TRABALHO DE GRADUAÇÃO

ESTUDO E IMPLEMENTAÇÃO DE RECONFIGURAÇÃO DINÂMICA EM INSTRUMENTAÇÃO, AUTOMAÇÃO E CONTROLE

Lucas Sousa de Oliveira

Brasília, Dezembro de 2013

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

ESTUDO E IMPLEMENTAÇÃO DE RECONFIGURAÇÃO DINÂMICA EM INSTRUMENTAÇÃO, AUTOMAÇÃO E CONTROLE

Lucas Sousa de Oliveira

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Jones Yudi, ENM/UnB _____
Orientador

Prof. Carlos Humberto Llanos, ENM/UnB _____
Co-orientador

Prof. Daniel Muñoz Arboleda, FGA/UnB _____
Examinador externo

M.Sc. Janier Arias Garcia, ENM/UnB _____
Examinador interno

Dedicatória

Dedico este trabalho a meus pais Marconi e Marta Oliveira, a minha namorada Ananda Medeiros e a todos os meus professores.

Lucas Sousa de Oliveira

RESUMO

Este trabalho caracteriza um estudo realizado em reconfiguração dinâmica parcial, mais especificamente no que tange a autorreconfiguração. Nele, estudou-se reconfiguração dinâmica com o auxílio do computador, o funcionamento e uso de memória *Block Ram*, Flash e DDR3, a implementação de várias possibilidades de arquiteturas e finalmente a autorreconfiguração através de um microprocessador embarcado. A maioria dos experimentos desenvolvidos foram bem sucedidos, apresentando uma boa literatura para se construir aplicações que se utilizam destas tecnologias.

ABSTRACT

This work describes a study made about partial dynamic reconfiguration, specially regarding self-reconfiguration. The topics studied include dynamic reconfiguration through a computer, the operation and usage of memories such as Block Ram, Flash and DDR3, the implementation of various possible architectures, and finally the self-reconfiguration itself through the use of an embedded microprocessor. Most of the experiments performed were successful, providing a good reference to build applications that use this sort of technology.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

A	Área	[m^2]
Cp	Calor especifico a pressão constante	[kJ/kg.K]
h	Entalpia especifica	[kJ/kg]
$\dot{m}$	Vazão mássica	[kg/s]
T	Temperatura	[°C]
U	Coeficiente global de transferência de calor	[W/m^2.K]

Símbolos Gregos

α	Difusividade térmica	[m^2/s]
Δ	Variação entre duas grandezas similares	
ρ	Densidade	[m^3/kg]

Grupos Adimensionais

Nu	Número de Nusselt
Re	Número de Reynolds

Subscritos

amb	ambiente
ext	externo
in	entrada
ex	saída

Sobrescritos

$\cdot$	Variação temporal
$-$	Valor médio

Siglas

ABNT	Associação Brasileira de Normas Técnicas
FPGA	*Field-Programmable Gate Array*
SPI	*Serial Peripheral Interface*

Parte I

Introdução

Capítulo 1

Introdução Histórica

Este capítulo contextualiza o tema apresentado, apresentando a motivação histórica para o desenvolvimento do mesmo.

O mundo atual é controlado quase que completamente por sistemas digitais. As informações obtidas pelos sensores são digitalizadas antes de serem tratadas. Tal processo de digitalização é importante, visto que elimina os ruídos intrínsecos ao processamento analógico (CHEN, 2004).

O primeiro computador de computação genérica surgiu por volta da década de 40. Sua invenção iniciou a terceira revolução industrial, conhecida como revolução da informação ou revolução técnico-científica-informacional (PATTERSON; HENNESSY, 2005). Os computadores dessa época liam e executavam instruções de forma linear, em um modelo conhecido como sequencial ou temporal.

Nos anos que se seguiram, a substituição das válvulas por transistores de silício ajudaram a reduzir o tamanho dos computadores de metros a centímetros quadrados. Tal mudança permitiu um aumento na popularidade destes dispositivos para o uso pessoal, efeito que impulsionou a indústria de produção de processadores (HENNESSY; PATTERSON, 2011). As empresas da época começaram então a guerra de miniaturizações de transistores, marcada pelo célebre artigo de Gordon E. Moore, cofundador da Intel, que dizia que o número de transistores dentro de um processador duplicaria aproximadamente a cada 2 anos (MOORE, 1965). A partir de 1970, a lei foi adaptada para a duplicação a cada 18 meses.

Com a integração de mais componentes dentro do processador, conjuntos de instruções cada vez mais complexas foram desenvolvidas. Estas intruções surgiram para acelerar a computação de funções de níveis mais altos. A integração também reduziu a potência dissipada por transistor, permitindo que as frequências de operação dos computadores fosse aumentada (HENNESSY; PATTERSON, 2011).

Com o aumento da complexidade das instruções, passou-se a adotar duas nomenclaturas diferentes para processadores: *Reduced Instruction Set Computer* (RISC) e *Complex Instruction Set Computer* (CISC) (FEDELI; POLLONI; PERES, 2003). A arquitetura RISC possui um conjunto pequeno e muito otimizado de funções, comandos exclusivos para acesso a memória (arquitetura *load/store*) e uma média de uma instrução completada por ciclo, quando desconsidera-se as instruções de acesso a memória. A arquitetura CISC possui várias funções para tarefas mais específicas, que por vezes demandam vários ciclos de relógio, e funções que realizam operações com informações lendo e/ou salvando direto na/para a memória. A

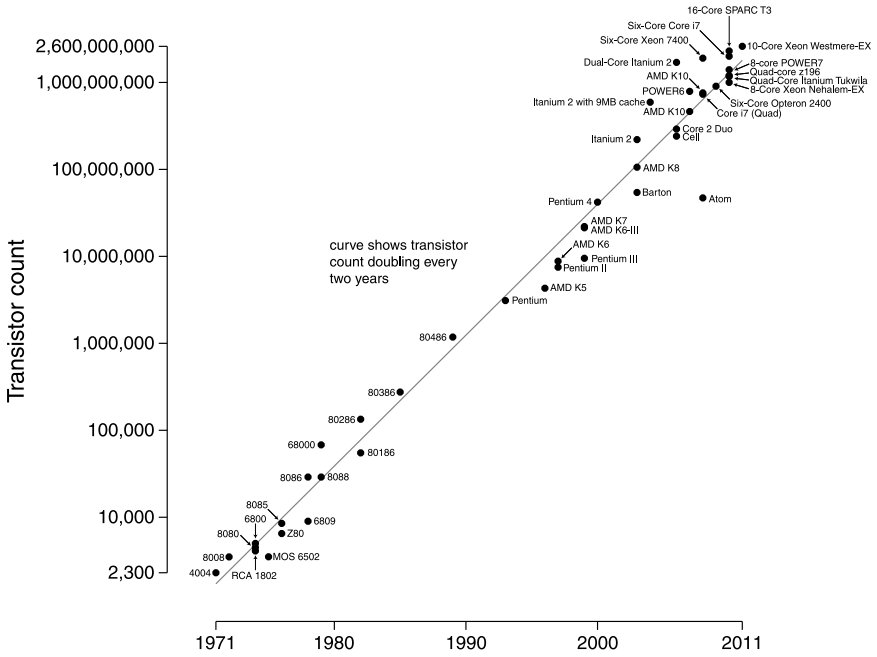


Figura 1.1: Visualização da Lei de Moore. Eixos em escala logarítmica. Gráficos extraidos do Wikipedia sob licença *Creative Commons*.

arquitetura RISC, que possui em seu portfolio dispositivos como ARM, IBM PowerPC, Sun SPARK e MIPS, dentre outros, é muito mais utilizada nos dias de hoje. Até empresas como a Intel, que ficaram populares com seus processadores CISC, tem se curvado a arquitetura RISC devido a seu uso mais eficiente de potência. Eles vem utilizando uma arquitetura conhecida como núcleo de RISC (*RISC core*), onde as instruções são recebidas em formato CISC e decodificadas para uma arquitetura interna RISC.

Por volta dos anos 2000, a potência dissipada em cada transistor, proporcional a frequência de operação, havia atingido o limite suportado pelo microprocessador. Por causa disso, o crescimento desenfreado da frequência teve que ser repensado. Começou-se então o desenvolvimento de microprocessadores *multicore*, que aumentam a vazão de instruções (*throughtput*) sem modificar o tempo de resposta, que corresponde ao tempo de processamento médio de uma instrução. Em meados de 2006, todas as grandes companhias já possuiam produtos com esta arquitetura (HENNESSY; PATTERSON, 2011).

Os microprocessadores com vários núcleos (*multicore*) abriram espaço para a chegada de processadores com muitos núcleos (*manycore*). Estes microprocessadores são projetados para placas gráficas e, apesar de possuirem centenas de núcleos, estes núcleos são simplificados (VAJDA, 2011). Em geral, eles são capazes de realizar apenas algumas poucas operações, mas abrem caminho para paradigmas de programação que transformem a computação concorrente em computação paralela (HAREL; FELDMAN, 2004).

Mesmo trabalhando com um ou vários núcleos de processamento, o modelo de computação atual ainda é dito temporal ou sequencial uma vez que blocos de instruções são executados em seu devido instante de tempo de forma sequencial, conceito destacado pela atomicidade estudada em programação paralela (WILLIAMS, 2012).

Do ponto de vista da programação, os primeiros computadores apresentavam programas que não podiam ser alterados. Parte desta limitação era justificada pela programação utilizando-se cartões, mas nas

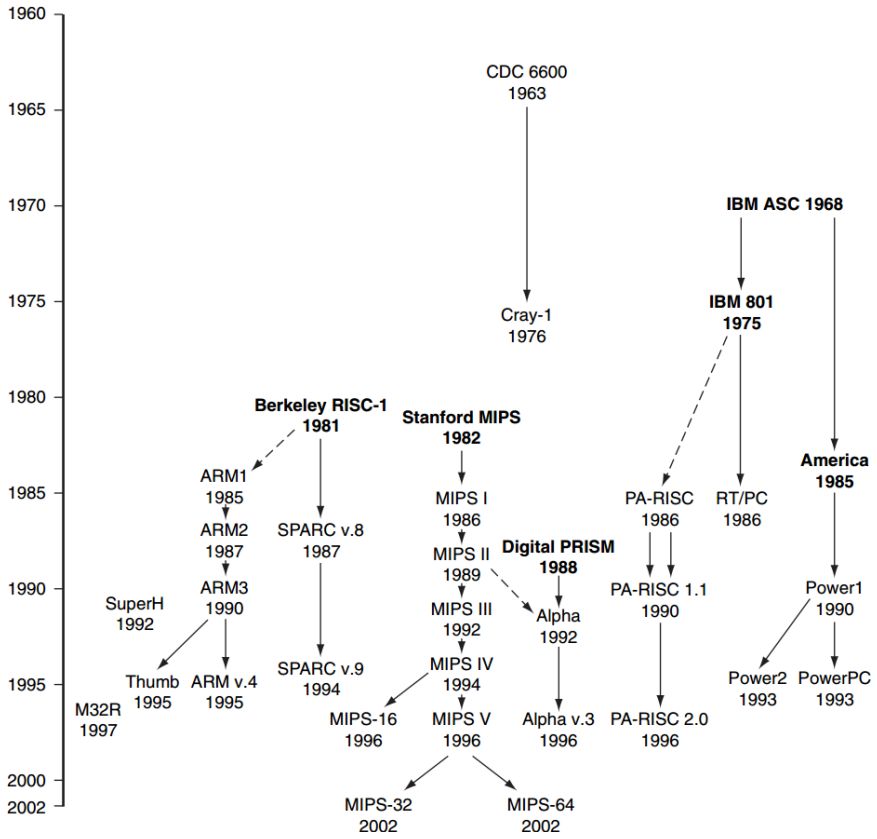


Figura 1.2: Linha do tempo das arquiteturas RISC, extraido de (HENNESSY; PATTERSON, 2011). Em negrito estão as iniciativas de pesquisa, em contraste às comerciais.

primeiras gerações de computadores com memórias eletrônicas o mesmo sistema foi utilizado.

A arquitetura de von Neumann, utilizada na primeira geração de computadores eletrônicos, era constituída de uma única unidade de memória, uma unidade de processamento e um canal de comunicação. Esta arquitetura possui tanto uma vantagem tremenda, a capacidade de modificação de programas em tempo de execução, quanto uma falha crucial, conhecida por gargalo de von Neumann. A vantagem aparece uma vez que, como não há distinção entre memória de programa e dados, uma instrução pode sobrescrever um endereço de memória marcado como programa. O problema diz respeito às restrições impostas pelo canal de comunicação, que permitia que apenas uma palavra, seja de programa ou de dados, fosse mandada para a unidade de processamento e de volta (BACKUS, 1978). Este problema se agrava a medida que o processador fica mais rápido que a memória, uma vez que o tempo de espera, em ciclos de relógios, para a obtenção da informação aumenta. Para solucionar o problema do gargalo de von Neumann, a arquitetura Harvard foi proposta.

A arquitetura Harvard original propunha que a memória de programa e a memória de dados fossem fisicamente separadas e possuissem cada uma seu próprio canal de comunicação com o processador. Essa modificação acelera a execução de certos programas, visto que programa e dados podem ser carregados das suas respectivas memórias simultaneamente. Uma pequena alteração na arquitetura Harvard, conhecida de arquitetura Harvard modificada, permitia que mais de um canal de comunicação ligasse a uma memória tanto de programa quanto de dados. Essas informações eram divididas em memórias temporárias (*cache*) específicas para o programa e para dados, formando assim uma arquitetura Harvard original. Essa modi-

ficação combina os benefícios da arquitetura de von Neumann, ou seja, a modificação de programas em tempo de execução, e da arquitetura Harvard original, ou seja, o tempo de acesso reduzido.

Atualmente, nossos modernos computadores multiprocessados utilizam a arquitetura Harvard modificada com diversos níveis de memória *cache* (HENNESSY; PATTERSON, 2011), sejam eles dedicados ou compartilhados entre os vários processadores. A sua capacidade de processamento atinge níveis extraordinários, ultrapassando 20 GFlops em computadores comuns (MAXXPI, 2013) e 54 PFlops em supercomputadores (TOP500, 2013). Apesar disso, a arquitetura Harvard original ainda é muito usada em microcontroladores e processadores digitais de sinal (*Digital Signal Processors* ou DSPs).

1.1 Computação Reconfigurável

A computação reconfigurável foi proposta por volta de 1960 por Gerald Estrin para resolver problemas que não podiam ser resolvidos pela computação da época (ESTRIN, 2002). Estrin propôs um microprocessador composto de uma parte fixa e uma parte variável, onde a parte variável seria usada para programar funcionamentos específicos para serem usados em determinados períodos de tempo. A idéia de Estrin foi deixada de lado à medida que os microprocessadores e *Application-Specific Integrated Circuits* (ASICs) se mostraram aptos a resolver os problemas da época. Por volta da década de 1990, porém, o primeiro microprocessador híbrido comercial foi desenvolvido (ESTRIN, 2002), trazendo novamente esta tecnologia à tona.

A tecnologia inventada por Estrin, também conhecido como estrutura *Fixed Plus Variable* (F+V), trouxe à tona um novo paradigma de processamento de dados (HARTENSTEIN, 2001). O motivo para tal é o fato de que a interação entre as unidades de processamento e os dados mudou completamente. O que antes se conhecia por modelo temporal de computação foi deixado de lado para, nesta nova arquitetura, se tornar um modelo espacial. Em outras palavras, os dados não eram direcionados um a um para uma unidade central de processamento, mas processados continuamente em um sistema distribuído no espaço (VASSILIADIS; SOUDRIS, 2007). Tal sistema distribuído é composto de células lógicas e suas conexões, ambas reprogramáveis, ajudando a se alcançar uma eficiência similar a presente em ASICs e flexivel como a computação genérica.

Ao contrário da estrutura F+V proposta por Estrin, a maioria dos sistemas reconfiguráveis atuais possuem apenas a parte reconfigurável. Apesar de sistemas reconfiguráveis de alta performance possuirem componentes fixos como processadores e unidades de processamento gráficos (GPUs) (EL-GHAZAWI et al., 2008), a sua ausência reduz o custo de projeto e a flexibilidade do projeto final.

Os sistemas reconfiguráveis atuais utilizam de três meios principais de programação: *Static Random-Access Memory* (SRAM), *Antifuse* e memórias não-voláteis. Usando SRAM, o resultado da síntese é armazenado nas células desta memória e controlam o estado dos transistores das células lógicas. No caso de células compostas de tabelas de busca (*look-up tables* ou LUTs), a SRAM armazena os dados dessas células. Outra segunda tecnologia de programação, o *antifuse*, faz uso de uma conexão com impedância variável, onde através do uso de altas voltagens pode-se modificar a resistência de uma via. Esse processo de programação é irreversível. As memórias não voláteis, como EPROM, EEPROM e FLASH, usam transistores especiais com uma ponte flutuante. Quando a ponte possui carga, o transistor pode ser controlado

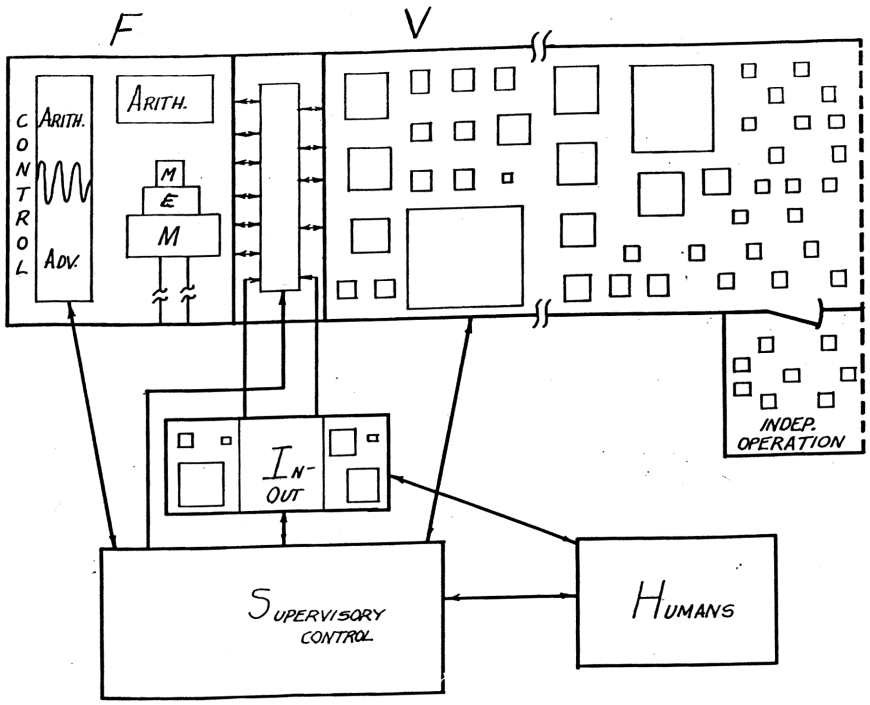


Figura 1.3: Esquemático com o F+V, extraido de (ESTRIN, 2002).

pela ponte de seleção, que permanece carregada até quando desligada. Estas técnicas permitem a resistência da *antifuse* e a reprogramabilidade da SRAM, sendo apenas mais complexa e demorada para ser programada (VASSILIADIS; SOUDRIS, 2007).

