PROJETO DE SISTEMAS DE COMPUTAÇÃO

COMPUTER SYSTEM DESIGN

José Gonzaga SOUZA JÚNIOR[*]

RESUMO

O projeto de sistemas de computação na atualidade é baseado na aplicação de técnicas, metodologias e conhecimento oriundos das áreas de software, arquitetura e hardware os quais, longe de formar um conjunto coeso, apresentam um carater heterogêneo facilmente visível, seja nas divisões internas entre as diversas metodologias de projeto para uma mesma área, seja no carater estanque de cada área. Entretanto, com o advento dos processadores RISC e a conseqüente mudança no paradígma de projeto arquitetural, cresce a demanda pela integração entre as áreas, fato que já é realidade para arquitetura e hardware como comprovam os recentes projetos de processadores VLSI, mas que tende rapidamente a envolver também o projeto dos sistemas de software associados. Duas razões despontam como fortes inibidoras desta integração: o enfoque de projeto evolucionário, dominante para sistemas de software, que inviabiliza um ciclo de projeto integrado com as demais áreas onde as metodologias baseiam-se em um enfoque descendente e; a própria inexistência de um paradígma sedimentado para o projeto arquitetural, hoje bastante influenciado pela metodologia de síntese vertical preponderante em projetos de hardware. Este trabalho identifica as principais diferenças entre as metodologias dominantes em cada área e propõe as condições básicas para o estabelecimento de um enfoque homogêneo de projeto, fator essencial à obtenção futura de um ambiente integrado de projeto.

Palavras-chave: sistemas de computação, projeto de software, projeto arquitetural de sistemas de computação, síntese vertical de hardware, co-projeto de hardware e software

1. INTRODUÇÃO

A evolução dos computadores teve como núcleo e catalisador, a arquitetura de von Neumann, em função da qual se desenvolveram e foram consolidadas as áreas de software e hardware, hoje camadas efetivas dentro da estrutura hierárquica dos sistemas de computação. O surgimento de opções à máquina de von Neumann propiciou o amadurecimento da área de arquitetura, que atualmente conta com um amplo espectro de modelos computacionais, alguns dos quais testados em protótipos de laboratório [1,2]. Entretanto, a grande maioria dos computadores tem sido concebida aplicando-se a metodologia de projeto sumariada em Gurd et al [3], que preconiza o projeto de sistemas de computação em três níveis que se complementariam: o "nível de máquina", correspondendo ao nível do conjunto de instruções; o "nível da linguagem", definido pelo escopo das lingua-

(*) Professor do Instituto de Informática da PUC-Campinas e Pesquisador da Fundação Centro Tecnológico para Informática - CTI (gonzaga@im.cti.br).

gens de programação e; o "nível da aplicação", orientado para domínios de aplicação específicos.

Este enfoque, embora adequado para a validação conceitual de soluções, não explora de forma eficiente o espaço de projeto[1] existente na atualidade [4] propiciado pela variedade de tecnologias de hardware e software disponíveis, estando sustentado primariamente no aumento de desempenho do hardware, fruto da crescente velocidade e compactação dos circuitos integrados. A demanda por maior integração por sua vez ocasionou um brutal aumento na complexidade dos circuitos[2], como resultado direto das dificuldades de empacotamento geradas pela redução do ciclo de relógio, culminando no desenvolvimento dos processadores VLSI. Com os processadores VLSI[3], formados por poucos tipos de elementos funcionais, com caminhos de dados e de controle simples e regulares e uso extensivo de "pipelining" [5,6,7], caracterizou-se a mudança no paradígma original de projeto arquitetural, com o estabelecimento de uma simbiose entre arquitetura e implementação, na forma sumariada por Hennessy [8]:

1. *Preferência pelo uso de estruturas paralelas.* "Pipelining" e arranjos paralelos de múltiplas unidades funcionais são usados para remediar a menor velocidade de operação dos circuitos MOS. A maior adequação da tecnologia para a replicação de estruturas regulares também penaliza arquiteturas formadas por módulos complexos.
2. *Minimização da comunicação.* O custo de implementação é bem maior para arquiteturas que requeiram alto grau de comunicação dos módulos internos entre si e com o exterior. Em particular busca-se concentrar o tráfego de alta velocidade nos limites da pastilha devido a largura de banda da comunicação extra-integrado ser muito menor e mais susceptível a interferências e atrasos de propagação.

A integração entre os projetos arquitetural e de hardware permite uma melhor exploração do espaço de projeto dos sistemas de computação, tendo entretanto como limitante a pouca interação com o projeto dos sistemas de software associados que acarreta efeitos negativos no desenvolvimento dos sistemas como um todo [9].[4] No item a seguir é apresentada uma panorâmica das principais metodologias de projeto adotadas para software, hardware e arquitetura e no item 3 são analisadas as condições para que estas metodologias evoluam para um cenário de maior integração.

2. METODOLOGIAS DE PROJETO

De forma geral, o projeto de sistemas de computação assistido por computador está, como resposta ao nível de complexidade atingido, concentrando-se no domínio da descrição do problema em contra-posição ao enfoque tradicional, orientado para a definição da solução, estratégia que tem garantido a tratabilidade computacional dos problemas, dentro de um cenário de complexidade crescente. Para projetos de sistemas de software isto implica em uma demanda por qualidade e produtividade com a conseqüente pesquisa em métricas e padrões. Nos projetos de hardware o objetivo é a síntese automática de circuitos integrados, feita a partir de uma descrição comportamental, e que gere soluções não inferiores [10][5] para um dado conjunto de especificações [11]. Para que seja viável, esta filosofia de projeto não prescinde do desenvolvimento de metodologias de projeto as quais encontram-se em estágios distintos de amadurecimento de acordo com a área e o nível de abstração envolvidos, como visto a seguir.

