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. Fulano de Tal 1, ENM/UnB _____
Examinador interno

Prof. Fulano de Tal 2, ENM/UnB _____
Examinador externo

Prof. Fulano de Tal 3, ENM/UnB _____
Examinador externo

Dedicatória

Dedicatória do autor 1

Lucas Sousa de Oliveira

Agradecimentos

A inclusão desta seção de agradecimentos é opcional e fica à critério do(s) autor(es), que caso deseje(em) inclui-la deverá(ao) utilizar este espaço, seguindo está formatação.

Lucas Sousa de Oliveira

RESUMO

O presente texto apresenta normas a serem seguidas por alunos do Curso de Engenharia Mecatrônica da Universidade de Brasília para redação de relatório na disciplina Projeto de Graduação 2. Tais normas foram aprovadas pela Comissão de Graduação do Curso de Engenharia Mecatrônica em julho/2005. São apresentadas instruções detalhadas para a formatação do trabalho em termos de suas partes principais.

ABSTRACT

The same as above, in english.

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*

Capítulo 1

Introdução

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

1.1 Introdução

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

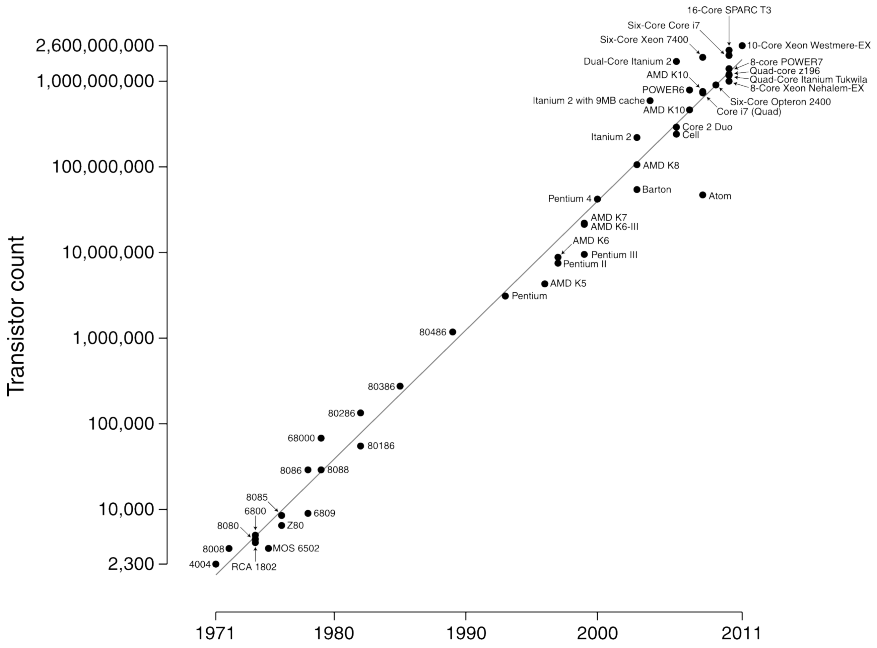


Figura 1.1: Visualização da Lei de Moore. Eixos em escala logarítmica. Gráficos extraidos do Wikipedia sob licença *Creative Commons*.

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. Ambas arquiteturas são muito utilizadas nos dias de hoje, sendo as RISC mais comum em dispositivos com restrições de consumo de energia e as CISC mais comuns em dispositivos de computação mais genérica.