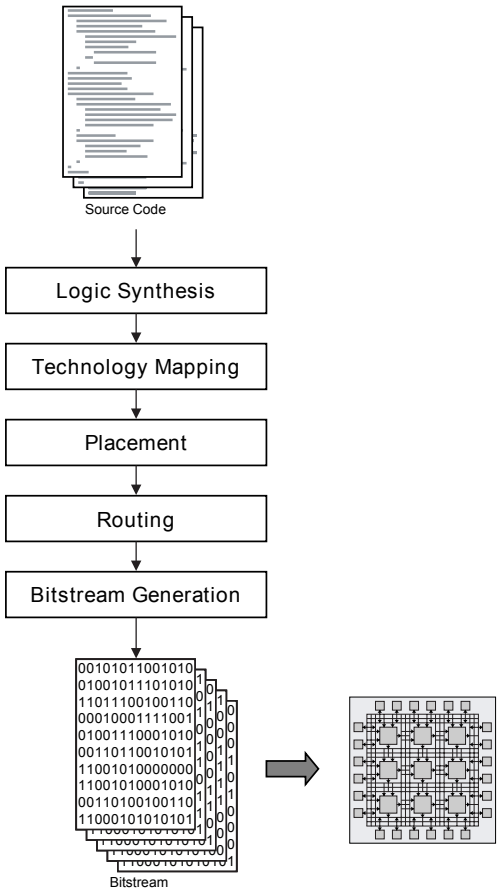
As interconexões entre células lógicas, que logicamente influenciam diretamente as células em si, podem ser de cinco tipos: ilha, linha, mar-de-portas, hierárquico e estruturas unidimensionais (VASSILIADIS; SOUDRIS, 2007). A arquitetura do tipo ilha consiste em células lógicas conectadas umas as outras através de caixas de conexões e de roteamento. Nesta arquitetura, a célula lógica está cercada por trilhas de conexões, o que explica o nome. A arquitetura do tipo linha consiste em várias linhas divididas em quantidades variadas de segmentos. As conexões são então realizadas usando-se linhas verticais através de blocos lógicos especiais. A arquitetura do tipo mar-de-portas consiste de blocos lógicos que cobrem todo o espaço do dispositivo e são conectados aos seus vizinhos diretos. Em geral este tipo de conexão é mais rápido. A arquitetura do tipo hierárquico agrupa as células lógicas em *clusters*, e agrupa estes em *clusters* de mais alto nível, formando de fato um sistema hierárquico. O último tipo de arquitetura, unidimensional, surge como uma tentativa de simplificar o roteamento complexo dos sistema bidimensionais apresentados anteriormente. Nele, restrições de alocação e roteamento são impostas para reduzir o número de possibilidades. O problema deste tipo de arquitetura é que se não houverem recursos de roteamento suficientes, o roteamento fica mais complexo que nas arquiteturas bidimensionais.

As arquiteturas reconfiguráveis podem ser classificada em três tipos segundo a granularidade. A granularidade diz respeito a quantidade de informação mínima que pode ser passada de uma célula lógica para outra. Ela separa as arquiteturas reconfiguráveis em três categorias: granularidade fina, grossa e híbrida. Nas arquiteturas com granularidade fina, como os *Field-Programmable Gate Arrays*, um único *bit* pode ser transferido de uma célula a outra, permitindo assim um maior controle sobre os dados. Nas arquite-

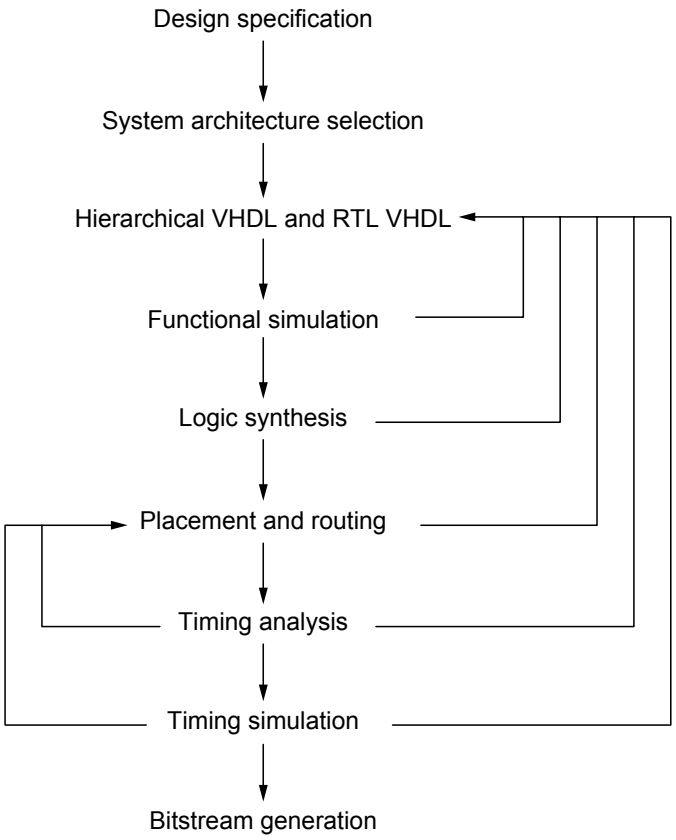
turas com granularidade grossa, os *bits* são agrupados em palavras de tamanhos fixos, reduzindo assim o espaço gasto com roteamento e melhorando a roteabilidade (HARTENSTEIN, 2001). No último tipo de arquiteturas, a híbrida, parte das conexões são grossas e partes são finas, combinando os benefícios das duas classes.

1.1.1 Compilação de *Hardware*

A compilação de *hardware* é um processo primordial no desenvolvimento de sistemas reconfiguráveis, equivalente à compilação para projetos de *software* (HAUCK; DEHON, 2007). A figura 1.4 apresenta duas imagens que apresentam este processo de compilação.



(a) Imagem ilustrativa do fluxo de compilação de um projeto de *hardware*, extraida de (HAUCK; DEHON, 2007).



(b) Imagem ilustrativa do fluxo de compilação de um projeto de *hardware* segundo visto pelo usuário, extraida de (HAUCK; DEHON, 2007).

Figura 1.4: Imagens do fluxo da compilação de *hardware*.

Descrição Assim como no *software*, se inicia com a descrição do funcionamento do sistema. Esta descrição em geral é feita utilizando-se especificações de um problema/aplicação. Ela pode ser realizada em Verilog, VDHL e diagramas de blocos, na maioria dos casos. Utiliza-se ainda a simulação funcional, como mostrado na figura 1.4b, para validar a descrição realizada.

Síntese Lógica A partir do código fonte construido, realiza-se a síntese lógica, processo que consiste em transformar a descrição comportamental ou estrutural em elementos lógicos (THOMAS; MOORBY, 1996; ASHENDEN, 2008). Assim como na compilação de *software*, existe uma fase de pré-processamento que expande macros, inclui arquivos e realiza a verificação léxica e sintática da descrição. Diversos algoritmos de otimização também são aplicados de forma a reduzir as equações lógicas, otimizando seu espaço no dispositivo e performance.

Colocação (*Placement*) e Roteamento (*Routing*) A colocação, que aqui também abrange a etapa de *floorplanning*, consiste em identificar onde a lógica deverá ser posicionada para satisfazer os requisitos de tempo, potência e performance, segundo as limitações do dispositivo-alvo. Ela recebe informações da lógica do projeto e dos recursos lógicos do dispositivo e aplica algoritmos de otimização para alcançar todos os objetivos e pré-requisitos impostos.

A etapa de roteamento, em conjunto com a colocação, tenta conectar os elementos necessários de acordo com as limitações do FPGA. Para projetos grandes, com muitos elementos lógicos, pode ser muito dificil, tornando o processo lento, ou até impossível de se realizar esta etapa. Note que o roteamente interfere diretamente com questões relacionadas a tempo e performance.

Análise e Simulação de *Timing* Durante a colocação e o roteamento, várias informações de tempo do projeto são calculados e associados ao projeto. Após estas etapas, uma análise estática de temporização e simulações extremamente precisas podem ser realizadas, removendo todas as dúvidas quanto ao projeto realizado e as descrições impostas.

Geração do Arquivo Binário (*Bitstream Generation*) A etapa de geração do arquivo binário corresponde a transformação das informações geradas na síntese, na colocação e no roteamento para bits de programação da FPGA. Estes bits popularão, durante a fase de programação, as LUTs e RAMs da placa, definindo seu funcionamento.

1.1.2 *Benchmark*

A avaliação de desempenho de sistemas reconfiguráveis não pode mais ser dada com base em quantidade de operações em ponto flutuante (FLOPS) como é feita em computadores e similares (PATTERSON; HENNESSY, 2005; CULLINAN et al., 2012). A melhor forma de se comparar dispositivos é através de do uso de uma mesma aplicação sintetizada com as ferramentas específicas de cada empresa. Neste caso, pórem, a comparação também leva em consideração o desempenho das ferramentas, especialmente em sua capacidade de otimizar as funções implementadas. Outra forma comum de se comparar desempenho é específico através de uma análise de aplicações específicas. Um exemplo para uma aplicação de filtragem de imagens é a quantidade de quadros ela consegue processar em um determinado periodo de tempo.

1.1.3 Dispositivos

As formas mais comuns de dispositivos reconfiguráveis são o *Programmable Array Logic* (PAL), o *Complex Programmable Logic Device* (CPLD), o *Field-Programmable Gate Array* e o *Reconfigurable Datapath Array* (rDPA). Cada um possui suas vantagens e desvantagens segundo a forma de implementação, comentadas a seguir.

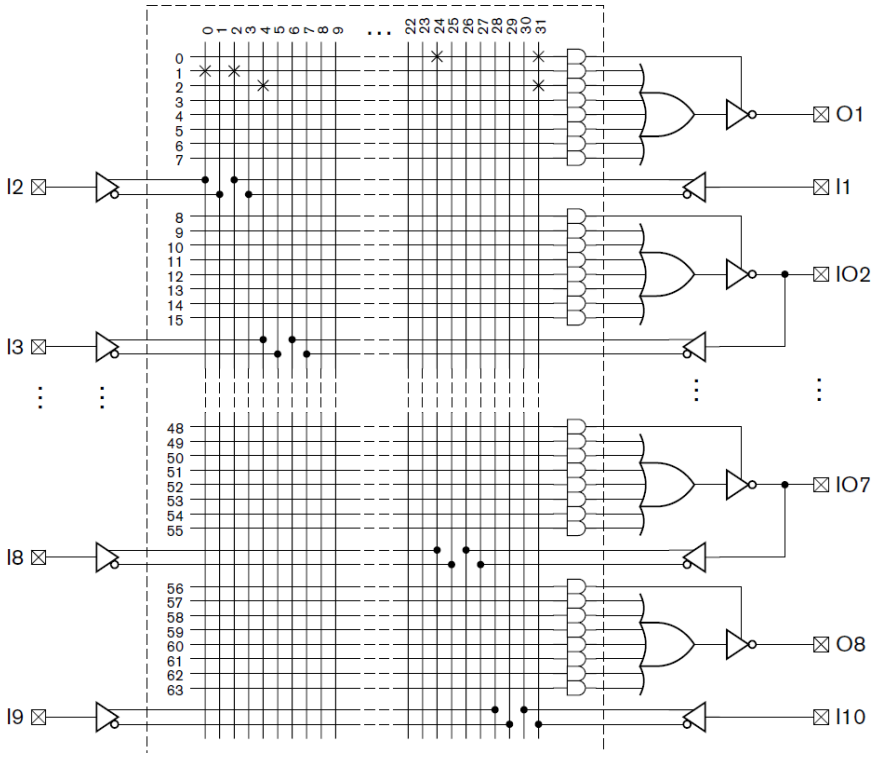


Figura 1.5: Circuito interno de um dispositivo do tipo PAL, extraido de (ASHENDEN, 2008).

Programmable Array Logic Os *Programmable Array Logics* (PALs) foram os primeiros dispositivos programáveis, desenvolvidos por volta de 1970. Os PALs podem ser configurados para desempenhar diversas funções lógicas com possibilidade de cascateamento. Alguns tipos especiais de PALs também contém registradores, permitindo a programação de circuitos sequenciais simples. Outra característica dos PALs que permitiam a construção de circuitos lógicas um pouco mais complexas era a realimentação, como pode ser visto na figura 1.5. Eles podem ser programados usando uma linguagem de descrição de *hardware* com informações na forma de expressões booleanas. Eles porém se tornaram obsoletos com a chegada dos *Generic Array Logics* (GALs) e *Complex Programmable Logic Devices* (CPLDs). Eles eram produzidos principalmente pela Data I/O Corporation.

Os PALs surgiram para substituir a lógica TTL, usada em granda escala na prototipagem e em dispositivos pequenos. Em geral, mesmo que para projetos com apenas alguns elementos de lógica TTL, o uso de PALs possuía um custo e confiabilidade maiores que na combinação de vários *chips* diferentes. Sua programação é feita através de conexões compostas por pequenos fusíveis, que são queimados quando a conexão não é necessária.

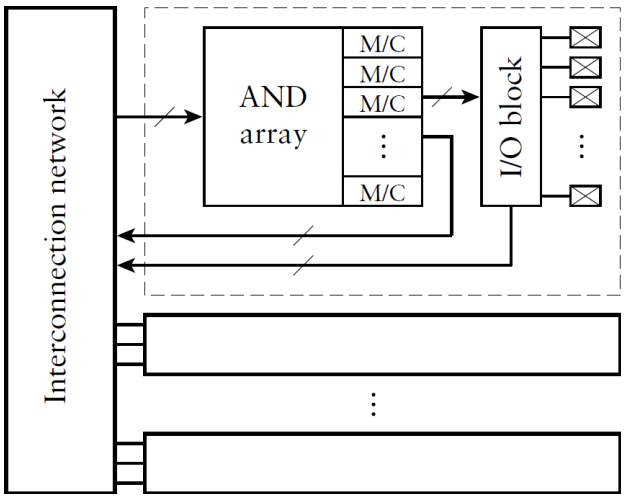


Figura 1.6: Representação de dispositivos do tipo CPLD, extraido de (ASHENDEN, 2008).

Complex Programmable Logic Device Desenvolvida depois dos PALs, os CPLDs são dispositivos similares aos PALs, mas com suporte a um número maior de blocos lógicos. Eles podem ser definidos como um conjundo de PALs conectados por uma rede programável de conexões, como tenta esquematizar a figura 1.6. Sua arquitetura é baseada no mar de portas, como mostra a figura 1.6. Detalhes de implementação variam de fabricante.

Além de conter mais recursos que os PALs, os CPLDs são diferentes na forma de programação. Ao invés de usar EEPROM, eles usam memórias RAM não-voláteis para armazenar a programação quando o sistema é desligado e células de memória SRAM para armazenar as informações de conexões e comportamento das células lógicas. Quando o dispositivo é energizado, a programação da memória RAM é passada para as células SRAM, que permitem que o sistema funcione. A memória não-volátil também possui pinos independentes acessíveis externamente, o que permitem que ele seja programado mesmo depois de soldado ao produto final permitindo sua atualização.

A maior vantagem dos CPLDs com relação a outros dispositivos é a sua não-volatilidade, tornando-o muito útil como *bootloaders* ou em aplicações que precisem rodar assim que o sistema é energizado. O mais famoso destes dispositivos, conhecido por Max, é desenvolvido pela Altera.

Field-Programmable Gate Array Formado de células programáveis consideravelmente menores que as dos CPLDs, que permitem uma maior integração, existe o *Field-Programmable Gate Array*. Como se pode notar, este dispositivo possui dentre as suas características principais a capacidade de ser programado e reprogramado em campo (*field*), isto é, sem a necessidade de processos especiais. Apesar disso, devido a complexidade dos circuitos internos destes dispositivos, simplificados na figura 1.7, eles não foram feitos para serem programados manualmente, o que ainda era possível com os CPLDs, fazendo-se necessário o uso de ferramentas de projeto auxiliado por computador (CAD) para, dado um código em linguagem de descrição de *hardware*, sintetizar, mapear, alocar e rotear o projeto automaticamente.

Desde a sua invenção, as FPGAs vem crescendo em capacidade e performance, tendo se tornado o principal componente de computação reconfigurável. A maioria das suas implementações se assemelha, em alto nível, ao circuito apresentado na figura 1.7. Eles incluem blocos lógicos que podem implementar tanto

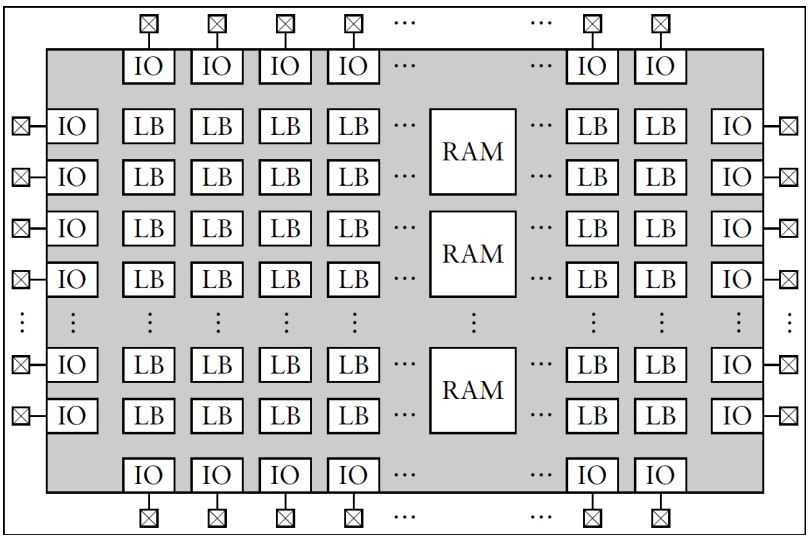


Figura 1.7: Representação de dispositivos do tipo FPGA, extraido de (ASHENDEN, 2008).

lógicas combinacionais simples quanto funções lógicas sequenciais, blocos de entrada e saída registrados ou não com possibilidade de funcionamento em diversos níveis lógicos e condições de temporização, células de memória RAM embutidas e uma rede de conexões programável. A razão para permitir que todas estas características sejam programadas é permitir que os FPGAs sejam usados nos mais diversos tipos de sistemas, que usam diferentes padrões de sinais entre *chips*. Detalhes de implementação porém variam entre fabricantes e famílias de dispositivos. Apesar disso, em sua maioria, utilizam de memórias RAM assíncronas de 1 bit conhecidas como *lookup tables* (LUTs), além de *flip-flops* e multiplexadores. Algumas FPGAs também incluem células especializadas de processamento como multiplicadores e processadores genéricos.

Existem dois tipos de FPGAs, um baseado em memória RAM e outro baseado em *antifuses*. Como foi falado na seção 1.1, a memória RAM é volátil, forçando o sistema a ser programado toda vez que energizado. Apesar disso, possui a vantagem de poder ter sua programação modificada em campo. O FPGA baseado em *antifuses* só pode ser programado uma vez, na fábrica.

	PAL	CPLD	FPGA
Custo	\$2-\$15	\$5-\$50	\$10-\$300
Blocos lógicos	8-10	32 - 128	100+
Pinos de I/O	20-24	44-160	84-256
Configuração	EEPROM	EEPROM	RAM ou OTP
Projeto	Equações Booleanas	HDL ou esquemático	HDL ou esquemático

Tabela 1.1: Tabela comparativa dos dipositivos reconfiguráveis dos tipos PAL, CPLD e FPGA.

A tabela 1.1 apresenta uma comparação ligeiramente grosseira entre os PALs, CPLDs e FPGAs.

