Conteúdo

Capítulo 1

Revisão Bibliográfica

Este capítulo visa apresentar uma breve descrição sobre reconfiguração dinâmica, autorreconfiguração e as ferramentas e conceitos necessários para o seu entendimento.

1.1 Compilação de *Hardware*

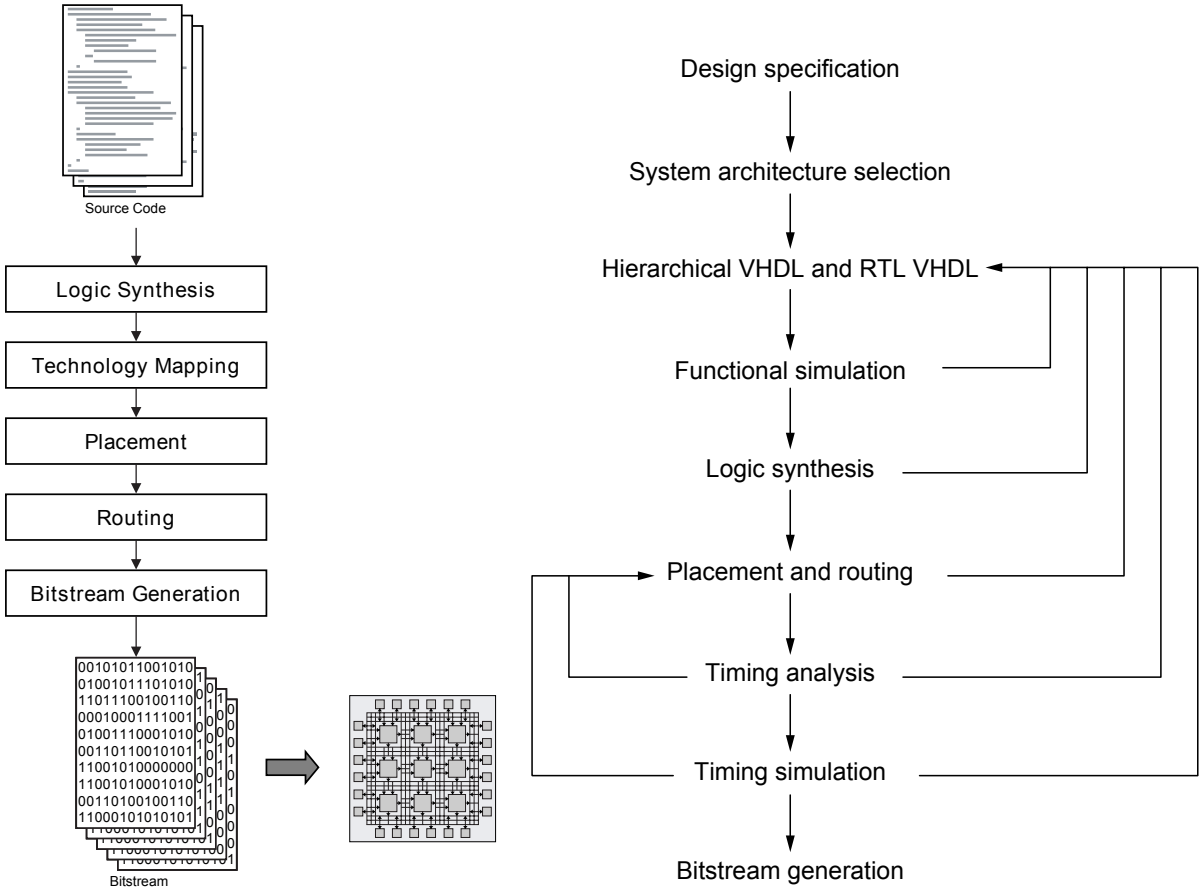
A compilação de *hardware* é um processo primordial no desenvolvimento de sistemas reconfiguráveis, equivalente à compilação para projetos de *software* (**??**). A figura 1.1 apresenta duas imagens que apresentam este processo de compilação.

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 1.1b, 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 (**????**). 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



(a) Imagem ilustrativa do fluxo de compilação de um projeto de *hardware*, extraida de (**??**).

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

Figura 1.1: Imagens do fluxo da compilação de *hardware*.

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.

1.2 *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 (**????**). 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.

1.3 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.