(1) Espaço euclidiano cujos eixos correspondem ao conjunto definido como métrica da qualidade do projeto.

(2) A complexidade sendo função da dimensão mínima e também do tamanho do integrado [28].

(3) O desbalanceamento inicial entre os avanços na tecnologia para fabricação de circuitos integrados e a capacidade de uso eficiente da área em silício, desembocou na chamada "crise do VLSI" nos anos 70 que levou alguns pesquisadores a preverem um estrangulamento iminente dos computadores sequenciais por conta da impossibilidade de obtenção de mais desempenho, fato que só foi superado a partir da metodologia de projeto proposta por Mead e Conway, embora continue válido o argumento de que, cedo ou tarde, tornar-se-á impossível sustentar o aumento de desempenho com base na redução constante do ciclo de relógio dos computadores, em uma dada tecnologia de integração.

(4) Estatísticas existentes para sistemas embutidos demonstram que embora 90% dos circuitos integrados para aplicações específicas sejam projetados corretamente apenas 50% do sistemas finais, compostos por hardware e software, têm a funcionalidade esperada.

(5) Uma solução não inferior é aquela que sem ser ótima, não pode entretanto ser superada por nenhuma outra desde que cada melhoramento sob o ponto de vista de algum dos critérios de sua métrica, provoque degradação ao menos em relação a algum outro critério. Em geral há um infinito número de pontos que compõem uma solução não inferior, definindo uma "superfície não inferior", dentro do espaço de projeto.

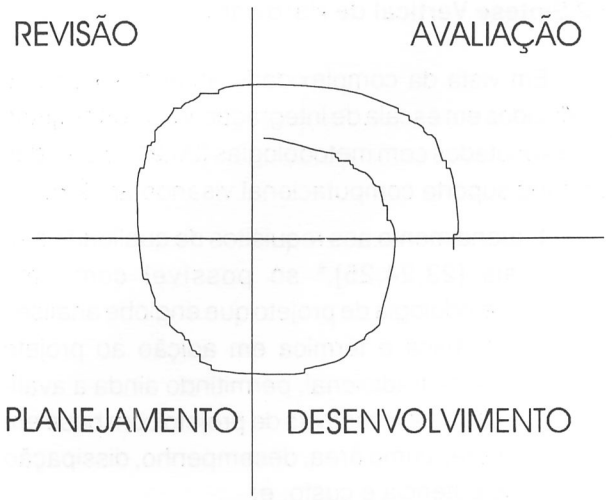

Figura 1: Metodologia de Projeto em Espiral. Proposta em 1986 por Boehm, esta metodologia se opõe ao modelo clássico de desenvolvimento em cascata, por introduzir a análise de riscos no ciclo de projeto e por romper com o clássico enfoque descendente (top-down) de desenvolvimento de software. Como visto no diagrama, Boehm modela o processo por meio de quatro etapas cíclicas: No quadrante denominado revisão, é feita a determinação de objetivos, identificação de restrições e a análise de alternativas; o quadrante de avaliação comporta a análise de riscos e interação com o usuário; o quadrante do desenvolvimento contempla a geração do nível seguinte de refinamento da solução e; o quadrante de planejamento contém as atividades de definição de novas metas. O processo de produção de software é feito de modo incremental com cada passagem pelo conjunto de quadrantes gerando uma versão mais refinada da solução, ou o abandono do projeto.

2.1 Projeto Integrado de Software

O projeto de sistemas de software encontra-se em transformação, migrando de uma forma linear de pensamento para uma visão iterativa ou evolucionária do software [12].

2.1.1 Modelos Sequenciais

O paradígma de engenharia de software dominante ainda é sequencial sendo os sistemas desenvolvidos segundo modelos de processo em que séries de passos sequenciais tratam a complexidade dos problemas manipulando-os por partes. O modelo sequencial mais antigo, conhecido como "codificação-e-correção" e usado nos primórdios da computação, é puramente empírico, com as atividades de depuração e teste sendo planejadas e executadas após a codificação ter sido feita [13]. O crescimento dos programas exigiu um planejamento prévio com a conseqüente caracterização do desenvolvimento de software na forma de um ciclo representado pelo "modelo em cascata" [14] proposto por Royce [15] em 1970, onde o ciclo de produção é dividido em estágios correspondendo às atividades de análise de viabilidade, especificação de requisitos, projeto, codificação, integração, testes e manutenção, sendo previstas realimentações entre estágios adjacentes.

2.1.2 Modelos Iterativos

Nesta categoria encontram-se os modelos baseados em linguagens de 4ª geração, prototipagem rápida e o modelo da espiral [16,13], os quais podem ser usados também de forma combinada [12]. Em qualquer dos casos, faz-se uso extensivo de ambientes integrados de projeto (I-CASE),[6] cuja arquitetura básica é vista a seguir, como forma de obtenção de ganhos em produtividade e qualidade pela exploração de macro-aspectos da metodologia de projeto como o uso de repositórios de software e prototipagem rápida [17,12].