Reconfigurable Datapath Array O *Reconfigurable Datapath Array* é um tipo de sistema reconfigurável com granularidade grossa, normalmente 32 bits (HARTENSTEIN, 2001). Apesar de relativamente mais novo que os dispositivos abaixo, ele é menos utilizado. Seus blocos lógicos, chamados de *datapath processing units* (DPUs), são um pouco mais complexos que os dispositivos de granularidade fina de forma a

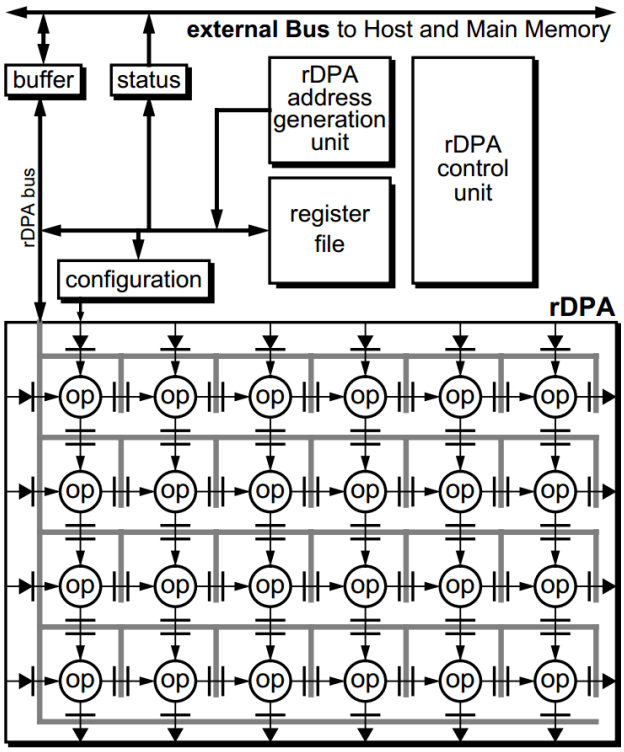


Figura 1.8: Modelo de rDPA do tipo KressArray-I, extraido de (HARTENSTEIN; KRESS, 1995).

conseguir tratar os dados maiores. Eles possuem múltiplas conexões uni e birecionais entre seus vizinhos diretos, além de trilhas verticais/horizontais completas ou segmentadas e um trilha principal mais externa que conecta todos os blocos, conforme mostra a figura 1.8. As vantagens deste tipo de sistema reconfigurável são um maior poder de processamento para uma mesma complexidade de roteamento em relação aos sistemas de granularidade mais fina além de um tempo de configuração reduzido. Em todos os outros aspectos, ele é extremamente similar a FPGAs. Sua desvantagem é o baixo controle dos bits individuais, uma vez que mesmo a descrição de *hardware* indique o uso de apenas um bit, toda uma palavra é usada.

1.1.3.1 Linguagens de Descrição de *Hardware*

Um dispositivo reconfigurável precisa de informações sobre o seu comportamento desejado para poder ser configurado. O primeiro passo desse processo é a descrição do sistema por uma linguagem de programação similar as usadas em computação geral. Dentre as linguagens mais comuns estão Verilog, VHDL e SystemC.

Verilog e VHDL descrevem o sistema através da abstração em *Register-Transfer Level* (RTL), ou seja, o comportamento dos circuitos digitais síncronos é definido em termos do seu fluxo de dados e operações realizadas. SystemC por outro lado usa o *Transaction-Level Modeling*, que descreve o comportamento do circuito através de modelos de canais de comunicações. Os módulos funcionais então realizam transações de informações entre si.

Além das linguagem já mencionadas, outra classe de linguagens de descrição de hardware é conhecida como Analog Mixed-Signal (AMS), sendo basicamente extensões das linguagens já mencionadas. Estas

extensões acrescentam a linguagem a capacidade de trabalhar com sinais analógicos. Tais linguagens tem sido bastante usadas em simulações, mas deixadas de lado na etapa de síntese pela falta de ferramentas.

Existem dois paradigmas principais para descrição de *hardware*: estrutural e comportamental (THOMAS; MOORBY, 1996). O paradigma estrutural constitui uma tentativa de se descrever um sistema através da conexão de elementos lógicos mais simples. Partindo dessas estruturas, constrói-se então elementos cada vez mais complexos. No paradigma comportamental, relações de mais alto nível, tais como somas, subtrações e operações lógicas, estão disponíveis para o uso do desenvolvedor, aproximando a descrição de *hardware* da programação de *softwares* tradicionais.

Verilog Verilog foi a primeira linguagem moderna de descrição de *hardware*. Ela foi desenvolvida com a intenção apenas de descrever e simular/validar circuitos digitais, mas nos anos seguintes a opção de síntese foi acrescentada. Padronizada pelo IEEE em 1995, o Verilog foi desenvolvido para ser similar a linguagem de programação genérica "C". Permite programação estrutural e comportamental.

VHDL VHDL foi desenvolvida logo em seguida ao Verilog, em um projeto solicitado pelo Departamento de Defesa dos Estados Unidos, como uma forma de documentar o comportamento de ASICs. Ela possui uma sintaxe similar a linguagem "Ada". A linguagem logo foi padronizada pelo IEEE, em 1987. Ela possui algumas diferenças básicas em relação ao Verilog que não serão mencionados aqui. A maior diferença prática porém é a presença de bibliotecas padronizadas pelo IEEE que disponibilizam funcionalidades muito úteis. Permite programação estrutural e comportamental.

SystemC SystemC foi desenvolvida em meados dos anos 2000, aproximadamente 15 anos depois do Verilog e VHDL, com o intuido de aproximar as linguagens de descrição de *hardware* às de programação genérica. Ela é basicamente um conjunto de classes e macros para "C++" que disponibiliza uma interface de simulação dirigidas por eventos. Essas ferramentas permitem que o projetista simule processos concorrentes, mas nos últimos tempos também foi adaptada para o desenvolvimento de sistemas reconfiguráveis. Uma vez que não foi desenvolvida com o propósito principal de descricão de hardware, possui um chamado "*overhead* sintático" com relação a Verilog e VHDL, onde mais texto tem que ser escrito para descrever um mesmo comportamento.

1.1.4 Fabricantes

As duas maiores fabricantes de FPGAs são as empresas Altera e Xilinx. A Xilinx foi a primeira fabricante de FPGAs do mundo e detém aproximadamente 51% do mercado de FPGAs, enquanto a Altera, sua maior competitora, possui aproximadamente 34% do mercado. Ambas possuem uma ampla linha de FPGAs e CPLDs. Eles serão descritos abaixo.

1.1.4.1 Altera

A Altera possui diversas linhas de FPGAs, dentre elas uma de baixo custo, chamada Cyclone, uma de médio custo, chamada Aria, e uma de alto, chamada Stratix, CPLDs, chamada Max, e uma série de ASICs, chamada *HardCopy*. Todos os seus dispositivos são programados a partir de um programa chamado Quartus, hoje na sua segunda versão. O Quartus possui ferramentas para a programação do comportamento do sistema, simulação, síntese, programação do *bitstream* do FPGA, construção de *System-on-Chips* (SoCs), IDE para programação destes SoCs e ferramentas para a verificação de projetos. Apesar da sua variada linha de dispositivos, sintetizada na tabela 1.2 e poderosa ferramenta de programação, a Altera apresenta poucas séries com possibilidade de reconfiguração parcial e autorreconfiguração.

Tipo	Família	Breve Descrição
CPLD	MAX® II	Tecnologia com numerosos blocos similares aos PALs.
FPGA	Cyclone®	Baixo custo, repleto de elementos de memória
FPGA	Arria®	Série *midrange*, com desempenho superior a Cyclone e inferior a Stratix. Pode ter *transceivers*.
FPGA	Stratix®	Alta performance, baixa potência.
FPGA	Stratix® GX	Série com *transceivers* seriais e *arrays* de alta performance escaláveis.
ASIC	HardCopy®	Série de ASICs estruturados de baixo custo e alta performance.

Tabela 1.2: Famílias de produtos da Altera, extraído de (WOODS et al., 2008) e do site da empresa.

O sistema de desenvolvimento da Altera, chamado Quartus, dá suporte a todos os dispositivos da empresa a partir da instalação de pacotes com informações dos mesmos. Apesar de muito robusto, este programa dá suporte a tecnologias mais atuais, como reconfiguração parcial ou dinâmica, através de extensões e componentes de propriedade intelectual (IP). Estas tecnologias, pórem, só estão disponíveis para alguns dispositivos mais avançados e caros, tipicamente com *transceivers*, do portfolio da companhia. A Altera possui também tecnologias de propriedade intelectual genéricas, tais como macros e componentes. O mais famoso deles é o *soft processor* Nios.

1.1.4.2 Xilinx

A Xilinx foi a responsável pela invenção do FPGA. Ela atualmente possui cinco famílias de FPGAs, as quais estão apresentadas, conforme mostrado em (WOODS et al., 2008), na tabela 1.3. Note que alguns dispositivos mais novos estão ausentes desta tabela. A Xilinx disponibiliza, como a Altera, ferramentas integradas de desenvolvimento de *hardware*, chamadas ISE e Vivado. O motivo da presença de duas ferramentas diferentes para uma mesma empresa é a recente aquisição da ferramenta Vivado, mais eficiente que a ISE. Além das funcionalidades oferecidas pela ferramenta da Altera, as ferramentas da Xilinx possuem ainda a capacidade de utilizar FPGAs como aceleradoras com uso do MatLab. Esta função permite que o projeto seja sintetizado e passado para uma FPGA real, mas que as entradas e saídas sejam controladas em

um ambiente de simulação do MatLab chamado de Simulink.

Tipo	Família	Breve Descrição
CPLD	XC9500XL	Tecnologia CPLD antiga.
CPLD	CoolRunner	CPLD de alta performance e baixo consumo.
FPGA	Virtex	Família principal da Xilinx, FPGAs de alta performance.
FPGA	Kintex	FPGA de bom desempenho e custo.
FPGA	Artix	Nova família de FPGAs de baixo custo e alto volume de vendas.
FPGA	Spartan	FPGA de baixo custo e alto volume de vendas.
SoC	Zynq	Dispositivo com ARM e diversos periféricos programáveis.

Tabela 1.3: Famílias de produtos da Xilinx, como apresentado em (WOODS et al., 2008) e no site da empresa.

A ferramenta de desenvolvimento principal da Xilinx, o ISE, é equivalente ao Quartus da Altera. Ela é responsável pela síntese dos esquemáticos e conta com programas como iMPACT e XPS, dentre outros, para formar uma solução de desenvolvimento de *hardware* completa. Recentemente, a companhia começou um processo de substituição da ferramenta ISE pelo Vivado, que possui um desempenho superior.

1.2 Objetivo

O projeto desenvolvido aqui tem por objetivo o estudo de autoreconfiguração parcial, incluindo as ferramentas e periféricos necessários para tal. Tentará-se, no decorrer dos próximos capítulos, apresentar o resultado das pesquisas bibliográficas, estudo de ferramentas, estudo de periféricos, estudo de tecnologias e análise dos resultados obtidos.

Capítulo 2

Revisão Bibliográfica

Este capítulo visa apresentar uma breve descrição sobre reconfiguração dinâmica, autorreconfiguração e as ferramentas necessárias para suas realizações.

Com a chegada de CPLDs e FPGAs, requisitos cada vez mais complexos foram sendo introduzidos ao projeto de sistemas digitais. Tais requisitos forçaram as ferramentas de síntese a suportar diferentes classes de reconfiguração. Note que reconfiguração diz respeito, como dito anteriormente, a modificação do comportamento, ou configuração, de um dispositivo reconfigurável.

2.1 Classes de Reconfiguração

2.1.1 Reconfiguração Total

A reconfiguração total, herdada da tecnologia tradicional, compreende a mudança do comportamento de todo o dispositivo reconfigurável, sem exceção de blocos lógicos. Tal reconfiguração é bastante dispendiosa, visto que maior parte das alterações realizadas são incrementais e dizem respeito à apenas uma pequena parte do dispositivo. Apesar disto, todos os FPGAs dão suporte a este tipo de reconfiguração.

2.1.2 Reconfiguração Parcial

A reconfiguração parcial, ao contrário da reconfiguração total, diz respeito à programação de apenas parte de um dispositivo reconfigurável (HAUCK; DEHON, 2007). Para tal, faz-se necessário a divisão do dispositivo nas chamadas partições, cada uma com sua configuração individual. Desta forma, mudanças feitas em uma partição não afetam as outras, acelerando o processo de síntese e programação. Outro processo que é acelerado é o de roteamento, uma vez que o particionamento introduz limitações no mapeamento das funções. Nem todos os FPGAs dão suporte a este tipo de reconfiguração, que pode ser realizado tanto de forma dinâmica (2.1.4) quando estática (2.1.3).

2.1.3 Reconfiguração Estática

O termo reconfiguração estática se refere a programação de um dispositivo reconfigurável enquanto ele não estiver executando. No caso em que alguma programação já tenha sido transferida para ele e ele a esteja executando, esta é parada para que o dispositivo seja configurado novamente. Por ser mais fácil de ser implementada e não necessita de circuitos adicionais, todos os FPGAs dão suporte a este tipo de reconfiguração.

2.1.4 Reconfiguração Dinâmica

A reconfiguração dinâmica acontece frente à necessidade de reprogramação parcial do dispositivo sem que ele pare de funcionar. As funcionalidades modificadas são interrompidas e substituidas sem afetar o funcionamento do todo. Normalmente este processo, quando não associado a autorreconfiguração, é realizado através de um circuito externo à FPGA, tal como um controlador, um microcontrolador, ou mesmo um computador, como apresentado na figura 2.1. Quase todos os FPGAs modernos dão suporte a esta tecnologia.

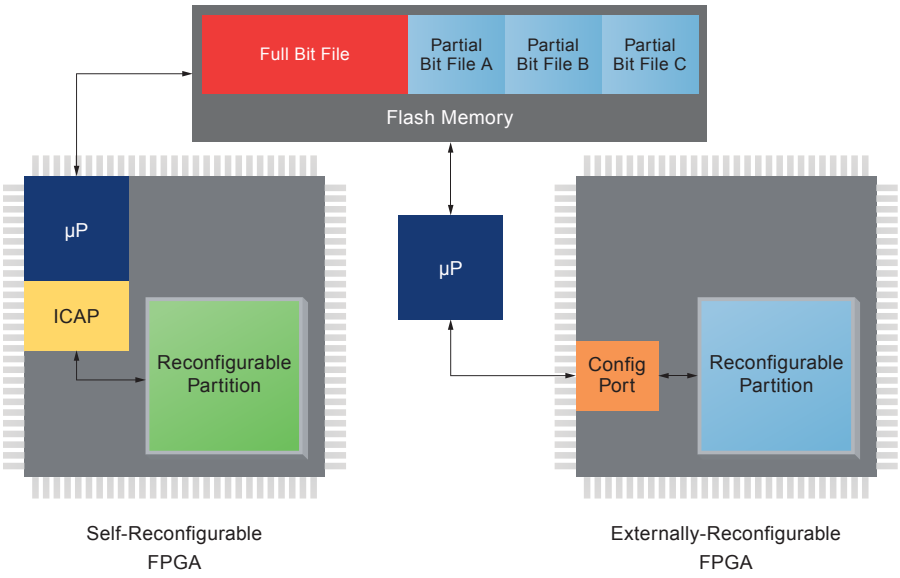


Figura 2.1: Imagem ilustrativa para diferenciação entre autorreconfiguração e reconfiguração externa, extraido de (**??**).

É implicito o uso da reconfiguração parcial com a reconfiguração dinâmica, uma vez que faz pouco sentido reconfigurar todo o FPGA enquanto ela ainda está em execução. Note que este tipo de reconfiguração em geral também necessita de uma parte permanentemente estática, para interfacear com o circuito controlador. Esta partição estática é responsável pelo menos por controlar a comunicação com o circuito controlador.

Vale lembrar que a este tipo de reconfiguração, apesar de abrir muitas possibilidades, introduz uma necessidade de preocupação com *overheads* de reconfiguração (HAUCK; DEHON, 2007). Este *overhead* é proporcional ao tamanho da partição que se deseja modificar e inversamente proporcional às velocidades

das interfaces de reconfiguração. Tal fator pode se tornar crucial na escolha entre o uso desta tecnologia ou alguma outra alternativa, possivelmente multiplexada. Note que existem outros fatores que influenciam na opção por reconfiguração dinâmica, tais como preço, capacidade e potência, dentre outros, que não serão considerados aqui.

2.1.5 Autorreconfiguração

Modalidade de reconfiguração dinâmica parcial onde a reconfiguração do dispositivo é decidida por uma lógica pertencente a ele mesmo. Normalmente usa-se um microcontrolador ou uma máquina de estados finitos para controlar a mudança de configurações. Este tecnologia é nova e ainda representa uma forte área de pesquisa. Por isso não são todos os FPGAs que dão suporte a este tipo de reconfiguração.

Para que a autorreconfiguração aconteça, os *bitstreams*, resultado da sintetização, devem ser passados para uma memória acessível ao FPGA durante a execução do mesmo. O circuito controlador identifica então um padrão que defina a necessidade de reconfiguração e transfere o *bitstream* correspondente a esta nova necessidade para a partição destino, que assim muda seu comportamento. Note que para tal, as entradas e saídas das partições tem que ser fixas, para que a mudança nas configurações das partições não danifique o FPGA em si.

2.2 Ferramentas

Diversas ferramentas foram utilizadas para a realização deste trabalho. Dentre eles pode-se citar os programas da Xilinx, interpretadores da linguagem Python, Perl e Tcl, compiladores de LaTeX e a ferramenta de controle de versão Git. Abaixo segue uma pequena descrição sobre as ferramentas mais críticas destas.

2.2.1 Xilinx ISE Design Suite

O *ISE Design Suite* é um conjunto de ferramentas da Xilinx para o desenvolvimento de projetos de *hardware*. Estes programas estão apresentados a seguir.

2.2.1.1 Xilinx ISE Design Tools

O *ISE Design Tools*, disponível para os sistemas Windows XP (32 e 64 bits), Windows 7 (32 e 64 bits), Windows Server 2008 (64 bits), Red Hat Enterprise 5 e 6 (32 e 64 bits) e SUSE Linux Enterprise 11 (32 e 64 bits) (**??**), controla todos os aspectos do fluxo de projeto (**??**). Através do *Project Navigator*, sua principal ferramenta, é possivel acessar todas as configurações e ferramentas de implementação de configurações.

Project Navigator O *Project Navigator* é a principal ferramenta do *ISE Design Tools*. Através dela é possível criar projetos, incluir arquivos de descrição de *hardware*, seja em VHDL, Verilog ou esquemáticos,

construir componentes de propriedade intelectual, impor restrições e compilá-los, dentre outras coisas. Em geral, todo o desenvolvimento começa através desta ferramenta para fins de verificação de lógica.

iMPACT O iMPACT é utilizado para se construir arquivos para a inicialização de memórias Flash e para a programação de FPGAs e suas memórias. Ele pode ser acessado por linha de comando, permitindo que outras ferramentas incorporem suas funções.

CORE Generator O *CORE Generator* é uma ferramenta muito útil, utilizada para a construção de blocos lógicos prontos de funções variadas. Em geral, seus blocos instanciam algum componente de propriedade intelectual da Xilinx.

2.2.1.2 *Embedded Development Kit*

O *Embedded Development Kit* é um conjunto de ferramentas voltados para o desenvolvimento de sistemas microprocessados embarcados em FPGA.

Xilinx Platform Studio A *Xilinx Platform Studio* permite a construção de sistemas microprocessados. Ela possui uma gama muito grande de componentes periféricos que podem acrescentados a este sistema e integrados de forma automática ou manual.

Xilinx Software Development Kit Com as informações do sistema microprocessado, é possível utilizar *Xilinx Software Development Kit* para a criação do programa que será embarcado. Esta ferramenta permite a programação do dispositivo, bem como da memória Flash, com ou sem o programa já adicionado, tornando-se uma ferramenta muito simples e útil.

2.2.1.3 PlanAhead

O PlanAhead é uma das ferramentas mais poderosas de toda a *suite*. Ele permite que projetos sejam integrados e pré-alocados em FPGAs, bem como gerenciar, modificar e verificar restrições de tempo, alocação e mapeamento, dentre outros. Ele permite ainda que módulos reconfiguráveis sejam acrescentados ao projeto, tornando a reconfiguração dinâmica um problema bem mais acessível.