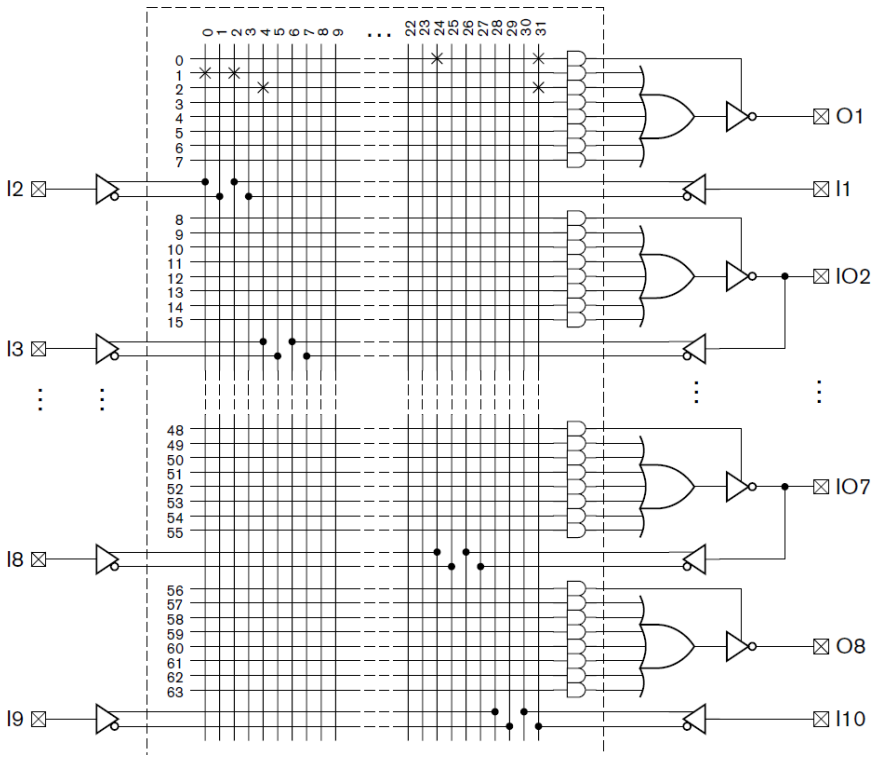


Figura 1.2: Circuito interno de um dispositivo do tipo PAL, extraido de (**??**).

1.3.1 *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 1.2. 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.

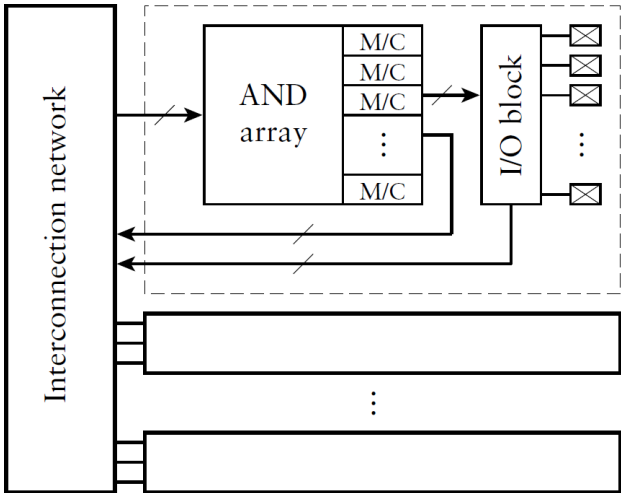


Figura 1.3: Representação de dispositivos do tipo CPLD, extraido de (**??**).

1.3.2 *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 1.3. Sua arquitetura é baseada no mar de portas, como mostra a figura 1.3. 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.

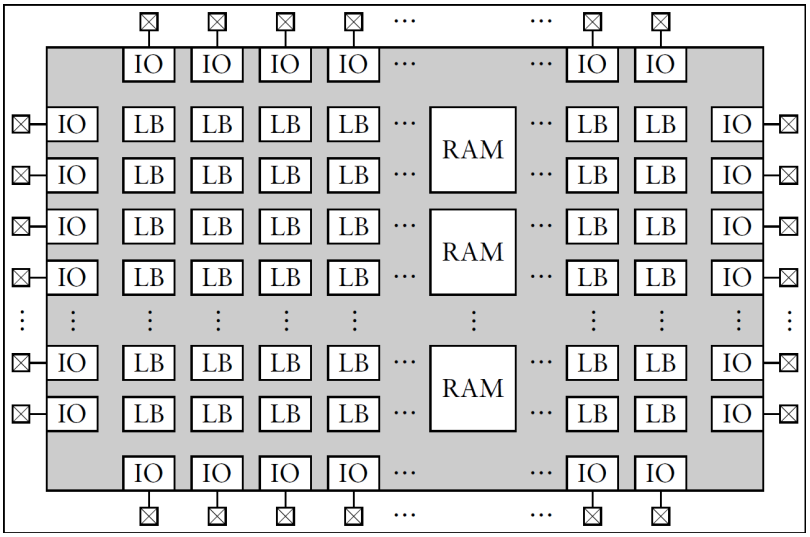


Figura 1.4: Representação de dispositivos do tipo FPGA, extraido de (**??**).