Como já notado, a principal mudança diz respeito ao abandono da forma sequencial de projeto caracterizada pelo modelo de produção de software em cascata, substituindo-o por modelos iterativos ou evolucionários. A essência desta mudança é a troca da mecânica de projeto orientada pela especificação por outra dirigida para a identificação e prototipagem dos "elementos de risco" do sistema. Pressman [12] sumaria as tendências de mudança através de uma proposta de ambiente para desenvolvimento de software construído a partir do modelo de espiral proposto por Boehm, comentado na Figura I, e suportado por tecnologias integradas de prototipagem e orientação a objeto.

Dentro do diagrama em espiral, o desenvolvimento do projeto seria feito com base no paradígma de orientação a objeto, o qual oferece como vantagens sobre os demais, a facilidade de derivação de componentes e o re-uso de software. A etapa de avaliação, como é inerente ao modelo proposto por Boehm, faria uso extensivo dos recursos de

(6) "Integrated Computer-Aided Software Engineering".

prototipagem, seja com o objetivo experimental, exploratório ou mesmo estrutural [18].[7]

2.1.2.1 Arquitetura de um Ambiente de I-CASE

Exceto por detalhes de implementação, os ambientes de I-CASE seguem a arquitetura básica vista e comentada na figura. São essenciais para a operação destes ambientes a presença de mecanismos de execução, através dos quais é possível o disparo, suspensão ou término de processos; e de mecanismo de comunicação, responsáveis pela gerência da comunicação entre processos, permitindo o uso integrado das ferramentas. Em geral, o ambiente é organizado de forma distribuída, com base em um sistema operacional multi-tarefas sob o qual várias ferramentas podem estar ativas concomitantemente. O elemento central do ambiente é o repositório de software [19], um banco de dados relacional ou orientado a objeto ao qual estão associados serviços exclusivos para CASE, como acomodação de tipos de dados complexos, modelos para a imposição de integridade, interfaces ricas em semântica, entre outros.

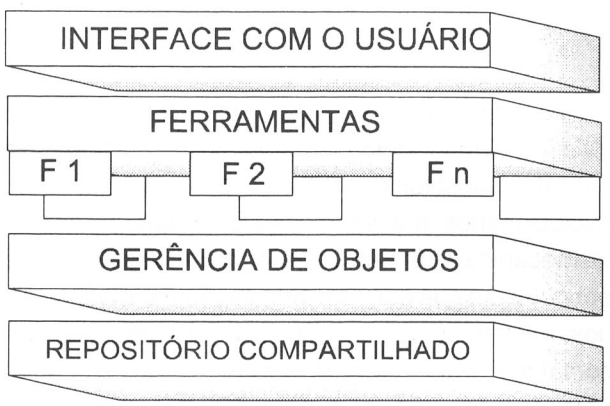

Figura 2: Arquitetura de um Ambiente de I-CASE. A estrutura em camadas é formada por ferramentas de interface homem-máquina baseadas em um protocolo de apresentação comum; por um conjunto de ferramentas de CASE suportadas por serviços de gerência de ferramentas comuns; por uma camada responsável pelas funções de gerência de configuração e; por um repositório de software contendo a base de dados do ambiente.

2.2 Síntese Vertical de Hardware

Em vista da complexidade atual dos circuitos produzidos em escala de integração VLSI, os projetos são executados com metodologias [20,21] baseadas em forte suporte computacional visando ao [22]:

1. atendimento aos requisitos de qualidade atuais [23,24,25],[8] só possível com uma metodologia de projeto que englobe análises mecânica e térmica em adição ao projeto elétrico tradicional, permitindo ainda a avaliação de alternativas de projeto com base em fatores como área, desempenho, dissipação de potência e custo, e;
2. manutenção do ciclo de projeto dentro de limites de tempo condizentes com as expectativas de um mercado altamente competitivo [20,26].[9]

A automação de projeto em hardware é a capacidade de um sistema de projeto de perfazer as atividades de síntese de uma forma autônoma e correta por construção, entre um nível do processo de projeto para outro [27,28].[10] Síntese refere-se, em termos gerais, ao processo de transformar uma descrição de projeto em um dado nível de abstração para uma representação equivalente de nível mais baixo [22], acrescentando-se detalhes estruturais e geométricos, com o objetivo de aproximar-se da realização física do sistema [29]. Os diversos níveis em que o processo de síntese é aplicado no projeto de hardware são sumariados a seguir.

2.2.1 Síntese ao Nível do Sistema

Etapa de síntese aplicável quando, usando-se a capacidade atual de integração, sistemas inteiros são acondicionados em uma única pastilha. Neste caso as técnicas de síntese funcional nem sempre são

(7) Riddle e Williams definem três tipos de prototipagem: o primeiro, chamado estrutural ou evolucionário, teria como meta a criação do software por meio de implementações de complexidade sucessiva, que agreguem funcionalidade; o segundo tipo definido como experimental teria objetivo investigatório, permitindo a comparação entre diferentes implementações; por fim o tipo exploratório seria adotado como apoio para a identificacão e formulação do problema.

(8) Ao contrário do que se poderia supor em um primeiro momento, a sistematização do projeto visando atingir níveis de qualidade profissional é foco de intensa pesquisa acadêmica, notadamente em hardware.