2.2.1.4 Ferramentas de Linha de Comando

Além de todas as ferramentas já apresentadas, a plataforma da Xilinx ainda oferece ferramentas de linha de comando, acessadas pelo *prompt* to Windows. Através destas ferramentas é possível realizar qualquer tarefa realizavel pela ferramentas com interface gráfica e mais algumas outras. Abaixo menciona-se a ferramenta mais importante e mais utilizada deste conjunto.

Xilinx Sinthesis Tool (XST) A XST é o equivalente ao compilador para a programação convencional. Ele realiza verificações léxicas e sintáticas dos arquivos para gerar na saída algum arquivo compilado, tipicamente NGC ou BIT.

Parte II

Desenvolvimento

Capítulo 3

Introdução

Este capítulo trata da concepção dos experimentos realizados. Nele serão descritos com detalhes cada um dos experimentos, ficando a parte de análise reservada ao capítulo 9.

Devido ao caráter experimental e exploratório do objetivo proposto na seção 1.2, decidiu-se dividir o projeto em vários experimentos menores. Desta forma, além de garantir algum material mesmo que tudo dê errado, consegue-se simplificar o processo de pesquisa e desenvolvimento através dos pequenos passos e análises frequentes.

Como o objetivo final do projeto é a familiarização com as ferramentas e processos envolvidos na autoreconfiguração, decidiu-se começar estudando os elementos necessários para se realizar a reconfiguração dinâmica. O passo seguinte mais lógico é o de estudar como funciona as memórias dos sistema e de que jeito elas seriam melhor utilizadas. O último passo seria entender como funciona a autoreconfiguração em baixo nível, ou seja, como os dados devem ser entregues aos devidos componentes para que ela aconteça. Para cada um destes experimentos foi proposto um teste que validasse o completo entendimento do mesmo.

Figura 3.1: Foto ilustrativa do kit de desenvolvimento Kintex-7 KC705 extraida do site da Xilinx.

Para o desenvolvimento desse projeto, escolheu-se utilizar o kit de desenvolvimento da Xilinx® chamado Kintex-7 KC705. O único critério utilizado foi a disponibilidade dos equipamentos no início do projeto e a capacidade do dispositivo de realizar a reconfiguração parcial dinâmica. Este kit possui FPGA modelo XC7K325T-2FFG900C, leitor de cartão de memória, conector PCIe®, memória DDR3, visor de 7-segmentos e porta ethernet, dentre outros.

Escolheu-se ainda, de forma arbitrária, o uso da linguagem VHDL para a descrição de *hardware* ao invés da Verilog.

3.1 Experimentos

Os experimentos desenvolvidos nos capítulos a seguir tem como objetivo o estudo da implementação da autorreconfiguração e todos os elementos que a acompanham. Sendo assim, prova-se inicialmente que a reconfiguração é de fato dinâmica através de um experimento. Em seguida, para suportar o requisito de memória da autorreconfiguração, estuda-se as memórias e desenvolve-se um experimento para colocar este conhecimento em prática. Logo depois, espera-se entender a formação dos arquivos binários utilizados na reconfiguração e desenvolver um experimento que consiga interpretar algum cabeçalho existente. Por último, a autorrenconfiguração se tornará o objetivo principal de um experimento visando entender as interfaces utilizadas para este fim.

Capítulo 4

Experimento 1 - Reconfiguração Dinâmica

De forma a dar validade a todo o projeto, desenvolveu-se um experimento para entender o processo de desenvolvimento de sistemas reconfiguráveis dinamicamente e algumas peculiaridades do kit de desenvolvimento. Este experimento tem como objetivo a construção de um projeto que se utilize de reconfiguração dinâmica para reprogramar um FPGA.

4.1 Introdução Teórica

Antes de se começar a construir o experimento, é necessário estudar as peculiaridades da placa e deste tipo de projeto, bem como seu fluxo de desenvolvimento. Este estudo é apresetado a seguir.

4.1.1 Peculiaridades

O kit de desenvolvimento utilizada apresenta várias peculiaridades com relação aos kits comuns. A única que afeta diretamente o experimento em questão é o uso de um relógio com sinal diferencial ao invés do sinal comum, explicado a seguir. Outra peculiaridade é uma diferente estrutura de organização de arquivos, que permite um desenvolvimento mais limpo e de fácil entendimento.

4.1.1.1 Relógio Diferencial

Diferentemente das FPGAs comuns, a que está presente neste kit contém um relógio diferencial, ou seja, ele é composto por dois sinais ao invés de um. A razão para tal é a presença de circuitos sensíveis a interferências, tais como *transceivers*, que são muito menores em sinais diferenciais. O kit disponibiliza duas opções de relógio: o SYSCLK e o USER_CLOCK. O primeiro possui uma frequência fixa de oscilação de 200 MHz. O segundo possui uma frequência original de 156,250 MHz, mas pode ser programado através de uma interface I^2C para ter frequências entre 10 MHz e 810 MHz. Nota-se porém que o uso do SYSCLK é bem mais simples que o do USER_CLOCK, não sendo necessário nenhum circuito controlador/programador.

O Xilinx Design Tools disponibiliza uma ferramenta chamada *CORE Generator* para a construção de

diversos tipos de circuitos de propriedade intelectual. Dentre eles se encontra o guia *Clocking Wizard*, que pode construir elementos para transformar o sinal do relógio em um sinal simples. Através dele também é possivel utilizar as PLLs da placa para modificar a frequência deste sinal para valores entre 20 MHz e 800 MHz.

4.1.1.2 Estrutura de Pastas

A questão da organização do projeto em pastas bem específicas foi bem reforçado na literatura (Xilinx Inc., 2013a; Xilinx Inc., 2013b; Xilinx Inc., 2013c). Os manuais recomendam a estrutura de pastas apresentada abaixo.

```
Projeto/
  Source/           //codigos-fonte organizados segundo particao
  Implementation/   //contem pastas para cada config. dinamica gerada
  Synth/            //contem pastas com os arquivos .xst e .prj
  Tools/            //ferramentas para automacao da sintese
  PlanAhead/        //pasta para o projeto do PlanAhead
```

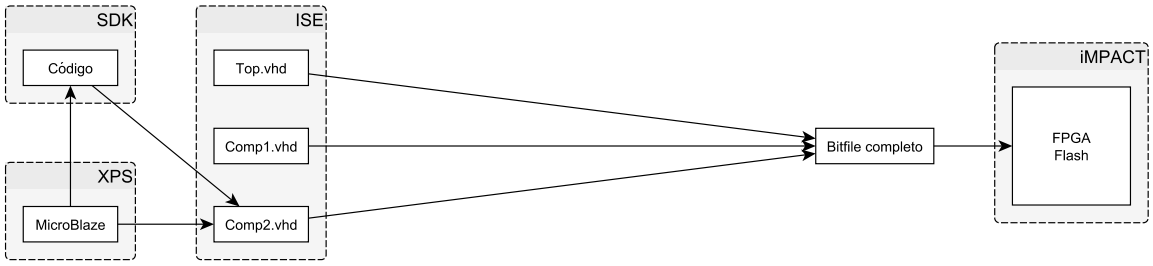
A principal razão para esta recomendação é o uso de ferramentas variadas, tais como *scripts* em TCL, o XST e o PlanAhead. Esta estrutura de pastas foi obedecida por ajudar a manter o ambiente de desenvolvimento limpo.

4.1.2 Fluxo de Ferramentas

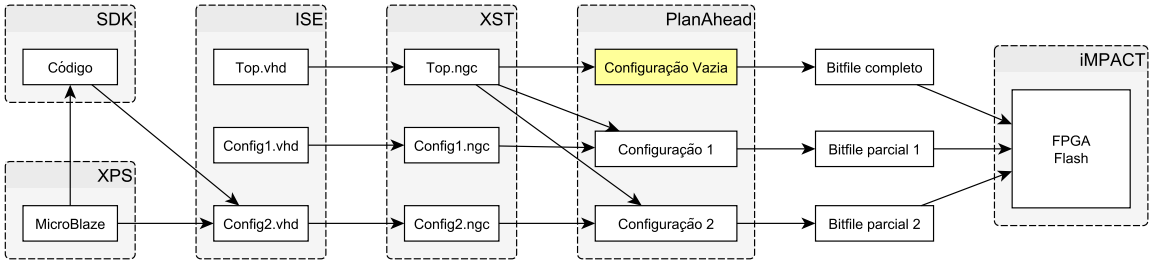
Uma das primeiras coisas que se destaca no desenvolvimento de dispositivos dinamicamente reconfiguráveis é a diferença no fluxo de ferramentas, também conhecido como *software tools flow*, em relação ao fluxo tradicional (Xilinx Inc., 2013b). Esta diferença é motivada pela necessidade de construção de diversos *bitfiles* parciais, ao contrário de outros projetos, onde objetivo final é um arquivo binário completo.

Como pode-se ver da figura 4.1a, o fluxo tradicional requer apenas o uso do programa ISE, e opcionalmente do XPS e do SDK, para a construção de um projeto de *hardware* e o iMPACT para a programação da FPGA. No fluxo para reconfiguração dinâmica, porém, mostrado na figura 4.1b, além das ferramentas do fluxo tradicional, faz-se necessário o uso da ferramenta XST para a síntese do *netlist* dos comportamentos das partições reconfiguráveis e do PlanAhead para a definição de partições e configurações. Note que estes fluxos não apresentam as únicas opções de fluxo de ferramentas, mas as que foram utilizadas neste projeto.

Note que a reconfiguração parcial pede uma síntese utilizando o método "de baixo para cima" (*bottom-up*), mas uma implementação "de cima para baixo" (*top-down*) (Xilinx Inc., 2013b), ou seja, a síntese acontece primeiro nos elementos de mais baixo nível, mas a implementação deve começar pelos de mais alto nível. Isto acontece para que os requisitos de interfaceamento dos componentes reconfiguráveis sejam encontrados antes de se determinar como incluí-los no projeto, a partir de onde se continua o processo normal de implementação. Note também que a implementação deve garantir que a lógica em comum, ou estática, seja implementada da mesma forma para as diferentes configurações (Xilinx Inc., 2013a).



(a) Foto ilustrativa do fluxo de ferramentas tradicional. Note que o uso do microcontrolador MicroBlaze é opcional, tornando os primeiros blocos, SDK e XPS, também opcionais.



(b) Foto ilustrativa do fluxo de ferramentas para a reconfiguração dinâmica. Assim como no caso tradicional, o uso do SDK e do XPS são opcionais. Note que o bloco em amarelo indica a configuração padrão, que será programada inicialmente. A escolha da configuração padrão é arbitrária.

Figura 4.1: Comparação dos fluxos de ferramentas. Note que estes fluxos não apresentam as únicas opções de fluxo de ferramentas, mas as que foram utilizadas neste projeto.

4.2 Experimento

Para se entender mais a fundo o fluxo de projeto, nada melhor que construir um projeto. Implementou-se para isto o sistema esquematizado na figura 4.2. Este sistema contém o "*Top*" para interfaceamento com a FPGA, o "*Clock_Station*" e o "*Clocks*" para tratamento do sinal de relógio, a "*Static*", que possui um lógica estática para demonstrar que a reconfiguração de uma partição não interfere com outra, e a "*Dynamic*", que possui a lógica a ser alterada dinâmicamente. A interface "*Top*" possui conexões com os LEDs e o SYSCLK do dispositivo FPGA, bem como conexões de entrada e saída com os componentes instanciados nela. O componente "*Clock_Station*" recebe a entrada de relógio diferencial e retorna 3 sinais de relógios simples, um com 1 Hz, outro com 2 Hz e o último com 5 Hz.

4.2.1 Comportamento

O projeto de um sistema parcialmente reconfigurável pode ser visto como vários projetos completos com partes em comum. Seguindo essa lógica, dois projetos com comportamentos diferentes foram construídos usando como base a figura 4.2. O comportamento individual de cada módulo ou componente será descrito a seguir. Este passo está ilustrado no fluxo de ferramentas da figura 4.1b como ISE, visto que esta ferramenta da Xilinx é a mais utilizada para projetos comuns.

O componente "*Static*" possui 3 entradas para relógios, um pulsando a 1 Hz, outro a 2 Hz e o teceiro a 5 Hz, e 3 saída paras LEDs. Ele simplesmente liga as entradas (relógios) às saídas (LEDs), criando assim uma forma visual de comprovar que a reconfiguração do componente "*Dynamic*" acontecerá dinamicamente.

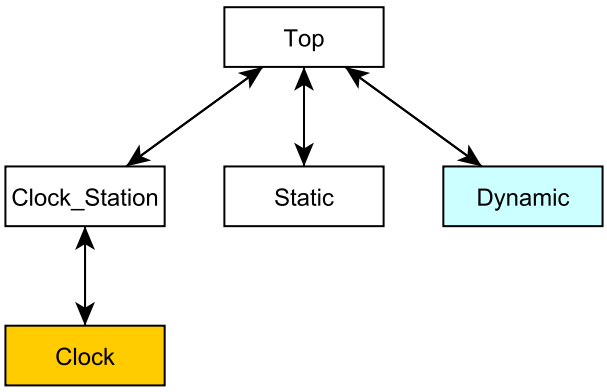


Figura 4.2: Foto ilustrativa do sistema desenvolvido para o teste de validação do experimento 1. Ele é composto por uma parte estática e uma parte dinâmica. O elementos em branco são estáticos, os em azul são dinâmicos e o em amarelo corresponde a um componente gerado automaticamente com o através das ferramentas da Xilinx.

O componente "*Dynamic*" possui dois comportamentos distintos. O primeiro deles é o de um simples contador crescente de 4 bits. O segundo é uma máquina de estados que alterna os 4 bits de saída entre "1100"e "0011"a cada pulso de relógio. Este componente possui uma entrada para um relógio pulsando a 1 Hz, que acionará as transições, e uma palavra de 4 bits de saída. A frequência de operação deste componente foi escolhida para, junto com as frequências do "*Static*", permitir a visualação de que "*Static*" não para de funcionar quando "*Dynamic*" está sendo reconfigurado.

O componente "*Clocks*" recebe os sinais diferenciais de relógio e o transformam em um sinal comum. O bloco lógico utilizado para isso foi construído usando ferramentas presentes no ISE. Uma vez que a ferramenta permitia a construção de um relógio com divisor de frequência, a frequência do relógio da placa, que nesse caso é de 200 MHz, foi reduzida para 20 MHz.

O módulo "Top" instancia os componentes descritos acima e faz a interface dos mesmos com a FPGA. O componente dinâmico precisa de uma declaração de protótipo para ser instanciado corretamente. Utilizou-se o código abaixo para esta finalidade.

```
component dynamic
    port ( clk  : in  std_logic;
           leds : out std_logic_vector (3 downto 0));
end component;
```

O módulo "Top" possui também um divisor de frequência para reduzir a frequência devolvida por "Clocks" para 1 e 2 Hz.

4.2.2 Síntese

Com o comportamento do projeto definido, o próximo passo segundo o fluxo de ferramentas é a síntese, identificada no fluxo de ferramentas da figura 4.1b como XST. Este passo é necessário uma vez que o próximo passo, referente ao PlanAhead, não aceita como entrada códigos-fonte, mas arquivos conhecidos como *netlists*. Os códigos-fonte precisam passar por uma etapa de síntese separada para poderem ser importados no PlanAhead.

O comando XST (*Xilinx Synthesis Tool*), responsável por realizar a síntese, recebe tipicamente um *script* contendo o endereço dos códigos-fonte, o nome do arquivo de saída, o tipo do arquivo de saida, o modelo da FPGA utilizada e uma indicação do código-fonte principal. O comando para iniciar o processo é o seguinte.

```
xst.exe -ifn Top.xst
```

O arquivo "Top.xst" contém os seguintes comandos.

```
run
-ifn Top.prj
-ofn Top
-ofmt NGC
-p xc7k325t-2-ffg900
-top top
```

O arquivo "Top.prj" contém os endereços dos arquivos, conforme a seguir.

```
vhdl work "../../Sources/static/top.vhd"
vhdl work "../../Sources/static/static.vhd"
vhdl work "../../Sources/static/clocks.vhd"
vhdl work "../../Sources/static/clock_station.vhd"
```

Note que estes comandos e arquivos indicados acima são para síntese dos componentes estáticos. Uma vez que não existe nenhuma restrição especial para tais componentes, eles podem ser síntetizados para um único arquivo de saída. O mesmo não pode ser dito para os elementos dinâmicos. Cada componente dinâmico precisa ser sintetizado em separado para depois ser incluído no projeto através do PlanAhead.

A síntese de componentes dinâmicos precisa ser realizada com um *script* ".xst" ligeiramente diferente. Como mostrado a seguir, faz-se necessária a inclusão do argumento "-iobuf NO", que desabilita a inserção de componentes de Entrada/Saída (Xilinx Inc., 2013b; Xilinx Inc., 2013d).

```
run
-ifn DynFSM.prj
-ofn DynFSM
-ofmt NGC
-p xc7k325t-2-ffg900
-top dynamic
-iobuf NO
```

Note que o arquivo "DynFSM.prj" contém informações sobre o código-fonte do componente dinâmico, como mostrado a seguir.

```
vhdl work "../../Sources/dynamic_fsm/dynamic.vhd"
```

Existe também a possibilidade de construção de um *script* para a síntese automática de todos os arquivos. Utilizou-se aqui uma adaptação do arquivo em linguagem TCL usado pela Xilinx em seus manuais e exemplos (Xilinx Inc., 2013a; Xilinx Inc., 2013b; Xilinx Inc., 2013c). A única função deste *script* é a construção dinâmica dos comandos com base em listas de arquivos pré-informados.

4.2.3 PlanAhead

Com os arquivos síntetisados, pode-se começar a etapa referente ao PlanAhead. Nela, importa-se os arquivos da etapa anterior, cria-se a partição reconfigurável, mapea-se esta partição no dispositivo, cria-se configurações alternativas, promove-se tais configurações e gera-se os *bitfiles* para a programação do dispositivo. Note que é preciso uma licença do ISE que permita o uso do PlanAhead e de reconfiguração parcial para a realização desta etapa.

O primeiro passo necessário no PlanAhead é a criação do projeto. Para isso, após a abertura do programa, clica-se no ícone superior esquerdo mostrado na figura 4.3, onde se lê "Create New Project". Na janela que aparece, indica-se o nome do projeto e seu caminho, lembrando que foi criada uma pasta anteriormente especificamente para conter este projeto. Indica-se também que o projeto é do tipo "*Post-synthesis Project*" e que deseja-se habilitar a reconfiguração parcial, indica-se quais *netlists* compõe o comportamento estático do sistema e qual destes corresponde ao arquivo principal ("*Top*"), qual é o arquivo de restrições (*constrains*) e qual é o modelo da FPGA.

Figura 4.3: Imagem do PlanAhead logo que aberto.

Após a criação do projeto, carregam-se os arquivos sintetizados abrindo "*Open Synthesized Design*", presente no painel "*Flow Navigator*" sob a opção "*Netlist Analysis*". Duas janelas de avisos aparecem, informando que existe uma partição não implementada e aviso sobre alguns pinos. Estes avisos podem ser desconsiderados.

Pode-se agora definir a partição que armazenará o componente dinâmico. Isso é feito através do painel "*Netlist*", selecionando-se a opção "*Set Partition*" do menu que aparece ao se clicar em "dynamic_i" com o botão direito. Na janela que aparece, seleciona-se a opção referente à partição reconfigurável, nomea-se

o módulo reconfigurável de acordo com o componente que será carregado e indica-se o arquivo (NGC) que corresponde ao arquivo sintetizado do componente reconfigurável. Não é necessário informar restrições, visto que o componente, seguindo recomendações, não acessa os pinos de entrada e saída diretamente. Note ainda que é recomendado que o primeiro módulo a ser incluído seja o mais complexo e sucetível a falhas, permitindo que erros e falhas na definição da região a seguir sejam identificados mais cedo.

