

Todo list

Falar mais sobre RISC e CISC (Patterson <file:///E:/resources/Chapters/CD2.20-P374493.pdf>). . .	2
Comentar benchmarking e como eles perdem um pouco do sentido com a configuração de hardware (Patterson <file:///E:/resources/Chapters/CD1.10-P374493.pdf>).	4
Comentar sobre a programação estrutural e comportamental.	10
Comentar a possibilidade de programação estrutural e comportamental, bem como imbutir informações de simulação e timing no Verilog (??).	10
Comentar sobre a programação estrutural e comportamental e dar mais importancia as bibliotecas do VHDL.	10
Me prolongar mais em reconfiguração dinâmica (Hauck2007 p.70)	11
Diferenciar autoreconfiguração de reconfiguração externa.	12
Incluir figura sobre autoreconfiguração do manual da Xilinx (ver Evernote).	12
Descrever melhor os produtos da Altera, atualizando tabela	13
Falar do Quartus	13
Descrever melhor os produtos da Xilinx, atualizando tabela	13
Falar do ISE e do Vivado	13
Descrever objetivos do projeto	14

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 (??).

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-científico-informacional (??). 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 (??). 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 (??). 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 instruções 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 (??).

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) (??). 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

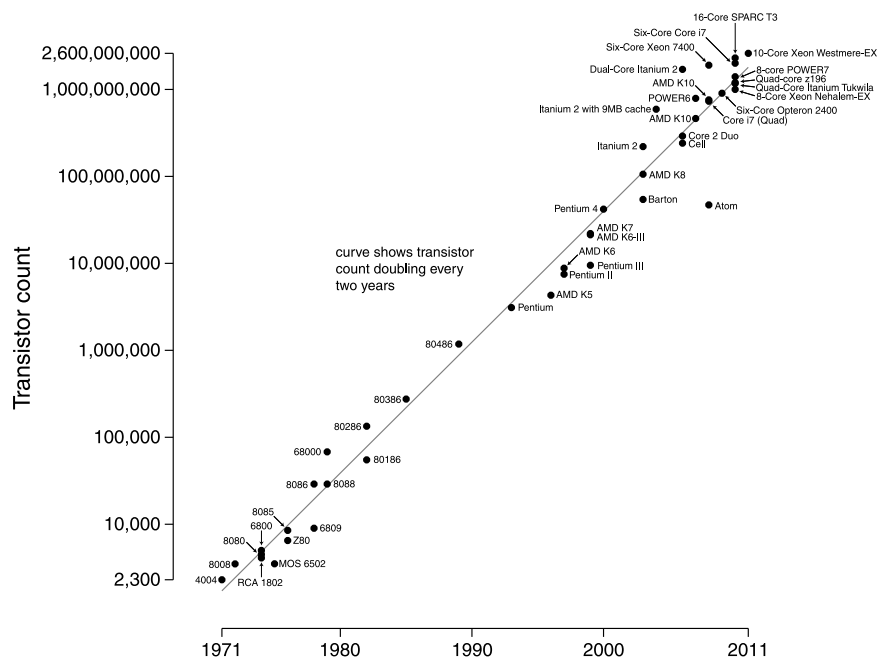


Figura 1.1: Visualização da Lei de Moore. Eixos em escala logarítmica. Gráficos extraídos do Wikipedia sob licença *Creative Commons*.

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.

Falar mais sobre RISC e CISC (Patterson <<file:///E:/resources/Chapters/CD2.20-P374493.pdf>>).

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 (*throughput*) 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 possuíam produtos com esta arquitetura (??).

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 possuírem centenas de núcleos, estes núcleos são simplificados (??). 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 (??).

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 (??).