(9) Dado o nível de integração atual se mantidos os padrões e ferramentas do início da década de 80, o tempo de proieto seria comparável ao tempo de vida do produto com um circuito integrado de 10^5 transitores demandando 60 homens/ano para o projeto e igual quantidade para a depuração. Extrapolando-se para um integrado com 10^7 transistores o esforço, apenas no projeto, seria de $6 \cdot 10^3$ h/a.

(10) Para a maioria dos métodos de síntese tratados neste item, valeria ainda a definição mais restrita de McFarland et al: "A tarefa de síntese é, a partir da especificação do comportamento requerido de um sistema, e de um conjunto de restrições e objetivos a serem atingidos, encontrar uma estrutura que implemente este comportamento, satisfazendo ainda as restrições e os objetivos".

adequadas ao nível de abstração da descrição comportamental de entrada por não enfocar questões como por exemplo, o desmembramento do sistema em processos assíncronos, o redimensionamento do bloco operacional para variação do grau de paralelismo, e o agrupamento de registradores isolados em "register files" [27]. Este nível de síntese pode ser também usada nos casos de sistemas implementados em mais de um integrado, que sejam gerados a partir de uma única descrição precedendo, em ambos os casos, à etapa de síntese funcional. A síntese ao nível do sistema como descrita por Lagnese e Thomas [31] determina, a partir de uma descrição comportamental, e segundo as etapas descritas a seguir, as características de projeto necessárias à síntese funcional, como o número de integrados ou módulos, e o tráfego global entre estes.

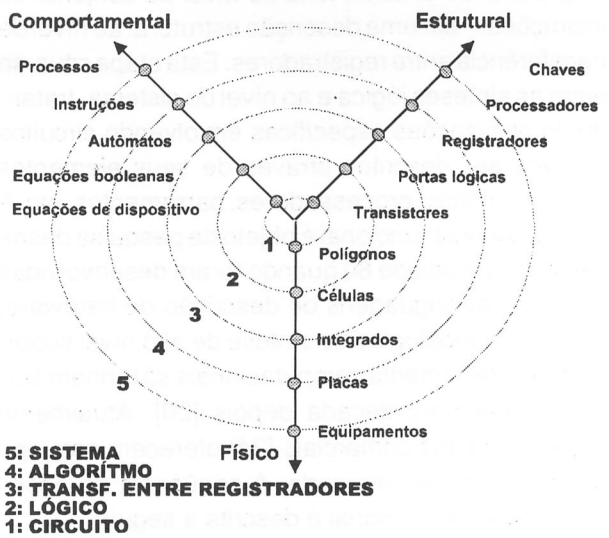

Figura 3: Diagrama de Gajski [26,1]. Em 1983 Gajski e Kuhn propuseram o esquema de representação posteriormente denominado diagrama de Gajski ou diagrama **Y**, que contempla os três eixos ortogonais usados para a representação de sistemas digitais, correspondendo aos domínios comportamental, estrutural e físico. A partir do centro, cada eixo cresce em abstração, podendo ser identificados pontos de mesmo Nível de abstração nos três eixos correspondendo a níveis de abstração definidos, embora estes permitam um "continuum" de representação [29].

2.2.1.1 Partição Arquitetural

Uma tarefa essencial que corresponde a determinação das relações de dados e de controle relevantes para a extração da estrutura implícita no comportamento do sistema, com o objetivo de:

1. otimização da área através da redução do hardware usado em cada partição;
2. aumento do desempenho pela redução do tráfego entre partições, obtido com o agrupamento das partes da arquitetura que se interagem mais pesadamente. Adicionalmente pode haver também uma otimização em área se houver redução nos dutos de comunicação;
3. detecção de paralelismo e concorrência ao nível do sistema, facilitando o escalonamento na etapa de síntese funcional e;
4. introdução de considerações físicas e estruturais precocemente no projeto, adotando assim um metodologia mista, não puramente ascendente ou descendente.

2.2.1.2 Síntese

Como nas demais etapas de síntese, a metodologia de síntese ao nível do sistema aplica métodos de otimização por critério múltiplo, neste caso baseadas na técnica de "agrupamento em múltiplos estágios".[11] As técnicas de agrupamento agregam objetos em conjuntos segundo alguma métrica de proximidade a qual, aplicada iterativamente, produz uma estrutura hierárquica em árvore para a disposição dos objetos, permitindo então que seja usado um critério de corte para a determinação dos grupos. A técnica de agrupamento em múltiplos estágios é o uso combinado das técnicas de agrupamento com a adoção de métricas de proximidade diferentes e desacopladas entre si, para facilitar sua tratabilidade computacional, e evitar a propagação de erros devido a aproximações e estimativas feitas em uma dada métrica. As métricas de agrupamento adotadas por Lagnese e Thomas são derivadas dos seguinte tipos básicos:

- *Transferência de controle.* Visa a redução da passagem de controle entre grupos, buscando agrupar-se os operadores em fluxos contínuos, de acordo com a probabilidade **P** de ativação sequencial destes operadores, como no exemplo da equação 1, onde $\mathbf{P}_{AB}$ repre-

(11) Adaptação para o português da expressão inglesa "multi-stage clustering".

senta a probabilidade de ativação sequencial dos grupos de operadores **A** e posteriormente **B**.