Precisa-se definir também uma região para a partição recém criada. Isto pode ser feito pelo mesmo painel "*Netlist*", selecionando-se a opção "*Set Pblock Size*" do menu que aparece ao se clicar em "dynamic_i" com o botão direito. Nesse momento, precisa-se selecionar na aba "*Device*" uma região do dispositivo que contenha uma quantidade de recursos maior que a requerida pelo projeto. Note que o processo de escolha dessa região não é bem definido, o que abre espaço para diversos erros de alocação. Para testar se a região foi bem alocada, abre-se a aba "*Design Runs*" do painel inferior, clica-se com o botão direito na única configuração disponível no momento e seleciona-se "*Run Launch*". Este processo pode demorar. Em tentativas subsequentes, seleciona-se a opção de recomeçar todo o processo do zero, evitando de se utilizar arquivos gerados em execução anteriores. Uma região já testada e comprovada para este experimento é a que contém as células (*slice*) de X80Y244 a X139Y205. Ela pode ser selecionada na aba "*Device*", após se ampliar o mapa de recursos até que os nomes dos elementos lógicos, em cinza, fiquem visíveis. Outra possibilidade, um pouco mais determinística, é descrita na seção 4.2.5.3.

Ao final do "*Design Run*" aparece uma janela que pergunta o que fazer em seguida. Deve-se selecionar a opçao "*Promote Partitions*" para exportar o resultado do mapeamento e roteamento da parte estática e informações da partição reconfigurável. Estas informações serão utilizadas nos passos seguintes para construir configurações alternativas compatíveis com a primeira. Na janela que se abre, deve-se marcar as configurações implementadas.

Após promover a primeira configuração, adiciona-se duas novas possibilidades de módulos reconfiguráveis para esta partição: um com comportamento definido e outro vazia. Para isso, clica-se em "*Synthesized Design*" novamente e seleciona-se a opção "*Add reconfigurable module*" do menu que aparece ao se clicar em "dynamic_i" no painel "*Netlist*" com o botão direito. O processo é o mesmo da definição da partição, sendo a única mudança na seleção do arquivo sintetizado e do nome do módulo. O módulo vazio pode ser criado usando esta mesma opção, mas selecionando-se a opção que indica a inclusão de uma *black box* sem o uso de uma *netlist*.

Com as possibilidades de módulos reconfiguráveis definidas, pode-se construir várias configurações. Isso é feito através do clique com botão direito na "*Design Runs*" do painel inferior e selecionando-se a opção "*Create Runs...*". A janela que abre possui um painel chamado "*Create Implementation Runs*". Nesse painel, na coluna "*Partition Action*", define-se, na coluna "*Module Variant*" da nova janela que aparece, o módulo desejado. Voltando para a primeira janela, clica-se em "*More*" para adicionar a última configuração desejada. Note que, quando definindo o "*Partition Action*", a linha referente a "*Static Logic*" deve possuir "*Import*" na coluna "*Action*", indicando que a parte estática não será implementada, mas sim importada de implementações anteriores.

Em seguida, uma janela aparece perguntando o que fazer com estas novas configurações. Deve-se selecionar a opção "*Launch runs on local host*" para iniciar suas implementações. Este processo é demorado, mas mais rápido que o da primeira configuração. Note que os avisos que aparecerem podem ser ignorados.

Também é necessário promover estas novas configurações. No menu de quando se clica com o botão direito sobre as configurações já existentes do painel "*Configurations*", seleciona-se "*Promote Configuration...*". A promoção de uma configuração é o equivalente a sua exportação (Xilinx Inc., 2013d). Promover a primeira configuração antes de implementar novas contribui para manter a parte estática, compartilhada, igual em todas as configurações, uma vez que elas não mais são sintetizadas e sim importadas.

Recomenda-se ainda fazer a verificação das configurações através do painel "*Configurations*". Clicando-se com o botão direito, encontra-se a opção "*Verify Configuration...*", que faz com que uma janela seja aberta. Selecionado-se todos os itens e clicando em "*OK*", a verificação se inicia. Nenhum erro deve ser encontrado.

O último passo necessário para a criação dos *bitfiles* é o "*Generate Bitstream*", localizado no menu a esquerda. Este passo recebe o resultado das sínteses e implementações e transforma-os em *bitfiles*. Ele precisa ser realizado com todas as configurações do "*Design Runs*" selecionados ou alguma delas não terá seus arquivos binários gerados. Após o termino deste processo, os tais *bitfiles* podem ser encontrados na pasta do projeto, dentro das pastas com nome de cada configuração que ficam dentro de da pasta "*.runs". Existem dois arquivos *.bit* dentro de cada pasta, um maior, que contém a configuração completa, e outro menor, que possui a configuração parcial.

4.2.4 iMPACT

Com os *bitfiles* em mãos, usa-se-os para programar a FPGA através da ferramenta iMPACT. A primeira coisa a se fazer após abrir o programa é permitir que o sistema crie um projeto automaticamente. Na janela que se abre, escolhe-se a opção "*Automatically connect to a cable and identify Boundary-Scan chain*" do item "*Configure devices using Boundary-Scan (JTAG)*". Quando pergunta-se se deseja-se atribuir uma nova configuração, pode-se clicar que sim e escolher um arquivo binário completo gerado na etapa anterior. Normalmente escolhe-se a configuração vazia como configuração inicial para poupar energia.

Quando a configuração for completamente transmitida e implementada, observa-se que um LED está piscando com uma frequência de 2 Hz e todos os outros (acionados) estão acesos. Isto acontece uma vez que o sitema atribui sinal ativo para os elementos desconectados.

Para realizar a reconfiguração parcial dinâmica, clica-se com o botão direito no símbolo do dispositivo que aparece no iMPACT e seleciona-se a opção "*Assign New Configuration File...*". Procura-se então pelos arquivos binários parciais localizados na pasta *PlanAhead* > *PlanAhead.runs* > "nome da configuração". Este arquivo possui "partial" em seu nome, o que o diferencia do arquivo binário completo. Note que utilizar os arquivos binários completos não gera erro, mas constitui reconfiguração total, não parcial. Após a seleção da configuração desejada, o último passo necessário é a programação, que pode ser realizada clicando-se com o botão direito no dispositivo e selecionando-se a opção "*Program*".

4.2.5 Possíveis Erros

Esta seção é destinada apenas para a solução de alguns erros encontrados durante a realização deste experimento. É muito provável que, se o experimento for desenvolvido exatamente como mostrado aqui,

nenhum destes erros aconteça.

4.2.5.1 Erros no código-fonte

Este é um dos erros mais comuns. A melhor forma de previní-lo é através da construção dos diversos comportamentos/configurações individuais utilizando o ISE. Para acelerar o processo, realiza-se apenas a síntese. Outra possibilidade é o uso do XST diretamente reduzindo ainda mais o tempo desta verificação.

4.2.5.2 Erro no carregamento do módulo reconfigurável

Este erro, do PlanAhead, em geral acontece quando o componente instanciado no "*Top*" possuem sinais diferentes do instanciado no "*Dynamic*". Para corrigí-lo, deve-se verificar os códigos-fonte, síntetizá-los novamente, limpar a pasta "PlanAhead" e recomeçar este passo (PlanAhead) novamente.

4.2.5.3 Erros na alocação de partições

Um erro bastante comum que aparece no PlanAhead é o de erro de alocação[1]. Existem duas possíveis formas de corrigí-lo: modificando-se o arquivo de restrições (UCF) ou alterando a região da partição. A primeira forma, que ajuda a garantir que todos os recursos reconfiguráveis estão incluidas na região da partição, é a inclusão de "INCLUSIVE=ROUTE" na linha que contém "INST "dynamic_i"AREA_GROUP = "pblock_dynamic_i"".

A segunda forma é simplemente mudando a posição da região da partição para a direita, para a esquerda ou sua largura, de acordo com a mensagem de erro retornada. Esta método não é determinístico e pode ser necessárias várias tentativas antes de se conseguir uma partição mapeável. Este processo pode também ser realizado através do arquivo de restrições (UCF). O código abaixo contém uma alocação de partição funcional para este experimento, podendo ser copiada sobre a definição atualmente existente no arquivo UCF.

```
INST "dynamic_i" AREA_GROUP = "pblock_dynamic_i";
AREA_GROUP "pblock_dynamic_i" RANGE=SLICE_X80Y205:SLICE_X139Y244;
AREA_GROUP "pblock_dynamic_i" RANGE=DSP48_X3Y82:DSP48_X5Y97;
AREA_GROUP "pblock_dynamic_i" RANGE=RAMB18_X3Y82:RAMB18_X5Y97;
AREA_GROUP "pblock_dynamic_i" RANGE=RAMB36_X3Y41:RAMB36_X5Y48;
```

O arquivo UCF pode ser aberto pelo painel "*Sources*" sob a árvore "*Constrains*". Em caso de mais de um arquivo UCF presente, o desejado pode ser identificado pela marcação "*(target)*" ao lado. Dois cliques são suficientes para abrí-lo.

[1] *AR# 53290: Partial Reconfiguration - 7 series device layout of tiles (CLB, DSP, BRAM, INT) and a shared clocking structure of vertical clock spines between interconnect (routing) tiles*. Disponível em <http://www.xilinx.com/support/answers/53290.htm>

4.2.5.4 Esquecer de promover a partição

A promoção da primeira configuração antes de se implementar as seguintes contribui para a implementação de configurações compatíveis. Esquecer de promover esta partição pode fazer com que erros sejam gerados na etapa de verificação das partições.

4.2.5.5 O PlanAhead pode travar enquanto implementando uma configuração

Apesar de mais raro, o PlanAhead pode travar quando implementando uma configuração. A melhor solução é o reinício do computador, visto que o programa bloqueia alguns arquivos durante a implementação e não os desbloqueia quando fechado forçadamente.

4.2.5.6 Erros na detecção da placa

Note que algum programa aberto pode interferir com a varredura realizada pelo iMPACT, fazendo-a falhar. Para prevenir este erro, deve-se fechar o XPS, o SDK e qualquer outro programa que possa se utilizar das interfaces USB. Note ainda que a placa deve estar ligada para poder ser detectada.

4.2.5.7 Erro na geração do arquivo binário

Na geração do arquivo binário, o erro BitGen:342 pode aparecer [2]. Este erro aparece devido ao uso de pinos não explicitados no arquivo UCF, fazendo com que o PlanAhead tenha que usar as configurações padrões nestes pinos. Uma forma de solucionar este problema é acrescentando a opção "-g UnconstrainedPins:Allow" no comando do BitGen.

4.3 Resultados

4.3.1 Síntese

A síntese ocorreu como esperado, gerando os arquivos sintetisados (NGC). A figura 4.4 mostra o resultado do processo de síntese utilizando o *script* Tcl.

```
[redacted]>make
Overriding with data file .\Tools\data_synth.tcl
Synthesis tool is set to xst
**** Synthesizing Reconfig Module named <DynCounter> ****
xst -ifn DynCounter.xst
**** Synthesizing Reconfig Module named <DynFSM> ****
xst -ifn DynFSM.xst
**** Synthesizing Reconfig Module named <Top> ****
xst -ifn Top.xst

**** Finished bottom-up synthesis of all RMs ****
```

Figura 4.4: Resultado da síntese.

[2] "*AR# 51813 14.2 BitGen - "ERROR:Bitgen:342 occurs after adding probes to the design in the case of 7 series devices'*", <http://www.xilinx.com/support/answers/51813.html>

4.3.2 PlanAhead

O passo do PlanAhead foi bem sucedido, tendo gerado os arquivos binários como esperado. O relatório de utilização do sistema, copiado abaixo, mostra que aproximadamente 1% dos recursos do dispositivo foi utilizado.

```
Slice Logic Utilization:
  Number of Slice Registers:                 2,734 out of 407,600    1%
    Number used as Flip Flops:               2,696
    Number used as Latches:                      3
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:               35
  Number of Slice LUTs:                      3,020 out of 203,800    1%
    Number used as logic:                    2,572 out of 203,800    1%
      Number using O6 output only:           1,962
      Number using O5 output only:             114
      Number using O5 and O6:                  496
      Number used as ROM:                        0
    Number used as Memory:                     338 out of  64,000    1%
      Number used as Dual Port RAM:            160
        Number using O6 output only:            96
        Number using O5 output only:             0
        Number using O5 and O6:                 64
      Number used as Single Port RAM:            0
      Number used as Shift Register:           178
        Number using O6 output only:           176
        Number using O5 output only:             1
        Number using O5 and O6:                  1
    Number used exclusively as route-thrus:    110
      Number with same-slice register load:    103
      Number with same-slice carry load:         6
      Number with other load:                    1
```

Note porém que a quantidade de elementos lógicos ocupados é maior devido ao roteamento. O relatório que mostra este fenônemo foi copiado abaixo.

```
Slice Logic Distribution:
  Number of occupied Slices:                 1,445 out of  50,950    2%
  Number of LUT Flip Flop pairs used:        3,851
    Number with an unused Flip Flop:         1,360 out of   3,851   35%
    Number with an unused LUT:                 831 out of   3,851   21%
    Number of fully used LUT-FF pairs:       1,660 out of   3,851   43%
    Number of slice register sites lost
      to control set restrictions:               0 out of 407,600    0%
```

A parte estática do dispositivo foi alocada como mostrado na figura 4.5.

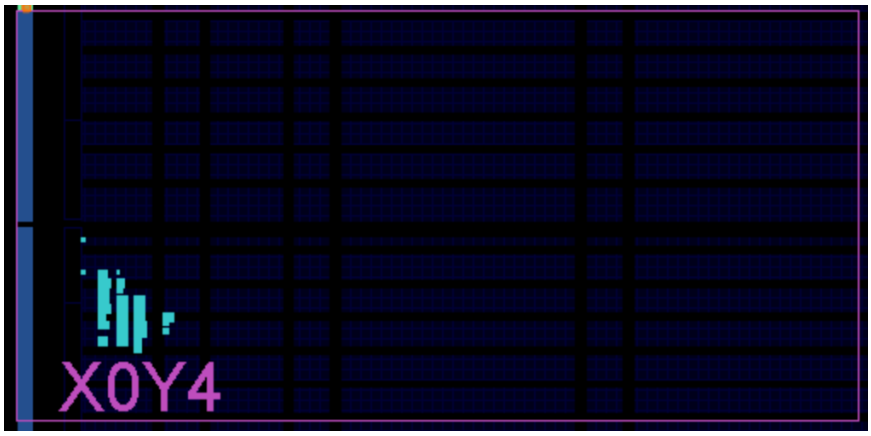


Figura 4.5: Alocação da parte estática do projeto no dispositivo. Em azul claro estão os elementos lógicos utilizados.

4.3.3 iMPACT

A etapa do iMPACT também foi bem sucedida. A programação da configuração completa levou 10 segundos e a programação da configuração parcial levou 1 segundo.

4.4 Conclusão

O processo de desenvolvimento de partições reconfiguráveis e sua programação foi explorado e compreendido com sucesso. Observou-se os pontos críticos do desenvolvimento e o fluxo mais adequado para a construção deste tipo de projeto, bem como alguns erros comuns e suas soluções.

Capítulo 5

Experimento 2 - Memórias

Seguindo o raciocínio apresentado no capítulo 3, o próximo passo natural no desenvolvimento deste projeto é o entendimento do funcionamento das memórias. Esta etapa abre caminho para que se armazene os arquivos binários de configurações parciais em uma memória embarcada, removendo a necessidade do computador para tal.

5.1 Introdução Teórica

Para o desenvolvimento deste experimento, faz-se necessário o estudo dos diferentes tipos de memórias, citando seus pontos fracos e fortes. Também é necessário o estudo do MicroBlaze e das ferramentas XPS e SDK, visto que a forma mais direta de se trabalhar com memória em FPGAs é através do uso de microcontroladores embarcados.

5.1.1 Tipos de Mémoria

No kit utilizado existem vários tipos de memórias que poderiam ser usados, cada um com suas peculiaridades (Xilinx Inc., 2013e). Este experimento foi dedicado à compreenção do funcionamento dos diversos tipos de memórias e à escolha da solução mais adequada.

5.1.1.1 Memória *Block RAM*

A memória do tipo *Block RAM* é construída usando-se os blocos de memória RAM restantes da FPGA. Esta memória consegue alcançar velocidades de leitura na ordem de várias centenas de hertz, mas possui uma capacidade de armazenamento bem reduzida, de apenas 445 blocos de 36 Kb para este kit, totalizando um máximo de aproximadamente 2 MB (**????**). Note que a configuração total da FPGA necessita de aproximadamente 11 MB, ou exatamente 91.548.896 bits, para sua programação total (**??**). Pode-se programá-la através do iMPACT, dentre outras formas [1].

[1] *Memory Initialization Methods*, escrito por Jim Wu em 31 de dezembro de 2011. Dísponível em <http://myfpgablog.blogspot.com.br/2011/12/memory-initialization-methods.html>

5.1.1.2 Memória *Distributed RAM*

A memória *Distributed RAM* é construida utilizando-se das LUTs disponíveis nas célula lógicas (**????**). Também são muito rápidas, apesar de síncronas, e também possuem pouca capacidade de armazenamento. Pode-se programá-la através do iMPACT, dentre outras formas.

5.1.1.3 Memória *Linear BPI Flash*

A memória *Linear BPI Flash* disponibiliza 128 Mb de memória não-volátil (Xilinx Inc., 2013e), acessadas em palavras de 16 bits. Apesar disso, sua velocidade de leitura máxima é de 33 MHz. Convertendo tal velocidade para a leitura de 32 bits, tem-se uma velocidade de leitura de aproximadamente 16 MHz. Pode-se programá-la através do iMPACT.

5.1.1.4 Memória *SPI Flash*

A memória SPI Flash é diferente na sua forma de acesso, que acontece através da interface SPI. Esta memória fornece 128 Mb de memória não volátil (Xilinx Inc., 2013e). Ela aceita três modos de operação, x1, x2 e x4, que corresponde a largura da palavra lida/escrita a cada pulso de relógio (N25Q128...,). A frequência de operação é de no máximo 50 MHz (**??**). Pode-se programá-la através do iMPACT.

5.1.1.5 Cartão de Memória SD

O cartão de memória SD dá acesso a uma memória não-volátil de tamanho arbitrário. Esta interface permite o uso de cartões de alto desempenho, lendo palavras de 4 bits a frequências de até 50 MHz (Xilinx Inc., 2013e). A limitação desta interface é a dificuldade de leitura e escrita devido ao sistema de arquivos inerente ao cartão. Note que também não existe a possibilidade de programação do cartão através do iMPACT, o que resolveria o problema do sistema de arquivos.

5.1.1.6 *CompactFlash* e o *System ACE*

System ACE é um sistema que permite a programação de FPGAs e memórias voláteis a partir de um cartão *CompactFlash* (**????**). Este é bem robusto e popular em séries que possuem um leitor deste tipo de cartão, o que não é o caso desta.

5.1.1.7 Memória DDR3

A memória DDR3 é uma memória volátil com 1 GB de capacidade de armazenamento e possui uma frequência de operação da ordem dos 800 MHz (DDR3..., 2010). Só pode ser programada apenas em tempo de execução, fazendo-se necessário o uso de um *bootloader*. Ela possui restrições de temporização extremamente apertadas e necessita do uso de componentes bastante escassos no FPGA.

5.1.2 Fluxo de Projeto

O procedimento para a construção de um projeto com MicroBlaze segue o fluxo descrito na figura 4.1a. A figura 5.1 apresenta o fluxo de ferramentas para este caso específico. Em outras palavras, o fluxo espera que primeiro se desenvolva o microprocessador com todos os seus periféricos para depois se desenvolver o programa que será embarcado. Tanto o programa quando o processador gerarão arquivos binários que serão carregados pelo iMPACT através do SDK.