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

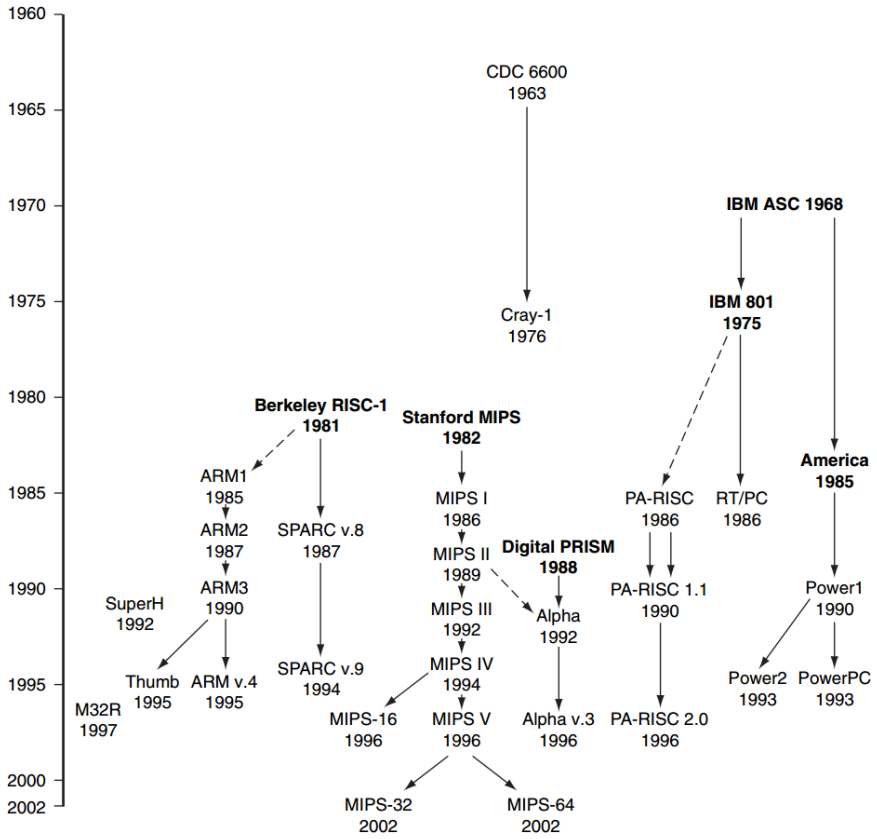


Figura 1.2: Linha do tempo das arquiteturas RISC, extraido de (HENNESSY; PATTERSON, 2011). Em negrito estão as iniciativas de pesquisa, em contraste às comerciais.

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 modificação 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).

1.1.1 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 pela ponte de seleção, que permanece carregada até quando desligada. Estas técnicas permitem a resis-

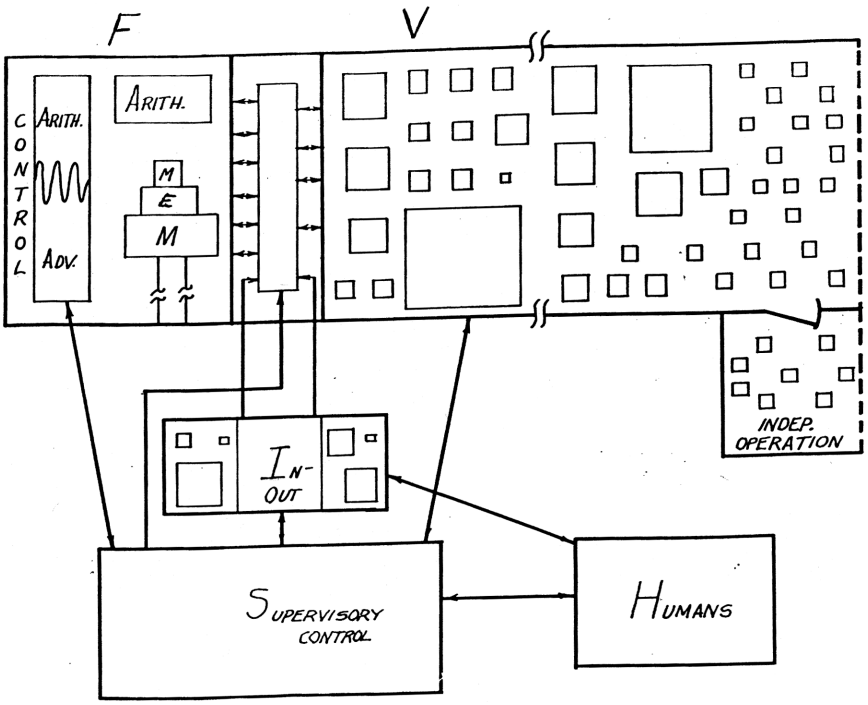


Figura 1.3: Esquemático com o F+V, extraido de (ESTRIN, 2002).

tência 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 arquiteturas 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.

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.