$P_{AB} = P_B / P_A$ (equação 1)

- *Transferência de dados*. Tem como meta reduzir o fluxo de dados entre grupos o que é feito com base na definição de grupos que tenham o valor mínimo para a razão $\mathbf{G}_{dado}$ que corresponde a fração de dados comuns entre grupos, como no exemplo da equação 2, válida para dois grupos de operadores **A** e **B**:

$$G_{DADO\_AB} = \frac{Comum(A, B)}{Total_A + Total_B} \quad \text{(equação 2)}$$

- *Redundância de Hardware*. Estágio executado após os agrupamentos por dado e controle, consistindo na redução do hardware pelo compartilhamento de partes do bloco operacional usadas por grupos de operadores similares, quais sejam, aqueles grupos que contenham os mesmos tipos de operadores e não sejam simultaneamente acionados.

Para a árvore obtida após a aplicação de cada estágio, a determinação da linha de corte faz uso de considerações físicas fornecidas adicionalmente à descrição comportamental. Os critérios de corte são baseados em considerações de área, conexões de dados e escalonamento.

- *Área*. Este critério tem como objetivo produzir agrupamentos dentro de limites máximos e mínimos de área, evitando a produção de objetos com dimensões por demais díspares, o que dificulta o leiaute automático posterior. É também usada informações do projetista relativas a área dos objetos para uma dada tecnologia-alvo.
- *Conexões*. Tem como meta a redução das interconexões entre grupos de operadores, através do que se obtém uma redução entre as interconexões das partições.[12]
- *Escalonamento*. Neste critério são introduzidas considerações de tempo, através da limitação pelo projetista do comprimento máximo das seqüências de operadores presentes em um dado grupo. Como restrição adicional é imposta a existência de apenas uma unidade funcional de cada tipo dentro da seqüência.

2.2.2 Síntese Funcional

A síntese funcional de circuitos digitais, também chamada síntese de alto nível [27,29,32,33], é a transformação automática de uma especificação comportamental em linguagem de descrição sequencial do circuito, feita ao nível do conjunto de instruções,[13] em uma descrição estrutural ao nível de transferência entre registradores. Esta etapa situa-se entre as sínteses lógica e ao nível do sistema, tratando de otimizações específicas envolvendo circuitos sequenciais, descritos através de seus elementos como memórias, processadores, barramentos, etc. A área de síntese funcional é objeto de pesquisa desde o final da década de 60 quando foram desenvolvidas as primeiras linguagens de descrição de hardware, embora metodologias de síntese de alto nível suportadas por ferramenta computacionais só tenham surgido quase uma década depois [20]. Atualmente alguns produtos comerciais [34] oferecem este tipo de ferramenta ao mercado. A seqüência de ações para a síntese funcional é descrita a seguir:

2.2.2.1 Compilação

Compilação da descrição comportamental, convertendo-a em um grafo orientado, representando o fluxo de dados e de controle.[14] Durante a compilação, além das otimizações de código comuns em software, são tomadas ações específicas orientadas para a

(12) Apenas a descrição comportamental não permite a aplicação deste critério, pois como também notado por Lagnese e Thomas [31], não há o mapeamento um-a-um entre o fluxo de dados na descrição comportamental ao nível do sistema e os dutos na implementação em hardware, o que faz este critério mais fraco que os demais.

(13) Chamado em alguns textos como nível algorítmico. A nomenclatura adotada neste texto é um pouco mais específica, e leva em conta o contexto particular do projeto de sistemas de computação, sendo consistente com as convenções encontradas na literatura especializada.

(14) As informações dados e de controle são geradas em grafos separados em alguns sistemas [32].

implementação final em hardware como aumento do paralelismo sempre que não houver dependência de dados, e redução do número de níveis no grafo.

2.2.2.1.1 Escalonamento

Escalonamento no tempo das operações descritas no grafo associando cada operação a um passo de controle, de forma que uma unidade de hardware só possa ser acionada uma vez em cada passo.[15] As ações de escalonamento e alocação convertem um comportamento em uma representação estrutural e podem ser aplicadas de forma intercambiável. Se, por exemplo, o bloco operacional for alocado primeiro, a partir deste é feito o escalonamento das operações.

2.2.2.1.2 Alocação

A alocação corresponde a atribuição de hardware para os elementos do grafo: unidades funcionais para a execução das operações, unidades de armazenamento para as variáveis, e barramentos para a comunicação.

2.2.2.1.3 Geração do Bloco de Controle

Geração da lógica de controle com base no escalonamento das operações no bloco operacional resultante da alocação.

2.2.3 Síntese Lógica

A síntese de funções lógicas tem como objetivo a implementação em hardware de uma dada rede de circuitos lógicos combinatórios, expressos originalmente através de equações booleanas, tabelas, máquinas de estado algorítmicas, ou linguagens de descrição de hardware, entre outros. Os algorítmos de minimização baseados na teoria convencional de chaveamento de circuitos, e orientados para a obtenção da solução ótima em termos do número de portas, conduzem a soluções do tipo NP-completa [35].[16] não sendo adequados para a implementação em computador tendo em vista o rápido aumento da complexidade destes algorítmos a medida que o número de variáveis de entrada cresce. A síntese lógica assistida por computador se faz necessária devido a dois fatores associados:

1. possibilidade tecnológica de integração de grande quantidade de lógica em um único circuito integrado, permitindo com isto o aumento da velocidade de operação dos circuitos e;
2. a conseqüente complexidade das fases de minimização, simulação e teste da rede lógica gerada. Deste modo, a síntese automática de um circuito correto por construção e que, mesmo sem ser o mínimo possível, atenda aos requisitos especificados apresenta-se como vantajosa [36].