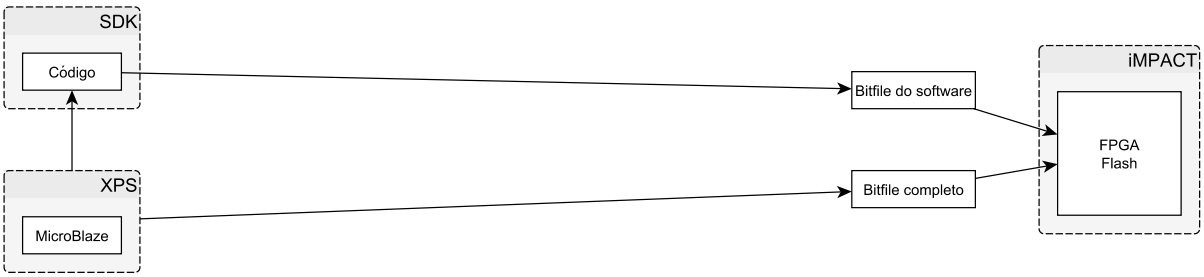


Figura 5.1: Foto ilustrativa do fluxo de ferramentas para o desenvolvimento de sistemas com MicroBlaze, extraído de (**??**).

5.1.3 MicroBlaze

O MicroBlaze é um microprocessador otimizado para implementação em FPGAs da Xilinx (**??**). Ele possui 32 registradores genéricos de 32 bits, instruções de 32 bits e endereços de 32 bits. Seu *pipeline* possui 3 ou 5 estágios e é construido em torno da arquitetura Harvard, como pode ser observado na figura 5.2. Todas as outras configurações, tais como o uso de *Big Endian* ou *Little Endian*, por exemplo, são opcionais (**??**).

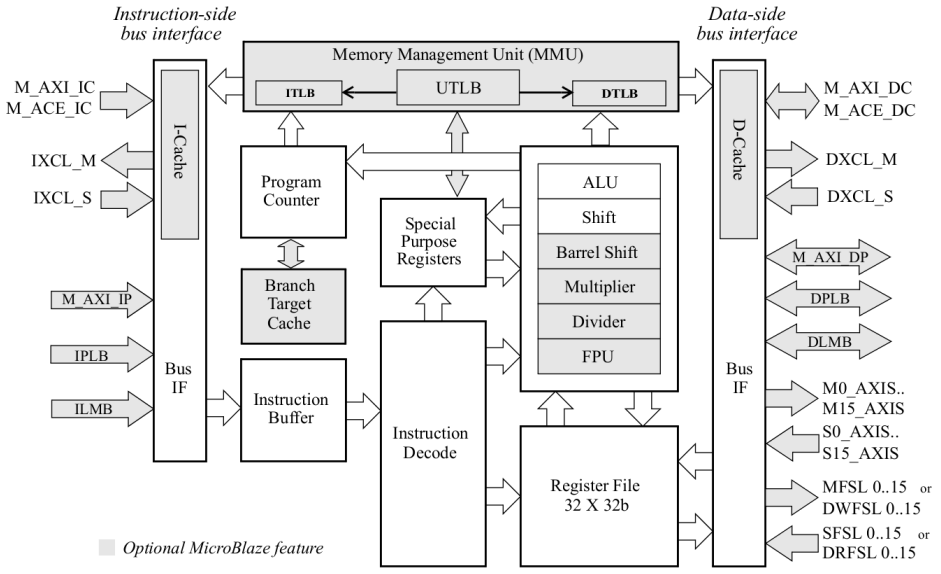


Figura 5.2: Diagrama de blocos do MicroBlaze.

O MicroBlaze permite o uso de diversas interfaces para comunicação com seus diversos periféricos, dentre elas a PLB, a LMB, a AXI e a ACE (**??**). A interface mais atual suportada, e que foi utilizada neste

experimento, é a Advanced eXtensible Interface 4 (AXI4) (**????**). A AXI4 é uma interface mapeada em memória que oferece produtividade, flexibilidade e disponibilidade. Ela possui três tipos de interfaces, a AXI4, a AXI4-Lite e a AXI4-Stream, onde as duas primeiras são compostam de 5 canais de comunicação: de leitura de endereço, de escrita de endereço, de leitura de dados, de escrita de dados e de escrita de resposta.

5.2 Experimento

Este experimento tem por objetivo o desenvolvimento de um sistema que instancie e acesse memórias capazes de armazenar as configurações parciais geradas no experimento anterior. Para isto, deve-se entender o processo de construção de um processador embarcado MicroBlaze, com periféricos para acesso às memórias, e de construção de programa embarcado. Deve-se ainda, antes de qualquer coisa, definir que tipo de memória é a mais apropriada para o objetivo traçado.

5.2.1 Escolha da Memória

Nota-se do primeiro experimento que os *bitfiles* parciais gerados possuiam 645 KB cada, totalizando 1935 KB, ou 1,89 MB. Com isso o uso das memórias que se utilizam de recursos internos da FPGA fica comprometido. Observa-se também que a interface de reprogramação dinâmica, ICAPE2, utilizada em experimentos mais a frente, opera a uma frequência de até 100 MHz e pode receber palavras de até 32 bits, ou seja, 3200 Mb/s (Xilinx Inc., 2013a). Por causa disso, todas as interfaces não-voláteis disponíveis acabariam se tornando gargalos no tempo de programação.

A única memória com tamanho e velocidade suficientemente grandes para este projeto é a memória DDR3. Apesar disto, esta memória é volátil, necessitando ser inicializada sempre que o sistema é ligado. Uma solução para este problema é a implementação de um sistema que carregue as informações de uma memória não-volátil, tanto a Linear Flash quanto a SPI Flash seriam suficientes, para a memória DDR3 em um momento inicial, e depois permita que a memória DDR3 seja acessada para a reconfiguração dinâmica, ou seja, através do uso de um *bootloader*.

Sendo assim, optou-se por utilizar a memória QSPI Flash para uso na inicialização e a memória DDR3 para uso na execução.

5.2.2 XPS

Para começar, deve-se abrir o programa e criar um novo projeto usando o *Base System Builder* (BSB), opção que corresponde ao item superior esquerdo do menu da tela inicial. Na janela que aparece, escolheu-se a placa de desenvolvimento Kintex-7 KC705 *Evaluation Platform*, *Board Revision* C, e a opção de um só processador no sistema, para simplificar o projeto. Dando prosseguimento, escolheu-se os periféricos desejados segundo a figura 5.3 e o tamanho das memórias local, de intrução e de dados, quiça, 128 KB, 8 KB e 8 KB respectivamente. Modificou-se ainda, como pode-se ver na figura 5.3, o *Baud Rate* da interface UART para 115200 bits/s e o C_SPI_MODE da QSPI_FLASH para "*Quad SPI Mode*".

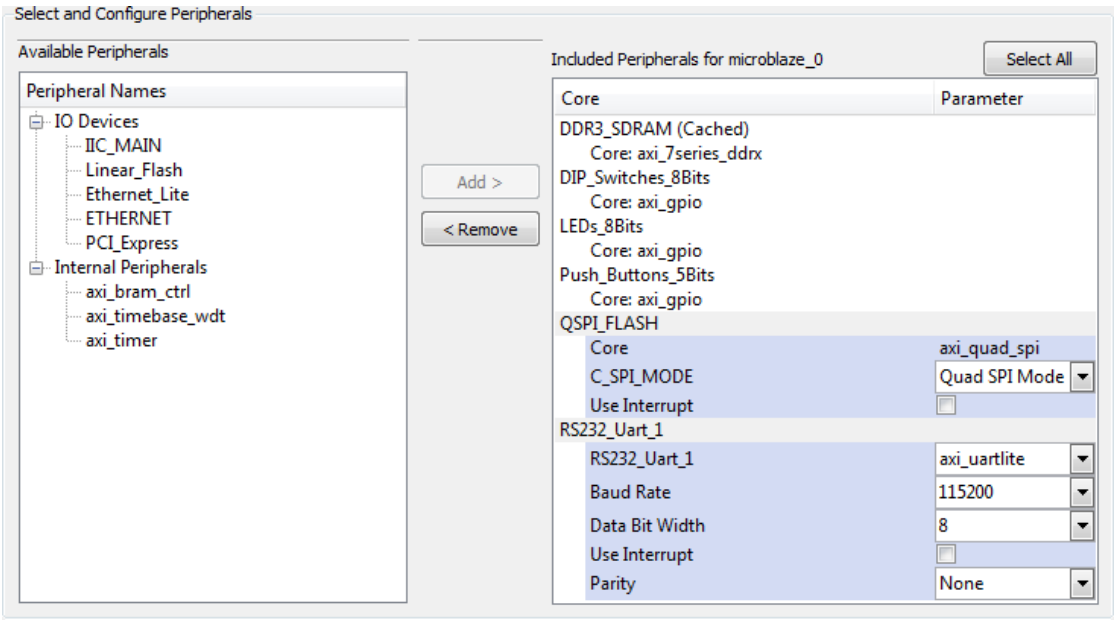


Figura 5.3: Escolha dos periféricos no BSB do XPS.

Antes de concluir o a construção do sistema, precisa-se ajustar os tamanhos das memórias e seus endereços, de forma que estes novos tamanhos possam ser corretamente acessados. Isto pode ser feito na aba *Addresses* do painel *System Assembly View*. Muda-se então o tamanho da memória SPI Flash para 128M, seu endereço base para 0x80000000, o tamanho da memória DDR3 para 1G e seu endereço base para 0xC0000000, conforme a figura 5.4. Note que o endereço-base da memória SPI Flash tem como única restrição de os 28 bits menos significativos iguais a zero e que o endereço-base da memória DDR3 tem esta mesma restrição para os 30 bits menos significativos. Estas restrições são devido aos tamanhos das memórias e o alinhamento dos espaços de dados na memória.

Instance	Base Name	Base Address	High Address	Size
microblaze_0's Address Map				
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x0000FFFF	64K
LEDs_8Bits	C_BASEADDR	0x40000000	0x4000FFFF	64K
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	64K
debug_module	C_BASEADDR	0x41400000	0x4140FFFF	64K
QSPI_FLASH	C_BASEADDR	0x80000000	0x87FFFFFF	128M
DDR3_SDRAM	C_S_AXI_BASEA...	0xC0000000	0xFFFFFFFF	1G

Figura 5.4: Aba *Addresses* do *System Assembly View* indicando os ajustes dos endereços e tamanhos das memórias.

Precisa-se ainda modificar as posições de memória cobertas pela memória *cache*. Isto é feito clicando-se duas vezes sobre "microblaze_0" na aba "*Bus Interfaces*" do painel "*System Assembly View*". Na janela que se abre, clica-se "*Next*" 3 vezes para chegar página sobre *caches*. Modifique os endereços nas duas colunas para 0xc0000000 e 0xffffffff, indicando que a memória de instruções e memória de dados podem acessar os endereços da memória DDR3. Apenas os endereços contidos neste intervalo da memória de dados podem ser escritos. Note que o sistema reserva para uso próprio os primeiros 64K endereços das memórias, que correspondem às posições com finais entre 0x0000 a 0x3fff. A tentativa de uso destes endereços comprometerá o correto funcionamento do sistema.

Algumas outras configurações também podem ser ajustadas, mas não sao estritamente necessárias para este experimento.

O projeto pode ser sintetizado através do botão "*Generate Netlist*" localizado no menu a esquerda e no menu "*Hardware*" da barra de menus. Este processo é demorado. Quando terminado, pode-se exportar o projeto através do botão "*Export Design*" para abrir o SDK com as informações deste processador. Na janela que aparecer, marque "*Include bitstream and BMM file*" e clique em "*Export & Launch SDK*". O arquivo binário contém informações da configuração da FPGA enquanto o arquivo BMM apresenta um mapeamento das unidades de memória onde o programa embarcado pode ser inserido. Após a execução desta etapa, se nenhum erro tiver acontecido, a ferramenta SDK será aberta.

5.2.3 SDK

Quando a janela do SDK aparecer, ela perguntará onde se quer colocar o *workspace*. Uma boa opção é a pasta SDK criada dentro da pasta do projeto do sistema durante sua exportação, apesar desta escolha ser completamente arbitrária. Escolhida a pasta, o ambiente de trabalho é aberto.

Começa-se o processo criando um projeto do tipo "*Board Support Package*". Este projeto compila automaticamente os drivers disponíveis para o projeto segundo as características do microcontrolador. Em seguida, pode-se criar os projetos dos *softwares* que irão embarcados. Para isto, cria-se um projeto do tipo "*Application Project*". Na janela que se abre, nomeie o projeto e selecione a "*Board Support Package*" no *dropdown* do item "*Board Support Package*". Na página seguinte, escolhe-se "*Empty Project*".

Adiciona-se arquivos ao projeto tanto arrastando os códigos para ele quanto selecionando a opção "*New/Source File*" do menu que aparece quando se clica no projeto com o botão direito. Note que o acesso a memória é feito simplesmente dereferenciando um ponteiro para a posição de memória desejada. A escrita é feita do mesmo modo. Encontra-se em anexo códigos-exemplo para o teste das memórias DDR3 e SPI Flash.

Antes de executar o programa, é interessante habilitar a opção de debugação. Isto é feito através do menu "*Run*", na opção "*Run Configurations...*". Na janela que aparece, na aba "*STDIO Connection*", habilita-se a conexão do STDIO com o console e modifica-se o "*Baud Rate*" para 115200. Clica-se então em "*Apply*" e em seguida "*Close*".

A programação do FPGA pode ser feita através do menu "*Xilinx Tools*", na opção "*Program FPGA*". Na janela que se abre, é padrão que as informações já estejam pré-preenchidas, mas caso isto não aconteça, procura-se pelos arquivos "system.bit" e "system_bd.bmm" na pasta "*implementations*" na raíz do projeto do processador. Clica-se em "*Program*" para inciar a programação.

Para se transferir o programa criado com o auxílio do SDK, seleciona-se o projeto deste programa, clica-se com o botão direito, seleciona-se o submenu "*Run As...*" e escolhe-se a opção "*1 Launch on Hardware (GDB)*". No caso dos códigos-exemplo, algumas informações são imprimidas no console caso tudo tenha ocorrido confome o esperado.

5.3 Resultados

O experimento foi bem sucedido. A seguir são apresentados alguns relatórios e amostras dos resultados.

5.3.1 XPS

O XPS, assim como o ISE, fornece relatórios que apresentam a utilização do dispositivo. Este relatório está mostrado a seguir. Note que mesmo com um sistema bem mais complexo, o grau de utilização do dispositivo não aumentou muito com relação ao experimento passado.

Slice Logic Utilization:				
Number of Slice Registers:	7,775	out of	407,600	1%
Number used as Flip Flops:	7,720			
Number used as Latches:	0			
Number used as Latch-thrus:	0			
Number used as AND/OR logics:	55			
Number of Slice LUTs:	8,916	out of	203,800	4%
Number used as logic:	7,603	out of	203,800	3%
Number using O6 output only:	5,826			
Number using O5 output only:	173			
Number using O5 and O6:	1,604			
Number used as ROM:	0			
Number used as Memory:	1,014	out of	64,000	1%
Number used as Dual Port RAM:	572			
Number using O6 output only:	120			
Number using O5 output only:	12			
Number using O5 and O6:	440			
Number used as Single Port RAM:	0			
Number used as Shift Register:	442			
Number using O6 output only:	441			
Number using O5 output only:	1			
Number using O5 and O6:	0			
Number used exclusively as route-thrus:	299			
Number with same-slice register load:	268			
Number with same-slice carry load:	30			
Number with other load:	1			
Slice Logic Distribution:				
Number of occupied Slices:	3,944	out of	50,950	7%
Number of LUT Flip Flop pairs used:	11,005			
Number with an unused Flip Flop:	3,939	out of	11,005	35%
Number with an unused LUT:	2,089	out of	11,005	18%

```
Number of fully used LUT–FF pairs:        4,977 out of  11,005   45%
Number of slice register sites lost
  to control set restrictions:                0 out of 407,600    0%
```

5.3.2 SDK

Os resultados apresentados pelo SDK se resumem a uma compilação e uma programação de dispositivos bem sucedidas.

5.4 Conclusão

Conclui-se assim o experimento para o teste de programação das memórias. Notou-se que existe muita pouca literatura no assunto, forçando o programador a fazer uso dos forúns de discussão e conhecimentos gerais de programação embarcada. Apesar disso, o objetivo do experimento, quiça conseguir ler/escrever de/em endereços das memória DDR3 e SPI Flash específicos, foi atingido com sucesso.

Capítulo 6

Experimento 3 - *Bootloader*

O *bootloader*, como mencionado no experimento anterior, é um sistema que carrega as informações de uma memória lenta, como a SPI Flash, para uma memória rápida, como a DDR3. Usou-se um microcontrolador MicroBlaze com interfaces para as memórias SPI Flash e DDR3. O experimento 2 foi dedicado a aprender a utilizar estas memórias. Neste experimento, espera-se entender como é formado o arquivo binário, permitindo assim carregá-lo e interpretá-lo enquanto o transferindo para a memória DDR3.

6.1 Introdução Teórica

Para se alcançar o objetivo proposto, faz-se necessário entender como é construido o arquivo binário e como se inicializa a memória não-volátil. O fluxo de projeto é o mesmo do experimento anterior.

6.1.1 Arquivo Binário

O arquivo binário é formado por três partes: um cabeçalho, uma palavra para sincronia e a configuração propriamente dita (**????**). A figura 6.1 apresenta o início deste arquivo, contendo as partes mais importante do cabeçalho. Os bytes selecionados correspondem ao conteúdo legível do cabeçalho.

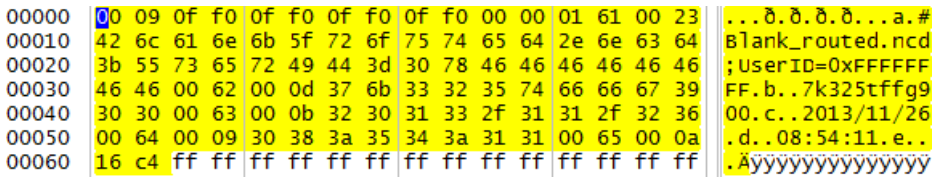


Figura 6.1: Cabeçalho do arquivo binário gerado no primeiro experimento para a configuração vazia.

O cabeçalho é formado por chaves e tamanhos, indicando os diversos campos deste. Ele contém pelo menos duas informações muito interessantes: o identificador do dispositivo alvo, que permite verificar a compatibilidade entre a configuração e o dispositivo que a está recebendo, e o tamanho da configuração, que permite que ela seja carregada de forma dinâmica sem necessidade de mais informações.

Na tabela 6.1 pode-se observar os tamanhos e campos apresentados na figura 6.1 [1].

Tamanho	Chave	Significado
2 bytes	9 (0x00 09)	Tamanho em bytes do próximo campo
9 bytes	0x0f f0 0f f0 0f f0 0f f0 00	Indica que a configuração a seguir é válida.
2 bytes	1 (0x00 01)	Tamanho em bytes do próximo campo
1 byte	"a"(0x61)	Indica que os próximos campos conterão informações sobre o projeto e sobre a configuração.
2 bytes	35 (0x00 23)	Tamanho em bytes do próximo campo
35 bytes	Blank_routed.ncd; UserID=0xFFFFFFFF	Apresenta o nome do *netlist* e o identificador do usuário. 0x00 ao final indica o final da string.
1 byte	"b"(0x62)	Indica que o próximo campo é um indentificador do dispositivo-alvo.
2 bytes	13 (0x00 0d)	Tamanho em bytes do próximo campo
13 bytes	7k325tffg900	Identificador do dispositivo-alvo. 0x00 ao final indica o final da string.
1 byte	"c"(0x63)	Indica que o próximo campo é a data de sintese da configuração.
2 bytes	11 bytes (0x00 0b)	Tamanho em bytes do próximo campo
11 bytes	2013/11/26	Data da síntese da configuração.
1 byte	"d"(0x64)	Indica que o próximo campo é a hora de síntese da configuração.
2 bytes	9 bytes (0x00 09)	Tamanho em bytes do próximo campo
9 bytes	08:54:11	Hora de síntese da configuração.
1 byte	"e"(0x65)	Indica que os próximos 8 bytes contém o tamanho da configuração.
4 bytes	661188 (0x00 0a 16 c4)	Tamanho em bytes da configuração a partir desta posição.