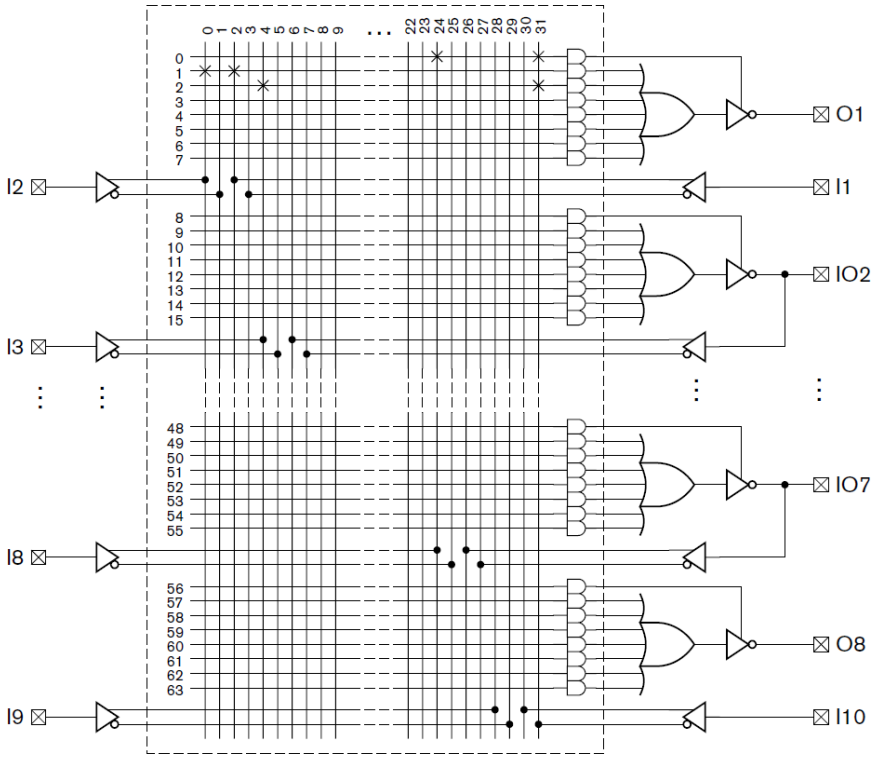


Figura 1.4: Circuito interno de um dispositivo do tipo PAL, extraido de (ASHENDEN, 2008).

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 1.4. 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.

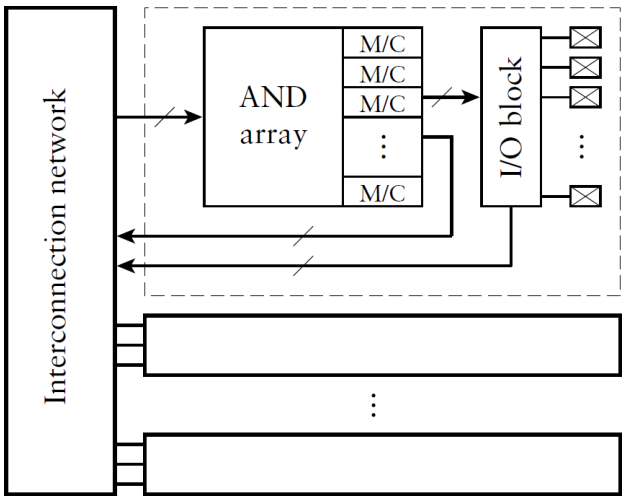


Figura 1.5: Representação de dispositivos do tipo CPLD, extraido de (ASHENDEN, 2008).

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 1.5. Sua arquitetura é baseada no mar de portas, como mostra a figura 1.5. 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 1.6, 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.

