Number of Route Wires :                         5    <==
Number of Buses :                               75   <==
     Local Buses :                              59
     Express Buses :                            16
...
-------------------------------------------------
Diretorio: C:\proj.lb\mac_fe_03\figaro\bmulfe.sts
-------------------------------------------------

>>>
Atmel Figaro version 7.5 (patch level 9 applied) [RESUMO]
Path Analysis December 16, 2004 at 2:27:55 pm
Design: bmulfe
Part: A (AT40K05-2RQC)
Imported from: c:\proj.lb\mac_fe_03\figaro\bmulfe.edf

Delay Path Report Section
-------------------------
  Long Delay Paths
  ----------------
    Path #1
      Delay = 35.24ns
      ...
      logic delay = 15.30ns (43%)
      route delay = 19.94ns (57%)   <==
  ...
-------------------------------------------------
Diretorio: C:\proj.lb\mac_fe_03\figaro\bmulfe.pdl
-------------------------------------------------

>>>
Macro information for library: c:\proj.lb\mac_fe_03\global\mac_fe_estudos [RESUMO]
Created December 16, 2004 10:27:16 am
...
Macro created December 16, 2004 2:29:36 pm
Macro Statistics
Name: bmulfe
Speed (MHz): 28.3768
Delay (ns): 35.24
Cells: 38
Size (x*y): 8x8
Power (mA/MHz): 0.1974
(80% duty cycle)
...
-----------------------------------------------------------------------
Diretorio: c:\proj.lb\mac_fe_03\global\mac_fe_estudos\mac_fe_estudos.sts
-----------------------------------------------------------------------
```

Apêndice D

Contribuições adicionais desta tese

D.1 Estudo, projeto e síntese em HDL de uma Rede Neuronal

Resumo: Neste trabalho de Projeto e Pesquisa descreve-se uma rede neuronal artificial de duas camadas que resolve o problema do Ou-Exclusivo proposto por Minsky & Papert para o Perceptron de Rosenblatt e que utiliza a Linguagem de Descrição de Hardware Verilog (sintetizável) para descrever a fase de execução da rede.

Palavras chaves: Neurocomputação, FPGA, RNA, Verilog.

Contribuições:

- *Estudo, projeto e Implementação de uma Rede Neuronal Reprogramável Dinamicamente*. Workshop do PROCAD - Sistemas de Aquisição e Tratamento de Sensores envolvendo a UFPA, UFMA, UFBA e UFPB realizado em Campina Grande (PB) em 11 de dezembro de 2001;

- *Estudo, projeto e Implementação de uma Rede Neuronal Reprogramável Dinamicamente*. Workshop do PROCAD - Uma Metodologia para o Desenvolvimento de Sistemas Embutidos: Gerando Competências e Suportando Propriedade Intelectual envolvendo a UFPE, USP, Unicamp e UFPB, realizado em Recife(PE) em 20 de fevereiro de 2002;

- *Estudo de caso voltado a Reconfiguração Dinâmica em Redes Neuronais*. Workshop do PROCAD - Uma Metodologia para o Desenvolvimento de Sistemas Embutidos: Gerando Competências e Suportando Propriedade Intelectual envolvendo a UFPE, USP, Unicamp e UFPB, realizado em João Pessoa(PB) em 19 de Junho de 2002.

D.2 Estudo, projeto e implementação de uma interface PCI mínima em FPGA

Resumo: Este trabalho tem o objetivo de servir de base para o estudo do barramento PCI visando futuros projetos e desenvolvimentos de placas neste padrão.

Palavras chaves: PCI, IP, FPGA.

Contribuições:

- Workshop do PROCAD - Uma Metodologia para o Desenvolvimento de Sistemas Embutidos: Gerando Competência e Suportando Propriedade Intelectual envol-

vendo a UFPE, USP, Unicamp e UFCG, realizado em Recife (PE) de 29 a 31 de janeiro de 2003;

- I Encontro Regional em Instrumentação e Metrologia Científica, realizado em Campina Grande (PB) de 15 a 18 de dezembro de 2003.

- Motivação de uma dissertação de mestrado: *Projeto de um núcleo IP de conversão da interface PCI para interface OCP-IP* de Ricardo Brandão Sampaio e apresentada na UFCG em 20 de fevereiro de 2004.

D.3 Estudo, projeto e implementação de um *backend* de memória RAM, com suporte ao modo rajada, para o barramento PCI

Resumo: Este trabalho implementa uma memória RAM que se comunica com o barramento PCI e suporta transferências de dados no modo rajada.

Palavras chaves: PCI, Rajada, FPGA.

Contribuições:

- I Encontro Regional em Instrumentação e Metrologia Científica, realizado em Campina Grande (PB) de 15 a 18 de dezembro de 2003.

D.4 Figuras de Execução

Durante o estudo e desenvolvimento dos conceitos das Figuras de Execução foram feitas as seguintes contribuições:

- *Considerações sobre Reconfiguração Dinâmica em FPGA - Computação por Figuras de Execução.* Workshop do PROCAD - Sistemas de Aquisição e Tratamento de Sensores envolvendo a UFPA, UFMA, UFBA e UFCG, realizado em Campina Grande (PB) de 02 a 19 de Dezembro de 2002;

- *Considerações sobre Reconfiguração Dinâmica em FPGA - Computação por Figuras de Execução.* Workshop do PROCAD - Uma Metodologia para o Desenvolvimento de Sistemas Embutidos: Gerando Competências e Suportando Propriedade Intelectual envolvendo a UFPE, USP, Unicamp e UFCG, realizado em Recife (PE) de 29 a 31 de Janeiro de 2003;

- Motivação para uma tese de doutorado *Projetos e modelos comportamentais com vistas à reconfiguração dinâmica: conjuntos de execução*[1] de Alisson Vasconcelos de Brito. Universidade Federal de Campina Grande. Início fevereiro de 2004. Foram ainda feitos dois projetos e pesquisas:

  - Estudo de reconfiguração dinâmica em modelos comportamentais com vistas à modelagem de Figuras de Execução.

[1] Título provisório.

  - Estudo de linguagens e ferramentas de projeto de modelos comportamentais com vistas as Figuras de Execução (em andamento).

- *Considerações sobre Reconfiguração Dinâmica em FPGA - Neurocomputação e Figuras de Execução.* Workshop do PROCAD - Uma Metodologia para o Desenvolvimento de Sistemas Embutidos: Gerando Competências e Suportando Propriedade Intelectual envolvendo a UFPE, USP, Unicamp e UFCG, realizado em Recife (PE) de 29 a 31 de Março de 2004.

- *Artigo aceito em congresso internacional:*

  BRUNELLI, L.; MELCHER E. U. K.; BRITO A. V. de; FREIRE R. C. S.

  A novel approach to reduce Interconnect Complexity in ANN Hardware Implementation. *IEEE International Joint Conference on Neural Networks.* Montreal, Canadá, 2005.

oOo