Tabela 6.1: Descrição do cabeçalho dos arquivos binários.

Existe ainda no cabeçalho 32 bytes de espaçamento preenchidos por com “0xff” e bytes para autodetecção de largura de banda (**????**). Estes bytes (“0x00 00 00 bb 11 22 00 44”) são usados no modo de configuração paralela para detectar automaticamente a largura de banda do arquivo de configuração. O modo serial ignora todos os bits anteriores a palavra de sincronia (**??**). Estes bits são então usados apenas para pré-processamento do arquivo binário.

A palavra de sincronia (“0xaa 99 ff 66”), encontrada a seguida, serve para indicar o inicio da configuração proprimamente dita e para alinhar o fluxo de dados nos registradores internos.

6.1.2 Inicialização da Memória SPI Flash

A memória SPI Flash, assim como todas as outras, precisa de um procedimento especial para poder ser inicializada com informações arbitrárias (**??**). Em geral, as únicas informações que podem ser gravadas nas memórias não-voláteis são configurações para o FPGA e programas para algum MicroBlaze embarcado (**??**). Este processo é conhecido como programação indireta da memória Flash (**??**).

[1] “*FAQ: Tell me about the format of the .BIT files please*”, <http://www.fpga-faq.com/FAQ_Pages/0026_Tell_me_about_bit_files.htm>

6.1.2.1 Compilação

Para a realização da programação indireta, o arquivo binário precisa ser compilado de forma a gerar um padrão de bits compatível. Este procedimento é realizado na etapa de construção do processador. Quando não especificado, o arquivo binário gerado utiliza a interface QSPI x1, que possui banda de transferência de 1 bit por leitura/escrita, e relógio de frequência 3 MHz. Com estas configurações, a programação do sistema demora apenas 1 minutos e 30 segundos (**??**), mas pode ser ainda mais reduzido.

No PlanAhead, as configurações de compilação podem ser modificadas através do arquivo "bitgen.ut" localizado na pasta "etc" do projeto. Através da opção "-g SPI_buswidth:X", onde X pode ser 1, 2 ou 4, pode-se alterar a interface utilizada neste tipo de programação (**????**), sendo a x4 a mais eficiente. Pode-se ainda forçar a opção "-g ConfigRate_en:Y", onde Y pode ser 3, 6, 9, 12, 16, 22, 26, 33, 40, 50 ou 66, para se utilizar de relógios de variadas frequências e obter assim o modo de configuração mais adequado para a memória em questão (**????**Xilinx Inc., 2013e). Existe também a opção "-g SPI_Fall_Edge:Yes", que permite melhora margens de tempo e pode aumentar as taxas de leitura para configurações (**????**). Uma opção alternativa é utilizar um relógio externo através da opção "-g ExtMasterCclk_en:Z", onde Z pode ser "Disable", "div-8", "div-4", "div-2" e "div-1".

6.1.2.2 Arquivo de Memória PROM

Após compilado o projeto, seu arquivo binário e qualquer outra informação a ser programada na memória Flash precisa ser adicionada a um arquivo do tipo MCS. Este processo é necessário para que o iMPACT consiga carregar e programar a memória Flash de forma correta. Pode-se construir este arquivo de memória PROM através do próprio iMPACT, processo que será descrito mais a frente.

6.2 Experimento

O teste deste experimento compreende a construção de um sistema microprocessador que carregue os arquivos binários da memória SPI Flash para a memória DDR3. Este experimento contribuirá para a compreenção do processo de inicialização de memórias não-voláteis e do tratamento do cabeçalho dos arquivos binários. Usará-se os programas XPS, SDK, que fazem parte do Embedded Development Kit (EDK), data2mem e iMPACT.

6.2.1 XPS

Utilizou-se o mesmo microcontrolador MicroBlaze construido no experimento passado, não sendo necessária nenhuma modificação. A interface SPI utilizada aqui foi a padrão, x1, por motivos de simplificação do projeto. O uso de uma interface x4 diminuiria o tempo de programação em 4 vezes, o que não é fator crítico para este experimento, mas acarretaria na necessidade de recompilar todos os projetos desenvolvidos até agora.

6.2.2 SDK

Logo após a construção do microcontrolador, recomenda-se construir o projeto do programa embarcado. O procedimento para o SDK nesse caso é bem semelhante ao dos experimentos anteriores, mudando-se apenas os arquivos importados e a geração do *linker script*. Os arquivos a serem importados podem ser encontrados no CD de anexos.

O projeto do bootloader consiste apenas de uma máquina de estados para ler o cabeçalho do arquivo binário e interfaces com os drivers controladores das memórias DDR3 e SPI Flash e da interface de comunicação através da porta UART.

Desta vez, tem-se como objetivo fazer com que o sistema se carregue e entre em funcionamento de forma autônoma. Para isto, precisa-se, depois da inclusão dos devidos arquivos, presentes no anexo, gerar o *linker script*. Este arquivo descreve como o arquivo binário do *bootloader* deve ser armazenado na memória interna do FPGA para execução. Este *script* pode ser gerado clicando-se com o botão direito sobre o projeto do prgrama embarcado e selecionando-se a opção "*Generate Linker Script...*" ou selecionando-se o projeto, abrindo-se o menu "*Xilinx Tools*" e selecionando-se a opção do mesmo nome. Para o escopo deste experimento, as configurações apresentadas na janela que se abre são suficientes, bastando clicar em "*Generate*". A criação deste *script* permite agora que se utilize o programa "data2mem" para construir um arquivo binário de configuração com um programa embarcado pré-programado.

6.2.3 data2mem

A ferramenta data2mem funciona através da linha de comando, mas normalmente é acionada através de programas com interfaces gráficas, tais como o ISE, o XPS, o SDK e o iMPACT (**??**). Sua função principal é o de mapear blocos contíguos de dados entre múltiplas *Block RAMs* distribuídas pelo FPGA mantendo um acesso lógico contínuo, i.e., dados em endereços de memória adjacentes podem estar em blocos completamente diferentes. No caso em questão, ela mapeará o *bootloader* desenvolvido nas *Block RAMs* de forma que no momento da programação, o MicroBlaze embarcado já possua um programa carregado.

Um comando típico do data2mem é construído com as opções "-bm" para indicar o caminho para o arquivo do tipo "*Block RAM Memory MAP*" (BMM), "-bt" para indicar o caminho para o arquivo binário do tipo BIT, "-bd" para indicar arquivos de programas do tipo ELF ou MEM, permitindo a inclusão de um identificador para associá-lo a algum dispositivo implementado, e "-o b" para indicar o caminho do arquivo binário (BIT) de saída. Um exemplo de uso do comando é apresentado abaixo.

```
data2mem \
-bm SDK/XPS_QSPI_Final_hw_platform/system_bd.bmm \
-bt SDK/XPS_QSPI_Final_hw_platform/system.bit \
-bd SDK/bootloader/Release/bootloader.elf tag microblaze_0 \
-o b SDK/XPS_QSPI_Final_hw_platform/download.bit
```

O comando acima também pode, como foi dito anteriormente, ser executado através do SDK. Para isto, deve-se acessar o menu "*Xilinx Tools*" e selecionar a opção "*Program Flash*". Uma janela aparece, onde deve-se selecionar os arquivos BIT e BMM gerados pelo XPS na compilação do sistema e o arquivo ELF gerado na compilação do programa embarcado. Este procedimento pode ser feito com a placa de desen-

volvimento conectada ou não, gerando um erro que pode ser desprezado quando ela estiver desconectada. O arquivo gerado pode ser encontrado na pasta “SDK/*_hw_platform” sob o nome “download.bit”.

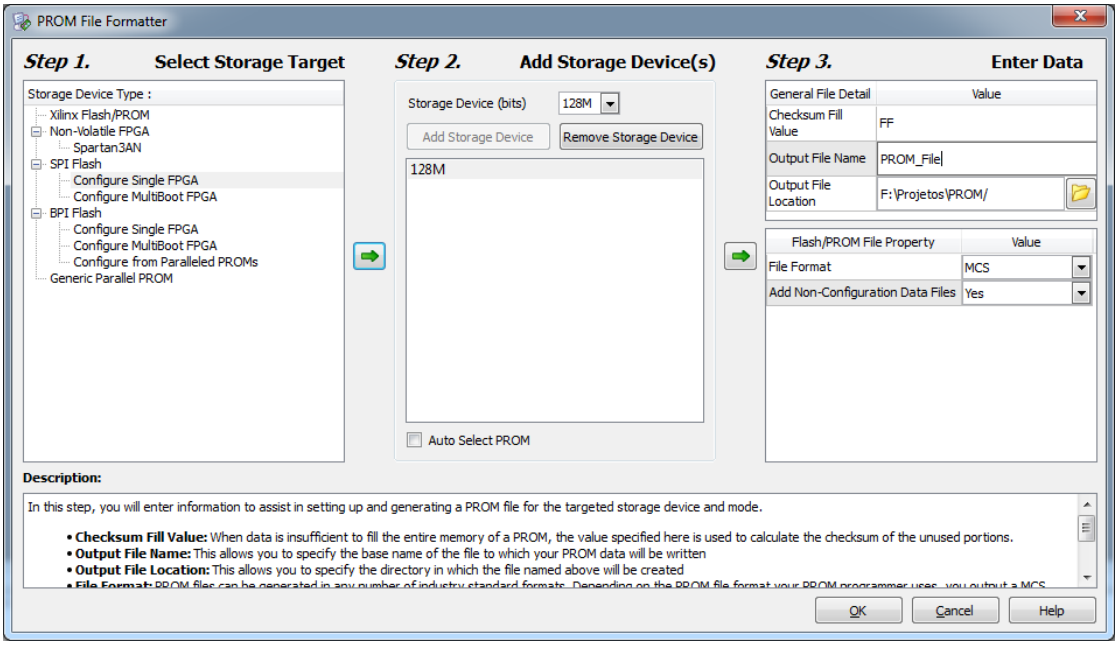


Figura 6.2: Janela para criação de um arquivo de memória PROM com as configurações devidamente ajustadas.

6.2.4 Programação Indireta da Memória Flash

Como todos os arquivos binários prontos, pode-se começar o processo de programação indireta da memória Flash. Este processo se inicia através da criação de um arquivo de memória PROM através da ferramenta iMPACT. Vale salientar apenas que faz-se necessário que todos os elementos (configurações totais e parciais) sejam previamente compilados para uma mesma interface SPI, seja ela x1, x2 ou x4. O uso de interfaces diferentes pode gerar erros durante a programação da memória Flash.

No momento da criação do novo projeto do iMPACT, seleciona-se a opção “*Prepare a PROM File*”. Na janela seguinte, seleciona-se “*SPI Flash/Configure Single FPGA*” no primeiro painel e “128M” no segundo e modifica-se “*Add Non-Configuration Data Files*” para “Yes”, conforme mostrado na figura 6.2.

Em seguida, adicionam-se os arquivos binários que se deseja programar na memória Flash. O primeiro arquivo a se adicionar é o de configuração total. Este arquivo é carregado durante o procedimento de inicio do FPGA. Apenas um arquivo deste tipo precisa ser carregado neste experimento, apesar de ser possível realizar um projeto com diversas revisões de configurações ou múltiplas possibilidades de configurações de *boot* (**????**). Para rejeitar a adicão de outras configurações, deve-se clicar em “*No*” na mensagem de título “*Add Device*”.

A mensagem seguinte, “*Add Data File*”, faz referência a adição de arquivos de dados neste projeto. Clicando-se em “*Yes*”, uma janela aparece com informações de endereçamento. Pode-se aceitar os valores iniciais. Estes arquivos podem ter qualquer conteúdo, mas o programa espera arquivos gerados pelo SDK, sendo necessário mudar a configuração de arquivos apresentados para “*All Files (*.*)*”. Após adicionar-

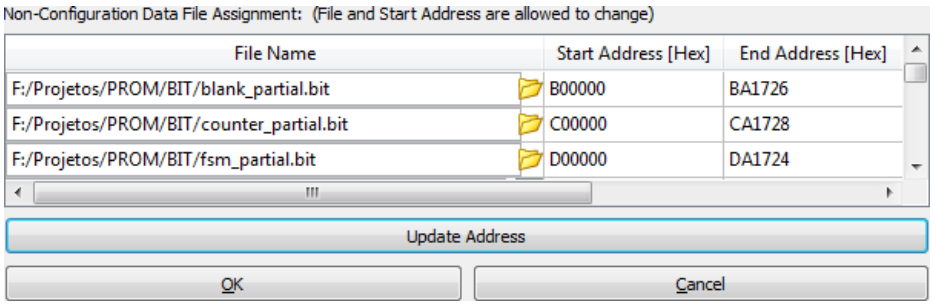


Figura 6.3: Janela para criação de um arquivo de memória PROM com as configurações devidamente ajustadas.

se todos os arquivos, deve-se clicar em "*No*" na janela de inclusão de novos arquivos de dados. Ao se fazer isto, uma janela para indicação dos endereços é mostrada. Recomenda-se mudar os endereços de início ("*Start Address*") para valores arredondados, como 0xB00000, 0xC00000 e 0xD00000, obedecendo os endereços das revisões, de forma a facilitar o trabalho de programação do sistema embarcado. O botão "*Update Address*" deve ser clicado para ajustar os endereços de fim ("*End Address*") antes de se prosseguir, obtendo-se algo similar a figura 6.3. Note que é possível incluir também um programa para o MicroBlaze na memória Flash, a ser carregado em tempo de execução, para controlar a mudança de configurações.

O último passo é gerar o arquivo, o que pode ser feito através do menu "*Operations*" ou do painel "*iMPACT Processes*", selecionando-se a opção "*Generate File...*". Este processo é rapido e resulta em um arquivo MCS gerado na pasta destino definida no passo da figura 6.2.

6.3 Resultados

Logo após a programação e após cada ciclo de energia (*power cycle*), o programa embarcado envia dados da sua execução através da porta UART para o computador, gerando a saída mostrada na figura 6.4. Pode-se observar que o programa foi executado perfeitamente.

Note que a programação da FPGA através do iMPACT demorou sempre entre 800 e 1200 segundos, ou seja, entre 13 e 20 minutos, independente da largura de banda utilizada da interface utilizada. O motivo para esta demora não é informado, mas suspeita-se que seja devido a velocidade de transmissão da porta JTAG ou devido a velocidade de escrita na memória SPI Flash.

6.4 Conclusão

O experimento realizado funcionou como esperado, tendo carregado as informações do computador para a memória Flash e, em tempo de execução, da memória Flash para a memória DDR3. Ainda foi possível interpretar o cabeçalho do arquivo binário, extraindo dele informações importantes para o correto carregamento das configurações. O processo de programação da memória Flash através do computador é bem demorado, chegando a levar 20 minutos, mas durante o *boot* do sistema, ele é bem rápido, demorando apenas alguns poucos segundos.

É relativamente complicado trabalhar com os periféricos e *drivers* que os acompanham devido a documentação escassa e a grande disperção das informações. Este experimento só pode ser realizado devido a uma imensa pesquisa e compilação de guias de usuário, relatórios de aplicações, *datasheets*, exemplos de projetos para diversos tipos de kits de desenvolvimento e comentários em fóruns de discussão.

```
Carregando configuracoes...
Processando cabecalho de Blank em QSPI@0B00000
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00
        - T: 02 byte(s), C: 0x00 01  (1 byte(s))
        - T: 01 byte(s), C: 0x61 ('a')
        - T: 02 byte(s), C: 0x00 23  (35 byte(s))
        - T: 35 byte(s), C: Blank_routed.ncd;UserID=0xFFFFFFFF
        - T: 01 byte(s), C: 0x62 ('b')
        - T: 02 byte(s), C: 0x00 0D  (13 byte(s))
        - T: 13 byte(s), C: 7k325tffg900
        - T: 01 byte(s), C: 0x63 ('c')
        - T: 02 byte(s), C: 0x00 0B  (11 byte(s))
        - T: 11 byte(s), C: 2013/11/26
        - T: 01 byte(s), C: 0x64 ('d')
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 08:54:11
        - T: 01 byte(s), C: 0x65 ('e')
        - T: 04 byte(s), C: 00 0A 16 C4 (661188 bytes)
Tamanho do cabecalho: 98 bytes
Carregando a configuracao Blank (QSPI@B00000 -> DDR3@C0104000)... Terminado!
Processando cabecalho de Counter em QSPI@0C00000
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00
        - T: 02 byte(s), C: 0x00 01  (1 byte(s))
        - T: 01 byte(s), C: 0x61 ('a')
        - T: 02 byte(s), C: 0x00 25  (37 byte(s))
        - T: 37 byte(s), C: Counter_routed.ncd;UserID=0xFFFFFFFF
        - T: 01 byte(s), C: 0x62 ('b')
        - T: 02 byte(s), C: 0x00 0D  (13 byte(s))
        - T: 13 byte(s), C: 7k325tffg900
        - T: 01 byte(s), C: 0x63 ('c')
        - T: 02 byte(s), C: 0x00 0B  (11 byte(s))
        - T: 11 byte(s), C: 2013/11/26
        - T: 01 byte(s), C: 0x64 ('d')
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 09:02:09
        - T: 01 byte(s), C: 0x65 ('e')
        - T: 04 byte(s), C: 00 0A 16 C4 (661188 bytes)
Tamanho do cabecalho: 100 bytes
Carregando a configuracao Counter (QSPI@C00000 -> DDR3@C0204000)... Terminado!
Processando cabecalho de FSM em QSPI@0D00000
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 0x0F F0 0F F0 0F F0 0F F0 00
        - T: 02 byte(s), C: 0x00 01  (1 byte(s))
        - T: 01 byte(s), C: 0x61 ('a')
        - T: 02 byte(s), C: 0x00 21  (33 byte(s))
        - T: 33 byte(s), C: FSM_routed.ncd;UserID=0xFFFFFFFF
        - T: 01 byte(s), C: 0x62 ('b')
        - T: 02 byte(s), C: 0x00 0D  (13 byte(s))
        - T: 13 byte(s), C: 7k325tffg900
        - T: 01 byte(s), C: 0x63 ('c')
        - T: 02 byte(s), C: 0x00 0B  (11 byte(s))
        - T: 11 byte(s), C: 2013/11/26
        - T: 01 byte(s), C: 0x64 ('d')
        - T: 02 byte(s), C: 0x00 09  (9 byte(s))
        - T: 09 byte(s), C: 09:02:02
        - T: 01 byte(s), C: 0x65 ('e')
        - T: 04 byte(s), C: 00 0A 16 C4 (661188 bytes)
Tamanho do cabecalho: 96 bytes
Carregando a configuracao FSM (QSPI@D00000 -> DDR3@C0304000)... Terminado!
Fim!
```

Figura 6.4: Resultado da execução do programa embarcado gravado na memória QSPI Flash.

Capítulo 7

Experimento 4 - Autoreconfiguração com *MicroBlaze*

Espera-se neste experimento projetar e implementar um sistema baseado em MicroBlaze que se reconfigure de forma autônoma, ou seja, sem a necessidade nenhum comando externo adicional. Para isto, deve-se entender o funcionamento de alguns periféricos e seus *drivers*, como integrar tudo de forma satisfatória, e o fluxo de projeto que permita as funcionalidades desejadas.

7.1 Introdução Teórica

O projeto em questão pretende utilizar um microcontrolador para acionar a troca de comportamentos de uma partição reconfigurável totalmente independente. Tal feito pode, em teoria, ser alcançado facilmente através do periférico AXI4 HWICAP, que acessa as portas de configuração ICAP e ICAPE2, descritas a seguir.