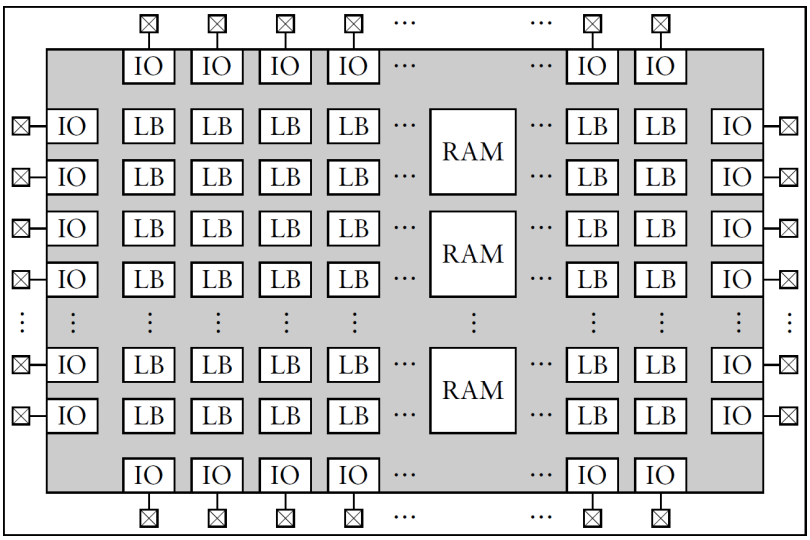


Figura 1.6: Representação de dispositivos do tipo FPGA, extraido de (ASHENDEN, 2008).

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 1.6. Eles incluem blocos lógicos que podem implementar tanto 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 1.1.1, 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.

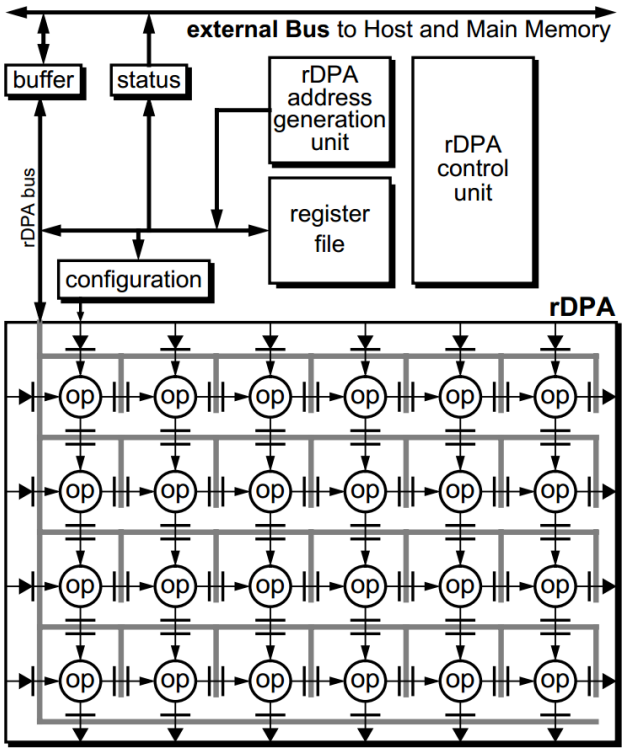


Figura 1.7: Modelo de rDPA do tipo KressArray-I, extraido de (HARTENSTEIN; KRESS, 1995).

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 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 1.7. 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.

1.1.1.1 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.

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".

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.

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.

1.1.2 Classes de Reconfiguração

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.

1.1.2.1 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.

1.1.2.2 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.

1.1.2.3 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.

1.1.2.4 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 é auto-reconfiguração, é realizado através de um circuito controlador, em geral um microcontrolador, presente no sistema, mas externo ao FPGA. Quase todos os FPGAs modernos dão suporte a esta tecnologia.

Este tipo de reconfiguração não pode ser realizado em conjunto com a reconfiguração total. Isto porque sistema reconfiguráveis dinamicamente necessitam 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.

1.1.2.5 Auto-reconfiguração

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 das partições. Este tecnologia é nova e ainda representa uma forte área de pesquisa. Não são todos os FPGAs que dão suporte a este tipo de reconfiguração.

Para que a auto-reconfiguração 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.

1.1.3 Dispositivos e Ferramentas

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.

1.1.3.1 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 auto-reconfiguração.

Tipo	Família	Breve Descrição
CPLD	MAX® II	Tecnologia com numerosos blocos similares aos PALs.
FPGA	Cyclone	Baixo custo, repleto de elementos de memória
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, como apresentado em (WOODS et al., 2008).

1.1.3.2 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/E/EM	Família principal da Xilinx, FPGAs de alta performance.
FPGA	Spartan	FPGA de baixo custo e alto volume de vendas.

Tabela 1.3: Famílias de produtos da Xilinx, como apresentado em (WOODS et al., 2008).

1.2 Projeto

Capítulo 2

Revisão Bibliográfica

Este capítulo visa apresentar as ferramentas disponíveis e o estado-da-arte da auto-reconfiguração.