Tomando-se como ponto de partida um conjunto de funções booleanas a tarefa de síntese consistiria na sua decomposição em sub-funções com um número limitado de variáveis, seguido pelo agrupamento destas através das variáveis comuns em blocos básicos, com os critérios de agrupamento visando minimizar os caminhos críticos e o número total de blocos [37,38,39], buscando assim obter soluções que apresentem atraso mínimo, área mínima ou uma combinação de ambos [11].

2.2.3.1 Síntese em Dois Níveis

As técnicas de síntese lógica são divididas em síntese de dois níveis e multi-nível. A minimização lógica em dois níveis produz uma expressão booleana formada por somas-de-produtos,[17] sendo orientada para a implementação de funções de controle em PLAs,[18] onde a minimização do número de mintermos corresponde a solução de menor área.

(15) Desta forma, a síntese funcional produz um circuito final síncrono, com cada passo de controle correspondendo a um período de relógio.

(16) Em relação ao tempo de execução, os algoritmos para a solução de um problema podem ser de "tempo exponencial", onde o tempo de execução é uma função exponencial do número de entradas; ou de "tempo polinomial", onde o tempo de execução é uma função polinomial do número de variáveis. Problemas com solução polinomial conhecida são ditos de classe **P**. Estes são um sub-conjunto dos problemas que, a princípio, têm uma solução polinomial embora só a solução exponencial seja conhecida, chamados de problemas classe **NP**. Dentro da classe NP existem alguns problemas gerais, aos quais os demais podem ser reduzidos, chamados de **NP-Completos**, e que não apresentam solução ótima.

(17) Na soma-de-produtos cada parcela da expressão representa um ponto onde a função assume o valor lógico 1.

(18) "PLA- programmed logic array". Estrutura usada para o mapeamento de lógica combinatória irregular. Apresenta nas suas saídas a forma canônica da soma-de-produtos das entradas.

2.2.3.2 Síntese Multi-Nível

A síntese lógica multi-nível por sua vez representa uma forma mais geral de solução possibilitando o re-uso de blocos de lógica comuns e permitindo resolver problemas de feixe-de-entrada e de saída [40],[19] bem como de potenciais "circuit hazards" decorrentes da solução em dois níveis [36]. A síntese lógica multi-nível é foco de intensa pesquisa na atualidade [38,41], mas em termos gerais os diversos métodos de síntese obedecem a seqüência de passos:

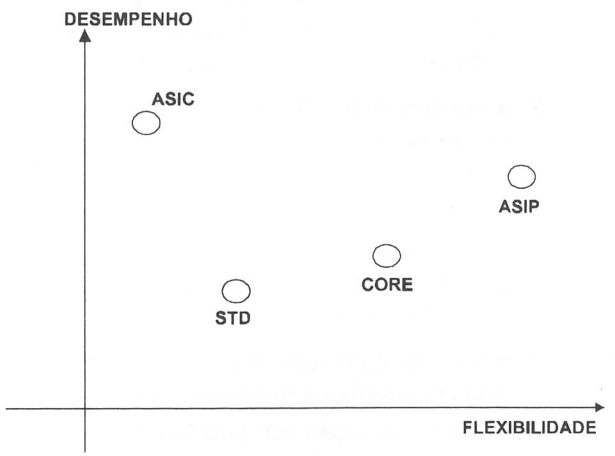

Figura 4: Espaço de projeto de processadores dentro da metodologia de co-projeto [46]. Dentro do escopo do projeto de sistemas digitais de aplicação específica, o desenvolvimento do bloco de controle varia de acordo com a opção de processamento adotada. Soluções que maximizam o desempenho as expensas do tempo de projeto e da perda de flexibilidade são implementadas com base em circuitos integrados para aplicações específicas (ASICs); Uma solução que aumenta ainda mais o tempo de projeto porém permite a implementação de computação em software, flexibilizando o uso do sistema, é obtida com o uso de processadores com conjuntos de instruções dedicados (ASIPs); redução adicional no tempo de projeto, com a consequente perda de desempenho é conseguida se for usado um núcleo de processador já existente como parte do controlador (CORE); finalmente, a solução mais flexível e de custo e ciclo de projeto menores é obtida se for usado um processador de propósito geral padrão (STD).

1. representação da função lógica como uma soma-de-produtos, como no caso da síntese de dois níveis;
2. fatoração desta expressão de onde se obtém sua representação multi-nível,[20] que não representa necessariamente a solução ótima, para o que não há algorítmo geral conhecido [11]. A expressão fatorada permite a implementação lógica da função com um menor número de transistores e conseqüentemente menor área, causando entretanto maior atraso [26] e;
3. mapeamento da função resultante na biblioteca de células da tecnologia a ser usada na implementação em hardware, visando a geração de um circuito otimizado em área e dentro das limitações de tempo de propagação especificadas.

2.3 Projeto Arquitetural

Metodologias de projeto ao nível arquitetural, englobando hardware [27,31][21] e software, são ainda fracamente suportadas por ferramentas computacionais, disponíveis atualmente apenas para sistemas com arquitetura fixa e pequena ou média complexidade [42,43,44], tipicamente usados em aplicações específicas, e projetados para exercer uma função determinada dentro de um sistema maior [45].