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

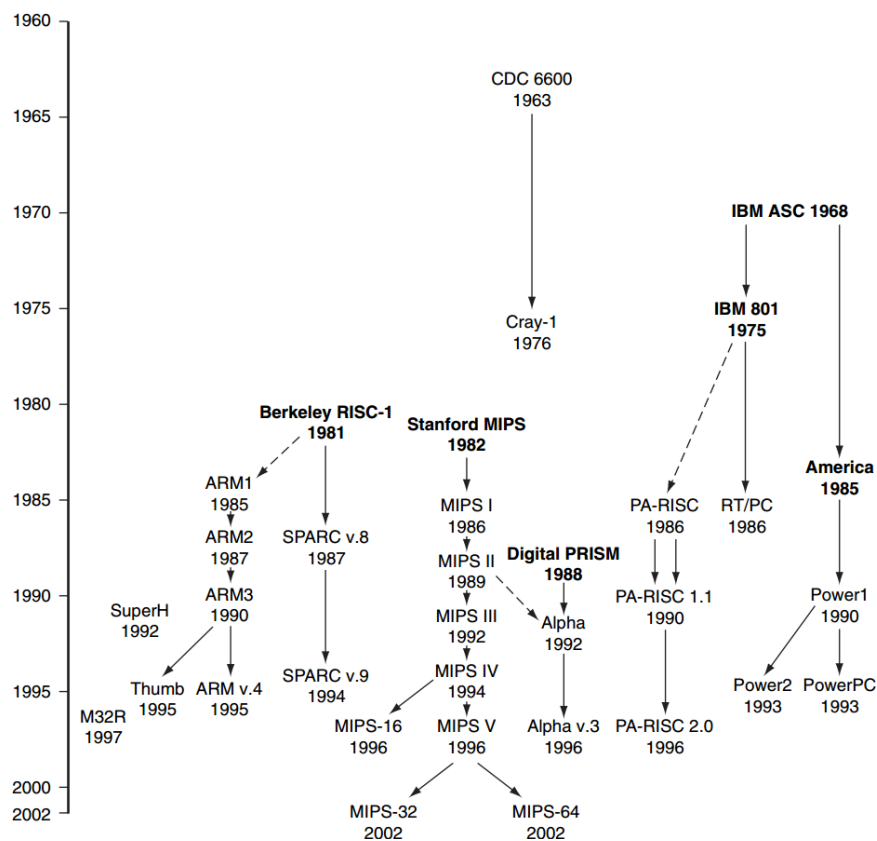


Figura 1.2: Linha do tempo das arquiteturas RISC, extraído de (??). Em negrito estão as iniciativas de pesquisa, em contraste às comerciais.

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 (??). 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 possuíssem 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* (??), sejam eles dedicados ou compartilhados entre os vários processadores. A sua capacidade de processamento atinge níveis extraordinários, ultrapassando 20 GFlops em computadores comuns (??) e 54 PFlops em supercomputadores (??). Apesar disso, a arquitetura Harvard original ainda é muito usada em microcontroladores e processadores digitais de sinal (*Digital Signal Processors* ou DSPs).

Comentar benchmarking e como eles perdem um pouco do sentido com a configuração de hardware (Patterson <<file:///E:/resources/Chapters/CD1.10-P374493.pdf>>).

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 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 (??), 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 (??). 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 (??). 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 flexível 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 possuírem componentes fixos como processadores e unidades de processamento gráficos (GPUs) (??), 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