2.1 Introdução

2.2 Ferramentas

2.3 Componentes

2.3.1 *Intellectual Property*

2.3.2 Interfaces

Capítulo 3

Desenvolvimento

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 4.

3.1 Introdução

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.

3.2 Experimento 1 - Reconfiguração Dinâmica

De forma a dar validade a todo o projeto, foi preciso desenvolver um experimento para se entender o processo de desenvolvimento de sistemas reconfiguráveis dinamicamente e algumas peculiaridades do kit de desenvolvimento.

3.2.1 Fluxo de Ferramentas

A primeira coisa 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. Como pode-se ver da figura 3.2a, 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 mostrado na figura 3.2b, além das ferramentas do fluxo tradicional, faz-se necessário o uso da ferramenta XST para a síntese do *netlist* 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.

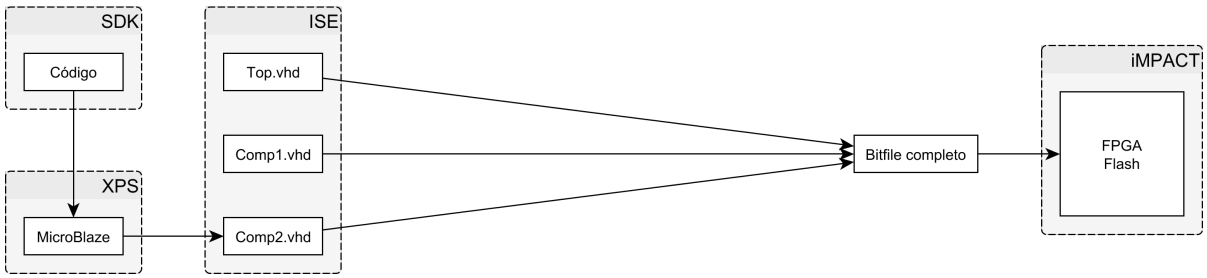
Reconfiguração parcial pede uma síntese utilizando o método “de baixo para cima” (*bottom-up*), mas uma implementação “de cima para baixo” (*top-down*) (Xilinx Inc., 2013b), ou seja, a implementação acontece construindo primeiro a interface com o sistema e depois os componentes auxiliares, mas a síntese precisa ser realizada no sentido oposto. Esta implementação é equivalente a se construir diversos projetos tradicionais com alguma lógica em comum, onde a síntese deve garantir que esta lógica em comum seja implementada da mesma forma para as diferentes configurações (Xilinx Inc., 2013a).

3.2.2 Peculiaridades

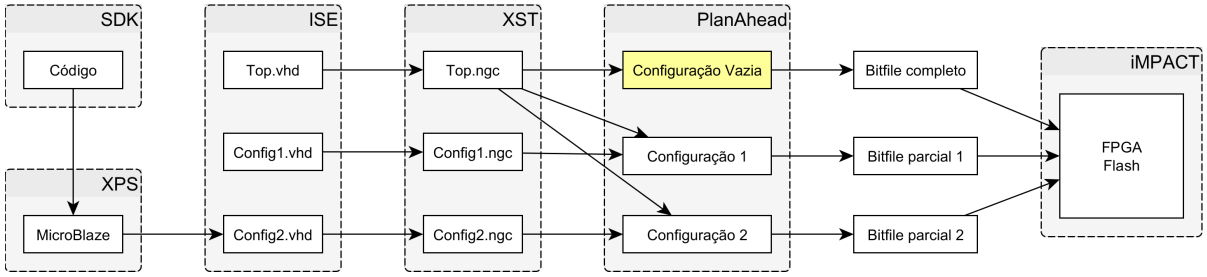
O kit de desenvolvimento utilizada apresenta algumas peculiaridades com relação aos kits comuns. A seguir serão apresentadas algumas destas peculiaridades.

3.2.2.1 Relógios

Diferentemente das FPGAs comuns, a que está presente neste kit contém um relógio diferencial, ou seja, dois sinais compoem tal relógio. A razão para tal é a presença de circuitos sensíveis a interferência, 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.



(a) Foto ilustrativa do *software-flow* tradicional. Note que o uso do microcontrolador MicroBlaze é opcional, tornando os primeiros blocos, SDK e XPS, também opcionais.



(b) Foto ilustrativa do *software-flow* 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 3.2: 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.