7.1.1 ICAP e ICAPE2

As interfaces ICAP e ICAPE2 (*Internal Configuration Access Port*) são formas de se acessar, tanto para leitura quanto para escrita, a configuração interna do FPGA. Algumas das aplicações mais comuns que a utilizam incluem sistemas *MultiBoot* (**????**), sitemas frequentemente atualizados e sistemas críticos que necessitam de frequente verificação do correto funcionamento do sistema (**????**), e.g. sistemas embarcados em aplicações espaciais sucetíveis a radiações capazes de alterar o conteúdo dos elementos lógicos. Estas portas de configuração se utilizam de um protocolo identico ao da interface SelectMAP (**??**Xilinx Inc., 2013a), que se utiliza de uma série de comandos para iniciar um dos vários procedimentos de programação. Felizmente utiliza-se o controlador AXI4 HWICAP (**??**), que permite o controle da ICAP e ICAPE2 através do MicroBlaze.

Estas interfaces, tanto a SelectMAP quanto as ICAP e ICAPE2, possuem um largura de banda de 32 bits e frequência máxima de operação de 100 MHz, permitindo uma taxa de tranferência de até 3.2 Gbps (Xilinx Inc., 2013a). Esta taxa de transferência permite que configurações parciais como a do experi-

mento 1 sejam programadas em apenas 1.65 milisegundos (611288 bytes $\cdot 8 \frac{\text{bits}}{\text{byte}} \div 3.200.000.000 \frac{\text{bits}}{\text{segundo}} = 0,00165$ segundo).

A única diferença entre as interfaces ICAP e ICAPE2 é a ausência de um sinal de espera na segunda. A ICAPE2, apesar de ter um comportamente mais determinístico no que diz respeito aos tempos de escrita e leitura, não está disponível para alguns dispositivos. A série de FPGAs 7 possui suporte a ICAPE2.

7.1.1.1 Inversão dos bytes

A SelectMAP necessita de uma inversão na ordem dos bits de cada byte do arquivo de configuração (**??**), incluindo a palavra de sincronia, mencionada na seção 6.1.1. Esta inversão só é aplicavel no uso das interfaces Serial, SelectMAP, e por consequência ICAP e ICAPE2, e BPI (**??**). Note que alguns tipos de arquivos sintetisados, como o MCS e o HEX, já podem ter estes bytes invertidos não sendo necessário invertê-los novamente. Não foi encontrada uma explicação para esta inversão.

7.1.2 Fluxo de Projeto

Este projeto começará no *Project Navigator*, onde se instanciarão os componentes importados do experimento 1. O arquivo referente ao MicroBlaze, gerando através de ferramentas de auxílio a construção deste tipo de componentes, também será adicionado. Note, porém, que o fluxo de projeto não pode ser determinado com precisão, como mostrado na figura 7.1. Cada um dos passos ISE, SDK e XPS necessitam de arquivos que são gerados em um dos outros, tornando este projeto de implementação indefinida.

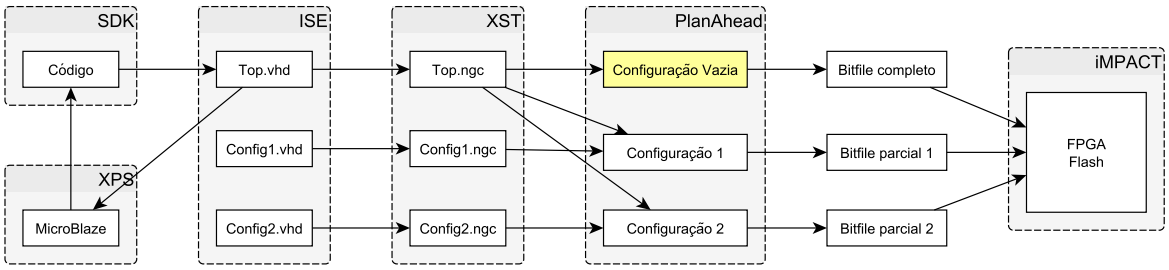


Figura 7.1: Fluxo de ferramentas para desenvolvimennto do projeto.

Em busca por soluções, descobriu-se que é possível quebrar este ciclo através da construção manual de um arquivo de mapeamento de memória do tipo *Block RAM*. Sendo assim, o passo referente ao SDK não precisaria de informações do passo XPS. Apesar disso, este passo possui um grau de complexidade muito elevado, além do escopo deste projeto.

7.2 Resultados

Este experimento foi abordado de diversas formas diferentes, mas nenhuma levou em nenhum resultado palpável. Tentou-se implementar o microcontrolador com e sem memórias DDR3, visto que ela foi a causadora de diversos erros, mas até sem ela observou-se a necessidade de construção manual de arquivos complexos, processo impraticável considerando o escopo deste projeto.

7.3 Conclusão

Muitos erros e pouca documentação forçaram este experimento a ser considerado fracassado. Sabe-se que sua realização não é impossivel, mas dadas as condições de tempo e conhecimento, deverá ser deixado de lado.

Capítulo 8

Experimento 5 - Autoreconfiguração com *MicroBlaze* 2

Espera-se neste experimento projetar e implementar um sistema baseado em MicroBlaze que se reconfigure de forma autônoma, ou seja, sem a necessidade nenhum comando externo adicional. Para isto, deve-se entender o processo de criação de periféricos reconfiguráveis e seus *drivers*, o funcionamento do periférico AXI4 HWICAP e integrar tudo de forma satisfatória.

Ao contrário do experimento anterior, neste experimento não se utilizará o *Project Navigator*, eliminando assim várias complexidades introduzidas por ele.

8.1 Introdução Teórica

O desenvolvimento de sistemas que suportem a autorreconfiguração ainda não é totalmente suportado pelas ferramentas da Xilinx, como foi mostrado no experimento anterior. Algumas tarefas, como a inclusão de um executável em um processador (através do mapa de memória) instanciado em um sistema com módulos reconfiguráveis, precisam ser feitas a mão. Apesar disso, as ferramentas suportam algumas outras opções, como a inclusão direta de um módulo reconfigurável como periférico em um microcontrolador. Esta segunda opção será a abordada neste experimento.

8.1.1 Periférico Reconfigurável

A ferramenta XPS permite a construção de periféricos com comportamentos completamente arbitrários. Estes periféricos podem ser utilizados para instanciar módulos reconfiguráveis (Xilinx Inc., 2013c). O PlanAhead deve ser usado em conjunto com o XPS para a definição das partições e configurações.

8.1.2 Fluxo do Projeto

O projeto se inicia desenvolvendo a lógica do periférico reconfigurável, processo similar ao passo ISE nos experimentos anteriores. Em seguida constrói-se o microprocessador utilizando este periférico. Com o processador pronto, pode-se programá-lo, mesmo que ainda não possa testá-lo. Por último, utiliza-se o PlanAhead para criar as diversas configurações e partições, sintetisando-as e implementando-as.

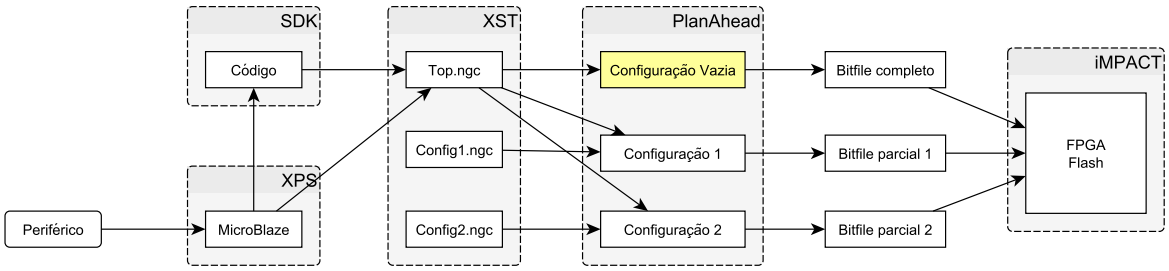


Figura 8.1: Fluxo de ferramentas para desenvolvimennto do projeto.

8.2 Experimento

Este teste foi arquitetado para utilizar o máximo possivel dos elementos já desenvolvidos neste trabalho. Utilizará-se os comportamentos reconfiguráveis desenvolvidos no experimento 1 para a modelagem do comportamento do periférico reconfigurável, o microprocessador desenvolvido no experimento 2 sem a memória DDR3 e o programa desenvolvido no experimento 3 considerando a ausência da memória DDR3, para se construir um sistema microprocessado que possa acionar a mudança de configurações de uma partição reconfigurável.

8.2.1 XPS

Ao se abrir o programa, deve-se construir um sistema utilizando o BSB, assim como no experimento 2. Quando questionado, seleciona-se os periféricos segundo a figura 8.2. Note que a interface UART tem *Baud Rate* de 115200 e o controlador da memória QSPI Flash tem modo de operação *Quad*.

Quando o programa termina de abrir, deve-se selecionar a opção "*Create or Import Peripheral...*" do menu "*Hardware*". Pode-se deixar todas as opções como estão, com excessão da "*Software reset*" na página de "*IPIF (IP Interface) Services*". Nota-se que foi criada a pasta "pcores" com uma pasta com nome igual ao periférico recém criado. Deve-se modificar os arquivos padrões criados para incluir a lógica do experimento 1. Um exemplo de como se modificar este arquivo está presente no anexo. Após modificado o periférico, deve-se reescanear a pasta de periféricos do usuário através da opção "*Rescan User Repositories*" do menu "*Project*". Em seguida pode-se adicionar este periférico ao sistema.

É necessário também a inclusão do periférico "*FPGA Internal Configuraion Access Port*" (AXI_HWICAP). Na adição, deve-se marcar a opção "*Instantiate STARTUP primitive in the HWICAP core*" para que o pino EOS (*End of Startup*) não seja mais necessário. Por causa disso é necessário também a mudança da opção "*Use Startup*" para falso nas configurações da memória QSPI Flash.

”

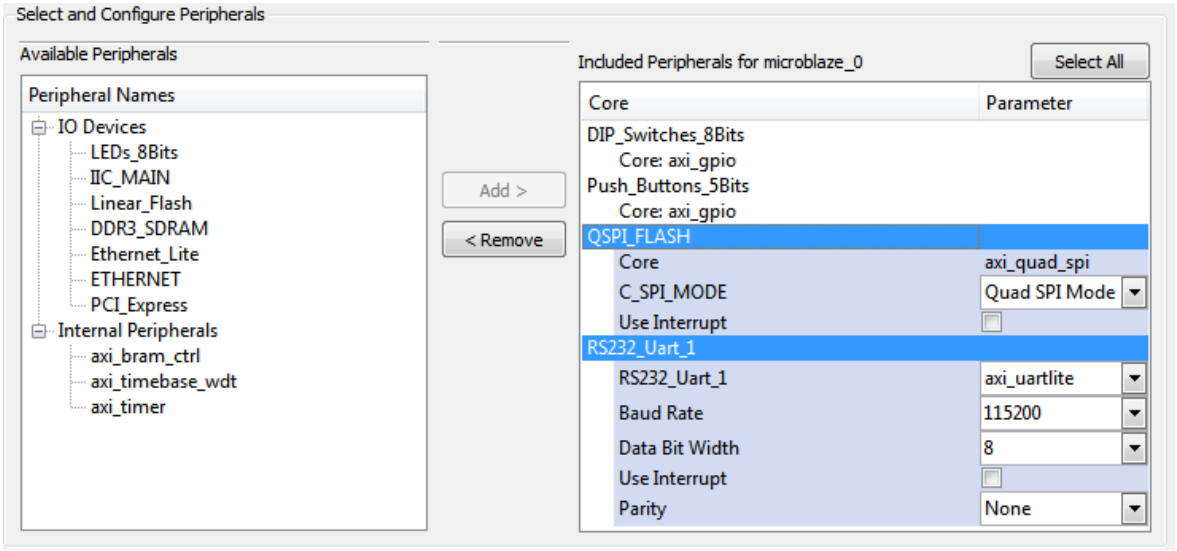


Figura 8.2: Fluxo de ferramentas para desenvolvimennto do projeto.

É preciso ainda ’

8.2.2 XST

8.2.3 PlanAhead

8.3 Resultados

8.4 Conclusão

Observa-se que ainda é possível construir um sistema totalmente independente de microcontroladores, apesar de isto aumentar muito a complexidade do projeto. Um bom balanço pode ser um sistema onde um MicroBlaze com um *bootloader* está implementado em uma partição reconfigurável e pode ser apagado após o início do sistema, restando assim apenas entender o funcionamento da interface ICAP ou ICAPE2 e do controlador de DDR3 criado pelo MIG.

Parte III

Resultados

Capítulo 9

Resultados Experimentais

Resumo opcional.

9.1 Introdução

Na introdução deverá ser feita uma descrição geral dos experimentos realizados.

Para cada experimentação apresentada, descrever as condições de experimentação (e.g., instrumentos, ligações específicas, configurações dos programas), os resultados obtidos na forma de tabelas, curvas ou gráficos. Por fim, tão importante quando ter os resultados é a análise que se faz deles. Quando os resultados obtidos não forem como esperados, procurar justificar e/ou propor alteração na teoria de forma a justificá-los.

9.2 Avaliação do algoritmo de resolução da equação algébrica de Riccati

O algoritmo proposto para solução da equação algébrica de Riccati foi avaliado em diferentes máquinas. Os tempos de execução são mostrados na Tabela 9.1. Nesta tabela, os algoritmos propostos receberam a denominação CH para Chandrasekhar e $CH + LYAP$ para Chandrasekhar com Lyapunov. As implementações foram feitas em linguagem *script* MATLAB.

Observa-se que o algoritmo $CH + LYAP$ apresenta tempos de execução superiores com relação ao algoritmo CH. Entretanto, era esperado que o algoritmo CH fosse mais rápido. Este resultado se justifica pelo fato de o algoritmo CH fazer uso de funções embutidas do MATLAB. Já o aloritmo $CH + LYAP$ faz uso também de funções *script* externas, aumentando bastante seu tempo computacional.

Tabela 9.1: Tempos de execução em segundos para diferentes máquinas

Algoritmo	**Laptop 1.8 GHz**	**Desktop PIII 850 MHz**	**Desktop MMX 233**	**Laptop 600 MHz**
Matlab ARE	649,96	1.857,5	7.450,5	9.063,9
CH	259,44	606,4	2.436,5	2.588,5
$CH + LYAP$	357,86	952,9	3.689,2	3.875,0

Capítulo 10

Conclusões

Este capítulo é em geral formado por: um breve resumo do que foi apresentado, conclusões mais pertinentes e propostas de trabalhos futuros.

REFERÊNCIAS BIBLIOGRÁFICAS

ASHENDEN, P. J. Book. *Digital design: an embedded systems approach using Verilog*. Morgan Kaufmann Publishers Amsterdam ; Boston, 2008. xx, 557 p. : p. ISBN 9780123695277 0123695279. Disponível em: <http://www.loc.gov/catdir/toc/ecip0719/2007023242.html>.

BACKUS, J. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Commun. ACM*, ACM, New York, NY, USA, v. 21, n. 8, p. 613–641, ago. 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359576.359579>.

CHEN, W. *The Electrical Engineering Handbook*. Elsevier Science, 2004. ISBN 9780080477480. Disponível em: <http://books.google.com.br/books?id=qhHsSlazGrQC>.

CULLINAN, C. et al. *Computing Performance Benchmarks among CPU, GPU, and FPGA*. 2012. Disponível em: <http://m.wpi.edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking_Final.pdf>.

DDR3 SDRAM SODIMM. [S.l.], 2010.

EL-GHAZAWI, T. A. et al. The promise of high-performance reconfigurable computing. *IEEE Computer*, v. 41, n. 2, p. 69–76, 2008.

ESTRIN, G. Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. *IEEE Ann. Hist. Comput.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 24, n. 4, p. 3–9, out. 2002. ISSN 1058-6180. Disponível em: <http://dx.doi.org/10.1109/MAHC.2002.1114865>.

FEDELI, R.; POLLONI, E.; PERES, F. *Introdução à Ciência da Computação*. 1. ed. [S.l.]: Thomson, 2003.

HAREL, D.; FELDMAN, Y. *Algorithmics: The Spirit of Computing*. 3rd. ed. [S.l.]: Addison Wesley, 2004.

HARTENSTEIN, R. A decade of reconfigurable computing: a visionary retrospective. In: *Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001. (DATE '01), p. 642–649. ISBN 0-7695-0993-2. Disponível em: <http://dl.acm.org/citation.cfm?id=367072.367839>.

HARTENSTEIN, R. W.; KRESS, R. A datapath synthesis system for the reconfigurable datapath architecture. In: *Proceedings of the 1995 Asia and South Pacific Design Automation Conference*. New York, NY, USA: ACM, 1995. (ASP-DAC '95). ISBN 0-89791-766-9. Disponível em: <http://doi.acm.org/10.1145/224818.224959>.

HAUCK, S.; DEHON, A. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 0123705223, 9780080556017, 9780123705228.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 012383872X, 9780123838728.

MAXXPI. *MaxxPI's TOP15 Flops List*. 2013. <http://www.maxxpi.net/pages/result-browser/top15---flops.php>. [Online; acessado em 16 de julho de 2013].

MOORE, G. E. Cramming More Components onto Integrated Circuits. *Electronics*, IEEE, v. 38, n. 8, p. 114–117, abr. 1965. ISSN 0018-9219. Disponível em: <http://dx.doi.org/10.1109/jproc.1998.658762>.

N25Q128: 128-Mbit 3 V, multiple I/O, 4-Kbyte subsector erase on boot sectors, XiP enabled, serial flash memory with 108 MHz SPI bus interface. [S.l.].

PATTERSON, D.; HENNESSY, J. *Computer Organization and Design: The Hardware/software Interface*. [S.l.]: Morgan Kaufmann, 2005.

THOMAS, D. E.; MOORBY, P. R. *The VERILOG Hardware Description Language*. 3rd. ed. Norwell, MA, USA: Kluwer Academic Publishers, 1996. ISBN 0792397231.

TOP500. *June 2013's Supercomputer Sites*. 2013. <http://www.top500.org/lists/2013/06/>. [Online; acessado em 16 de julho de 2013].

VAJDA, A. *Programming Many-Core Chips*. Dordrecht: Springer, 2011.

VASSILIADIS, S.; SOUDRIS, D. *Fine- and Coarse-Grain Reconfigurable Computing*. Springer London, Limited, 2007. ISBN 9781402065057. Disponível em: <http://books.google.com.br/books?id=2Vsvwyq7BREC>.

WILLIAMS, A. *C++ Concurrency in Action: Practical Multithreading ; [for the New C++ 11 Standard]*. Manning Publications Company, 2012. (Manning Pubs Co Series). ISBN 9781933988771. Disponível em: <http://books.google.com.br/books?id=EttPPgAACAAJ>.

WOODS, R. et al. *FPGA-based Implementation of Signal Processing Systems*. [S.l.]: Wiley Publishing, 2008. ISBN 0470030097, 9780470030097.

Xilinx Inc. *UG702: Partial Reconfiguration User Guide*. [S.l.], 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/ug702.pdf>.

Xilinx Inc. *UG743: Partial Reconfiguration Tutorial*. [S.l.], 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/PlanAhead_Tutorial_Partial_Reconfiguration.pdf>.

Xilinx Inc. *UG744: Partial Reconfiguration of a Processor Peripheral Tutorial*. [S.l.], 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/PlanAhead_Tutorial_Reconfigurable_Processor.pdf>.

Xilinx Inc. *UG748: Hierarchical Design Methodology Guide*. [S.l.], 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/Hierarchical_Design_Methodology_Guide.pdf>.

Xilinx Inc. *UG810: KC705 Evaluation Board for the Kintex-7 FPGA*. [S.l.], 2013. Disponível em: <http://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf>.