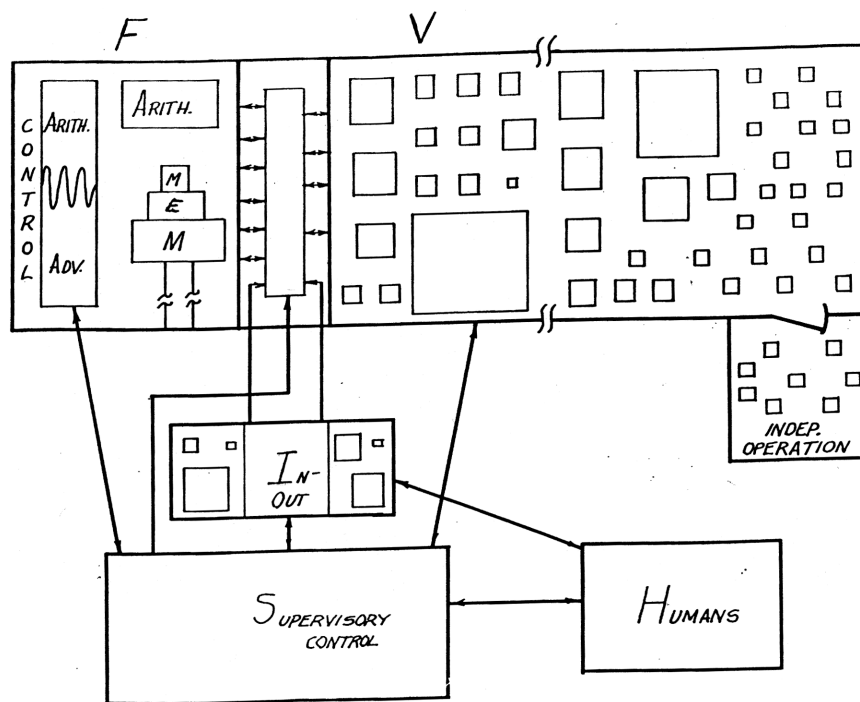


Figura 1.3: Esquemático com o F+V, extraído de (??).

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 (??).

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 (??). 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 sistemas 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 classificadas em três tipos segundo a granularidade. A granularidade diz respeito à 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 (??). 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.

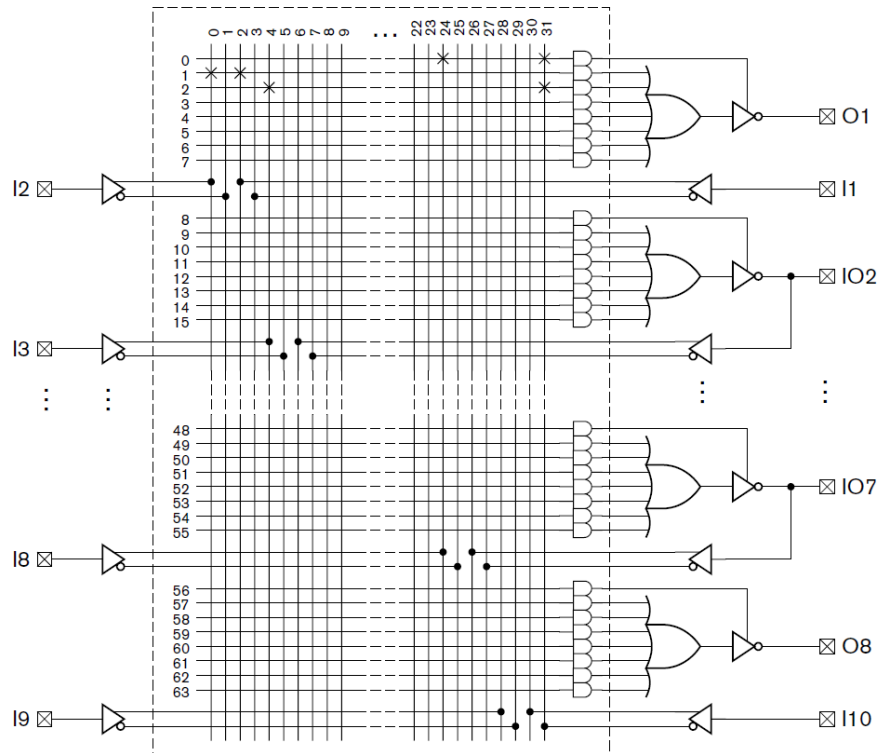


Figura 1.4: Circuito interno de um dispositivo do tipo PAL, extraído de (??).

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 cascadeamento. Alguns tipos especiais de PALs também contêm registradores, permitindo a programação de circuitos sequenciais simples. Outra característica dos PALs que permitiam a construção de circuitos lógicos um pouco mais complexos 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 grande 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.