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. Por motivos de simplicidade, utilizou-se o SYSCLK. Para poder se trabalhar com o sinal diferencial, construiu-se, utilizando as ferramentas do ISE, um componente para tratamento do sinal de relógio. Este componente recebe o sinal diferencial, reduz sua frequência para 20 MHz, que corresponde ao menor valor suportado pelas PLLs da placa, e emite um sinal convencional.

3.2.3 Teste

Para se entender mais a fundo o fluxo de projeto, nada melhor que construir um projeto. Para isso, implementou-se o sistema esquematizado na figura 3.3. Este sistema contém o "Top" para interfaceamento com a FPGA, 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.

Estrutura de Pastas A questão da organização do projeto em pastas bem específicas é sempre bem mencionado na literatura (Xilinx Inc., 2013a; Xilinx Inc., 2013b; Xilinx Inc., 2013c). Os manuais recomendam a seguinte estrutura de pastas.

```
Projeto/
  Source/          //codigos-fonte organizados segundo particao
  Implementation/  //contem pastas para cada config. dinamica gerada
```

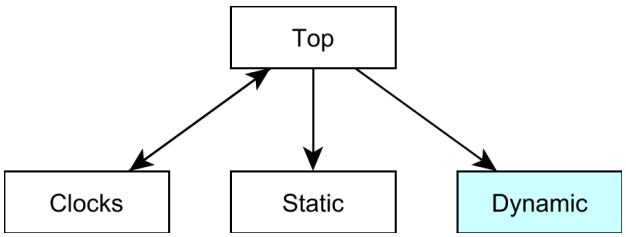


Figura 3.3: Foto ilustrativa do sistema desenvolvido para o teste de validação do experimento 1. Ele é composto por uma parte estática e uma parte dinâmica. O elementos em branco são estáticos e os em azul são dinâmicos.

```
Synth/          //contem pastas com os arquivos .xst e .prj
Tools/          //ferramentas para automacao da sintese
PlanAhead/      //pasta para o projeto do PlanAhead
```

Esta estrutura de pastas foi obedecida por ajudar a manter o ambiente de desenvolvimento limpo.

3.2.3.1 Comportamento

Como explicado anteriormente, 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 3.3. O comportamento individual de cada módulo ou componente será descrito a seguir. Este passo está ilustrado no fluxo de ferramentas da figura 3.2b como ISE.

O componente “Static” possui uma entrada para um relógio pulsando a 2 Hz e uma saída para um LED. Seu comportamento apenas faz com que o LED pisque a uma frequência de 2 Hz, o que permite observar seu funcionamento durante a reconfiguração do componente “Dynamic”.

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 e uma palavra de 4 bits de saída. A frequência de operação deste componente foi escolhida para ser a metade da frequência da “Static” para poder ser visualmente comprovado 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.

3.2.3.2 Síntese

Com o comportamento do projeto definido, o próximo passo segundo o fluxo de ferramentas é a síntese. Este passo é necessário uma vez que o próximo passo, referente ao PlanAhead, não aceita como entrada códigos-fonte. Os códigos-fonte precisam passar por uma etapa de síntese separada para poderem ser importados no PlanAhead. Este passo está ilustrado no fluxo de ferramentas da figura 3.2b como XST.