2.3.1 Co-Projeto Hardware-Software

A área de co-projeto hardware-software[22] lida com o desenvolvimento de metodologias unificantes de projeto automatizado para sistemas compostos por software, e a hardware programável de propósito geral e de aplicação específica, e componentes eletromecânicos, englobando também questões relativas ao empacotamento mecânico do sistema [42], visando uma maior eficiência em relação às metodologias tradicionais de projeto devido ao [46]:

1. melhor suporte ao sistema operacional e programas aplicativos, bem como ao ciclo evolutivo dos produtos, tendo em vista a metodologia integrada de projeto e;
2. melhor desempenho do sistema final pelo balanceamento da computação entre

(19) Tradução para as expressões "fan-in" e "fan-out".

(20) Ao contrário do que ocorre com a soma de produtos, o complemento da expressão fatorada é uma expressão que possui o mesmo número de literais que a função original.

(21) A faceta de hardware das questões arquiteturais no entanto, tem sido favorecida pela existência de metodologias e ferramentas, desenvolvidas originalmente para suporte ao projeto de circuitos integrados que, devido a sua alta integração, permitem hoje a inclusão de pequenos sistemas computacionais em uma só pastilha.

(22) Em inglês "hardware-software codesign".

módulos de hardware programável e de software. Os enfoques básicos são a síntese de hardware programável dedicado para a aceleração da execução de funções,[23] e a transferência de funções não críticas para o software, visando a redução de custos e aumento da ftexibilidade.

2.3.1.1 Metodologia de Projeto

A tarefa inicial em um co-projeto é a obtenção de uma arquitetura para o sistema dado um conjunto inicial de especificações, que discrimine os elementos de hardware eletrônico e mecânico, e de software, sobre os quais são então aplicadas metodologias específicas de projeto, geralmente realimentadas de algum modo que permita a consistência final frente às especificações originais. A definição da arquitetura envolve dois aspectos [42] distintos e sequenciais:

1. escolha do modelo computacional ao nível arquitetural, a partir do qual os elementos de hardware e software serão organizados e;
2. criação de uma instância específica do modelo computacional que atenda aos requisitos de projeto.

Devido a forte inter-relação entre as metodologias de co-projeto e de síntese funcional, as pesquisas atuais têm se concentrado no universo dos sistemas reativos,[24] particularmente em "embedded systems",[25] onde a geração de sistemas baseados em um controlador síncrono comandando um bloco operacional, organizados em uma arquitetura pré-definida, produz resultados eficientes. A opção "a priori" por uma arquitetura-alvo praticamente suprime considerações referentes ao item I, restringindo o espaço de projeto a decisões quanto a natureza dos elementos processadores como visto na figura 4.

2.3.1.2 Aplicações

Várias propostas de metodologia de projeto para a área de co-projeto têm sido apresentadas. Gupta e De Micheli [44] propõem uma metodologia para síntese de sistemas compostos por microprocessadores e ASICs orientada a uma implementação em barramento comum, a partir de uma especificação comportamental a qual são aplicadas restrições de tempo, o que também é feito por Thomas et al [43], os quais no entanto se concentram na definição de mecanismos de sincronização. Um ambiente orientado à simulação para aplicações em processamento digital de sinais que admite a convivência de ferramentas heterogêneas é proposto por Kalavade e Lee [47]. Srivas-tava e Brodersen [42,48] apresentam uma metodologia mais abrangente, para a síntese de sistemas em uma organização de barramentos hierárquicos, suportada por um mecanismo próprio de comunicação entre processos.

Tabela 1: Panorâmica das metodologias para o projeto de sistemas de computação assistido por computador.

CAMADA	METODOLOGIA	APLICAÇÃO	ENFOQUE DE PROJETO
SOFTWARE	método da espiral	gerência de projeto	iterativo
	prototipagem	projeto de software	iterativo
	proj. orientado a objeto	projeto de software	misto
ARQUITETURA	co-projeto	hardware & software	descendente
	síntese de sistemas	projeto de sistemas/VLSI	descendente
HARDWARE	síntese funcional	projeto VLSI	descendente
	síntese lógica	projeto VLSI/LSI	descendente
	projeto físico	leiaute de microeletrônica	descendente

(23) Um exemplo é o uso de co-processadores baseados em "programmable active memories-PAMs", estas formadas por um conjunto de "fuse programmable gate arrays-FPGAs" e memórias RAM, os quais são carregados com porções críticas do software compiladas para execução em hardware.

(24) Um sistema reativo executa funções em resposta a estímulos externos de entrada, dentro de janelas específicas de tempo.

(25) "Embedded Systems" podem ser classificados como sistemas digitais para aplicação específica cujo projeto é sujeito a restrições de tempo associados a eventos assíncronos [44].

Como deficiência comum em todos os trabalhos, a partição entre hardware e software é sempre feita manualmente o que, adicionado ao uso de arquiteturas fixas, limita bastante o escopo dos ambientes de co-projeto.

3. INTEGRAÇÃO ENTRE AS METODOLOGIAS DE PROJETO

A complexidade dos projetos é enfrentada atualmente por duas linhas básicas de ação [26,30], que se distinguem quanto ao papel designado para as ferramentas computacionais:

1. a primeira linha postula que o projetista é o centro da atividade de projeto, cabendo às ferramentas uma função auxiliar, incrementando a produtividade do elemento humano através do suporte a captura de projeto, análise, verificação, etc;
2. a segunda linha de ação preconiza a possibilidade de codificação do conhecimento humano em termos de algorítmos ou regras, permitindo que o processo de projeto seja totalmente automatizado, por meio de atividades de síntese e compilação.