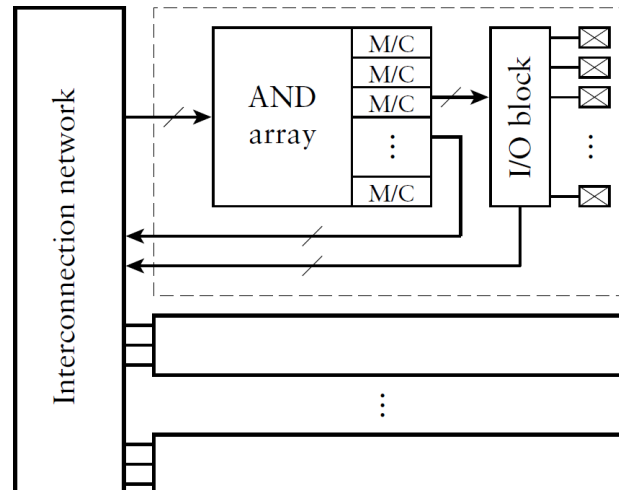


Figura 1.5: Representação de dispositivos do tipo CPLD, extraído de (??).

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 conjunto 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 permite 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.

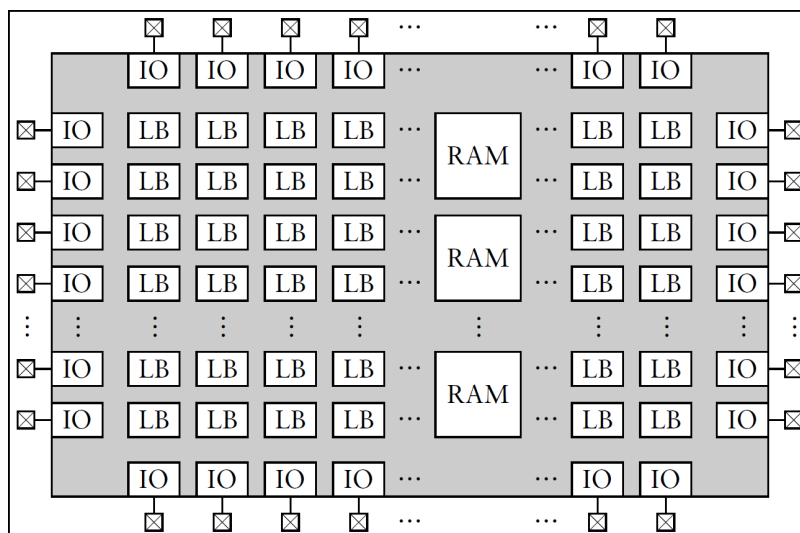


Figura 1.6: Representação de dispositivos do tipo FPGA, extraído de (??).

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 dispositivos reconfiguráveis dos tipos PAL, CPLD e FPGA.

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

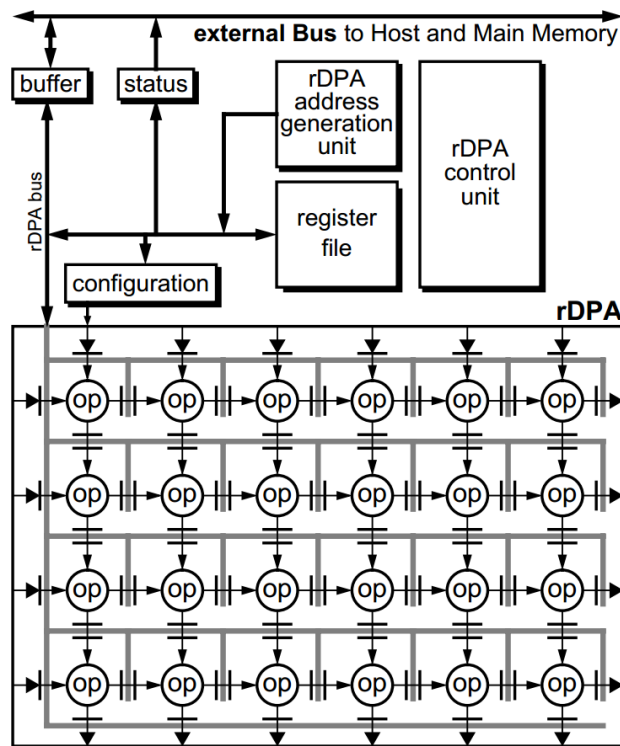


Figura 1.7: Modelo de rDPA do tipo KressArray-I, extraído de (??).