O comando XST 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"
```

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 cógigo-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 (Xilinx Inc., 2013a; Xilinx Inc., 2013b; Xilinx Inc., 2013c). A única coisa que se faz neste *script* é a construção dinâmica dos comandos com base em listas de arquivos pré-informados.

3.2.4 PlanAhead

Com os arquivos síntetisados, pode-se começar a etapa referente ao PlanAhead. Nela, importaremos os arquivos da etapa anterior, criaremos a partição reconfigurável, mapearemos esta partição no dispositivo, criaremos configurações alternativas, promoremos tais configurações e geraremos 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.

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 3.4, onde se lê “Create New Project”. Na janela que aparece, indicamos o nome do projeto e seu caminho, lembrando que foi criada uma pasta anteriormente especificamente para conter este projeto. Indicamos também que o projeto é do tipo “*Post-synthesis Project*” e que desejamos habilitar a reconfiguração parcial, indicamos quais *netlists* compõe o comportamento do projeto e qual corresponde ao arquivo principal, qual é o arquivo de restrições (*constrains*) e qual é o modelo da FPGA.

Após a criação do projeto, carregam-se os arquivos sintetizados abrindo “*Open Synthesized Design*”. Duas janelas de avisos aparecem, informando que existe uma partição não implementada e aviso sobre alguns pinos.

Pode-se agora definir a partição que armazenará o componente dinâmico. Isso é feito através do painel “*Netlist*”, selecionando a opção “*Set Partition*” do menu que aparece ao se clicar em “dynamic_i” com o botão direito. Na janela que aparece, selecionamos a opção referente à partição reconfigurável, nomeamos o módulo reconfigurável de acordo com o componente que será carregado e indicamos o arquivo 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 definida. Isto pode ser feito pelo mesmo painel “*Netlist*”, selecionando 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

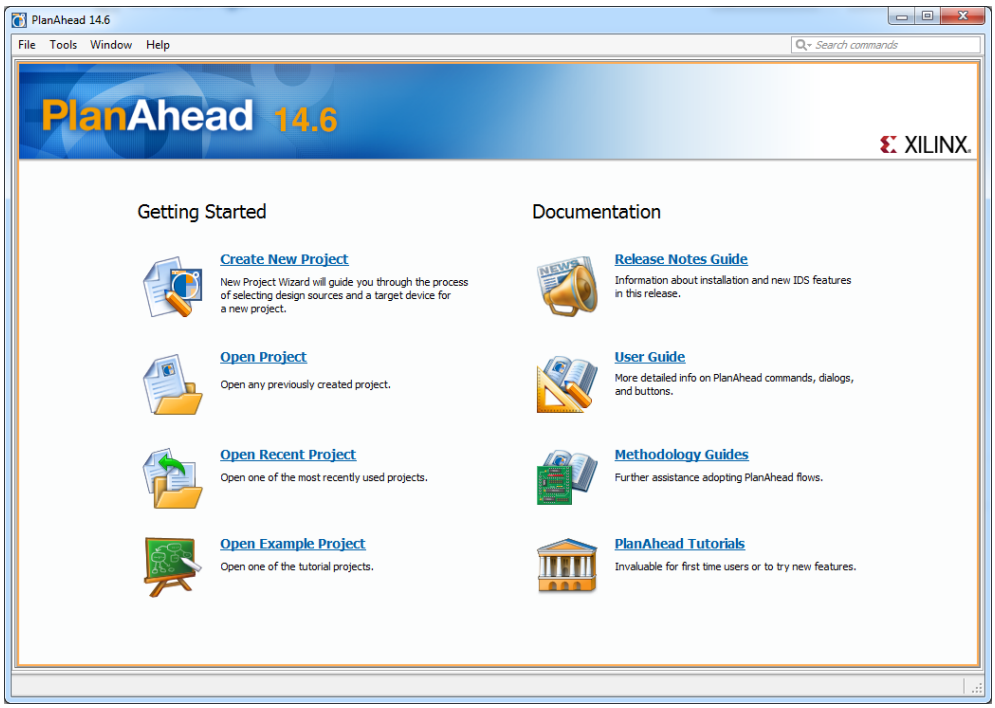


Figura 3.4: Imagem do PlanAhead logo que aberto.

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 erros gerados nos estágios iniciais. Uma região possível, que foi utilizada nesse projeto, foi a que contém as células (*slice*) de X80Y244 a X139Y205. Os nomes das células ficam visíveis através da ampliação do dispositivo.

Uma vez terminado o "*Design Run*", 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. Um módulo vazio pode ser criado usando esta mesma opção, mas selecionando a opção que indica a inclusão de uma *blackbox* 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. Em seguida, as duas novas configurações serão criadas através de "*Design Runs*". Não é necessário promover as partições neste momento, apenas iniciar os processos de criação das partições. Note que os avisos que aparecerem podem ser ignorados.

3.2.5 iMPACT

3.3 Experimento 2 - Teste de Acesso a Memória

3.4 Experimento 3 - Teste do *Bootloader*

3.5 Experimento 4 - Teste da Autoreconfiguração com *Bootloader* Dedicado

3.6 Experimento 5 - Teste da Autoreconfiguração

Capítulo 4

Resultados Experimentais

Resumo opcional.

4.1 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.

4.2 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 4.1. 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 4.1: 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 5

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>.

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>.

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

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. *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>.