1.3.3 *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 1.4, 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 1.4. 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 **??**, 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.

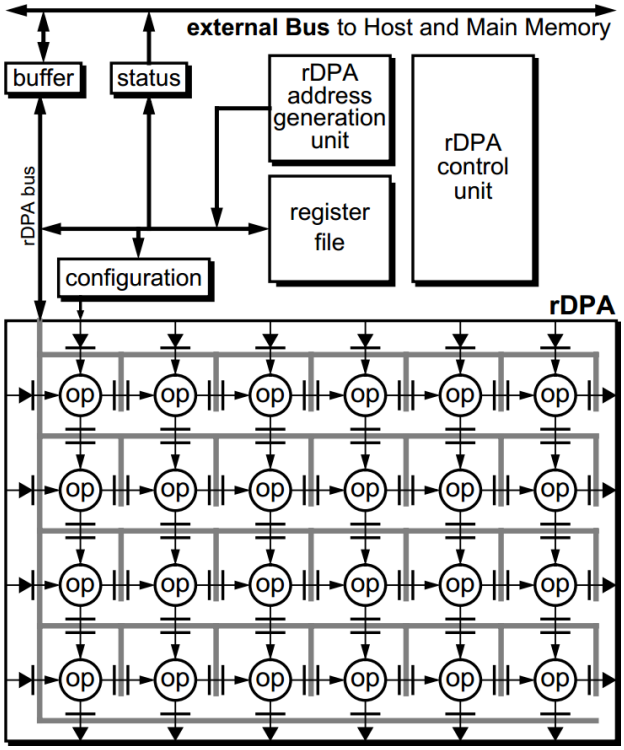


Figura 1.5: Modelo de rDPA do tipo KressArray-I, extraido de (**??**).

1.3.4 *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 1.5. 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.

1.4 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 (**??**). 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.

1.4.1 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.

1.4.2 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.

1.4.3 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.

1.5 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.

1.5.1 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.

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

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 (**??**) e do site da empresa.

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.

1.5.2 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.

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 (**??**) 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.

1.6 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.

1.6.1 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.

1.6.2 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 (1.6.4) quando estática (1.6.3).

1.6.3 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.

1.6.4 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 1.6. Quase todos os FPGAs modernos dão suporte a esta tecnologia.

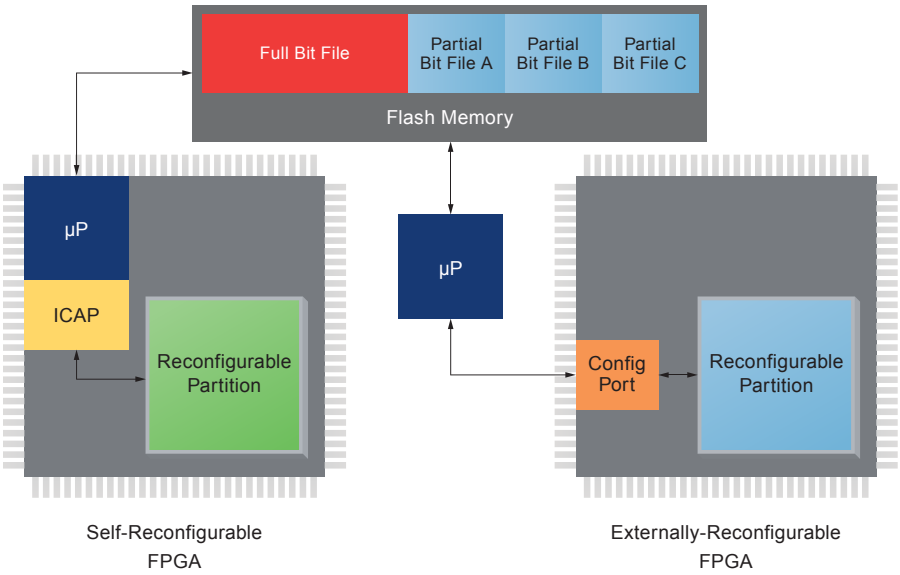


Figura 1.6: Imagem ilustrativa para diferenciação entre autorreconfiguração e reconfiguração externa, extraido de (**??**).

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 (**??**). 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.

1.6.5 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.

1.7 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.

1.7.1 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.

1.7.1.1 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.

1.7.1.2 *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.

1.7.1.3 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.

1.7.1.4 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.