Reconfigurable Datapath Array O *Reconfigurable Datapath Array* é um tipo de sistema reconfigurável com granularidade grossa, normalmente 32 bits (??). 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.

Comentar sobre a programação estrutural e comportamental.

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".

Comentar a possibilidade de programação estrutural e comportamental, bem como imbutir informações de simulação e timing no Verilog (??).

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.

Comentar sobre a programação estrutural e comportamental e dar mais importância as bibliotecas do VHDL.

SystemC SystemC foi desenvolvida em meados dos anos 2000, aproximadamente 15 anos depois do Verilog e VHDL, com o intuito 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 descrição 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 (??). 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 1.1.2.4 quando estática 1.1.2.3.

Me prolongar mais em reconfiguração dinâmica (Hauck2007 p.70)

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 substituídas sem afetar o funcionamento do todo. Normalmente este processo, quando não associado a autoreconfiguração, é realizado através de um circuito externo à FPGA, tal como um controlador, um microcontrolador, ou mesmo um computador, como apresentado na figura 1.8. 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 interfacer com o circuito controlador. Esta partição estática é responsável pelo menos por controlar a comunicação com o circuito controlador.

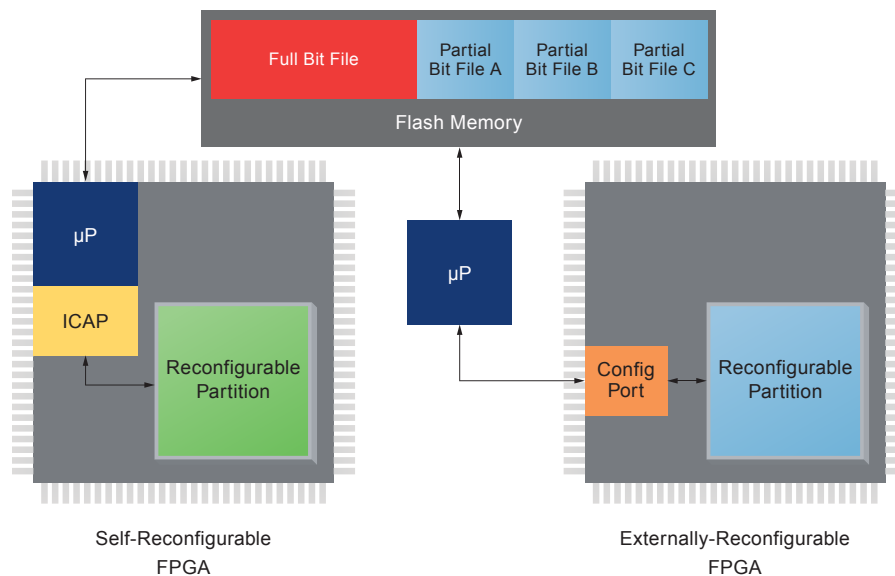


Figura 1.8: Imagem ilustrativa para diferenciação entre autoreconfiguração e reconfiguração externa, extraído de (??).

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. Esta 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.

Diferenciar autoreconfiguração de reconfiguração externa.

Incluir figura sobre autoreconfiguração do manual da Xilinx (ver Evernote).

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 competidora, 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 <i>transceivers</i> seriais e <i>arrays</i> 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 (??).

Descrever melhor os produtos da Altera, atualizando tabela

Falar do Quartus

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 (??), 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.

Descrever melhor os produtos da Xilinx, atualizando tabela

Falar do ISE e do Vivado

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 (??).

1.2 Projeto

Descrever objetivos do projeto