Em linhas gerais, metodologias de projeto evolutivas relacionam-se com o ambiente de projeto na forma descrita em 1, enquanto que metodologias descendentes[26] são baseadas em ambientes como o descrito em 2, conforme sumariado na Tabela 1. Para que haja uma melhor exploração do espaço de projeto, através da maior integração entre as diferentes metodologias existentes para software, arquitetura e hardware, três pontos são fundamentais:

Foco no domínio da descrição do problema. Como visto em 2.1, 2.2 e 2.3.1, as metodologias de projeto aplicadas aos sistemas de computação, antes confinadas ao "domínio da solução", evoluíram para a situação sumariada na tabela, onde a ênfase é concentrada no "domínio do problema" com o projeto sendo orientado pela descrição deste, tendo em vista a complexidade crescente dos sistemas, esta uma função de dois fatores básicos:

1. Uso de estruturas elementares em hardware. Apesar da complexidade apresentada pelos circuitos VLSI, estes ainda são concebidos, em sua forma final, em termos de transistores [49,50], os mesmos elementos básicos no projeto eletrônico há quase 50 anos. As metodologias de síntese de hardware, embora trabalhem em níveis de abstração que permitem a manipulação de módulos e blocos funcionais que muitas vezes independem da implementação, organizam fisicamente estes blocos em estruturas baseadas em arranjos bidimensionais de transistores, do que resulta uma solução complexa ao se fazer o mapeamento entre funcionalidade e implementação.
2. Linguagem de montagem composta por operações com baixo significado semântico. Com o fracasso das arquiteturas de processadores HLLCA [51,52][27] e a subseqüente popularização dos processadores RISC, baseados em conjuntos de instrução simplificados e com baixo significado semântico associado, ocorre nos sistemas de software fenômeno análogo ao descrito no item 1 para os circuitos VLSI, com o aumento crescente da complexidade do software.

Uso de metodologias de projeto evolutivas. O enfoque dominante em projetos de hardware (e conseqüentemente em arquitetura) é hoje marcadamente descendente, dificultando em muito a interação com a instância de software associada ao sistema, normalmente projetada segundo uma metodologia iterativa, e reduzindo a própria qualidade do projeto de hardware como argumentado McFarland e Kowalski [53]. Ambientes para o projeto evolutivo de

(26) Tradução para "top-down" segundo Wagner et al [29].

(27) Uma derivação em relação às máquinas CISC foi proposta ao final dos anos 70 quando alguns projetistas desenvolveram arquiteturas voltadas diretamente para a programação em alto nível, genericamente classificadas como "Arquiteturas de Computador Orientadas às Linguagens de Alto Nível", em inglês HLLCA-High Level Language Computer Architecture, que buscavam mapear os algorítmos diretamente na arquitetura da máquina [52] por meio da extensão do uso dos operadores existentes nos conjuntos de instruções também sobre as estruturas complexas manipuladas ao nível das linguagens de alto nível ou; pela introdução das estruturas de dados na arquitetura e uso de operações simples em sua manipulação. Sob o ponto de vista conceitual as máquinas HLLCA, como o intel iAPX 432, apresentam apenas diferenças de grau em relação às máquinas CISC existentes. Entretanto, as arquiteturas e implementações resultantes da aplicação de seus postulados apresentavam uma relação custo-desempenho altamente desvantajosa, o que terminou por determinar seu uso comercial apenas de forma residual.

hardware, voltados a nichos específicos de aplicação [54,55,56] são objeto de pesquisa na atualidade, o que pode servir para popularizar o enfoque iterativo no projeto de hardware.

Exploração do espaço de projeto ao nível arquitetural. O atrelamento do projeto arquitetural às metodologias de projeto de hardware, limita artificialmente o espaço de projeto ao uso de arquiteturas [57] baseadas na Máquina de von Neumann em sua versão original ou mesmo em derivações multiprocessadas.[28] No caso de aplicação das técnicas de síntese funcional a restrição é ainda maior, pois os circuitos sintetizados são formados por um bloco operacional ("datapath") comandado por um controlador síncrono [32]. Este cenário, embora indesejado, não apresenta perspectivas de alteração a curto prazo tendo em vista a inexistência de um paradígma independente de projeto arquitetural de escopo geral, que seja suportado por ferramentas computacionais.

4. CONCLUSÃO

Este trabalho apresentou uma visão panorâmica suscinta das principais metodologias de projeto aplicadas aos sistemas de computação procurando identificar suas características-chave e limitações, e propondo as pré-condições a serem satisfeitas, sob o ponto de vista do autor, para uma efetiva integração destas metodologias em um ambiente de projeto homogêneo e iterativo.

5. REFERÊNCIAS

[1] John Gurd, Ian Watson & John Glauert, *A Multilayered Data Flow Computer Architecture.* University of Manchester, Department of Computer Science, (March, 1981)

[2] Arvind, Kim Gostelow & Wil Plouffe, *An Assinchronous Programming Language and Computing Machine* University of California, Irvine, Department of Computer Science, (December, 1978).

[3] John Gurd, Wim Bohm & Yong Teo, *Performance Issues in Dataflow Machines* (Elsevier Science Publishers B. V., 1987), 285-297.

[4] Daniel Gasjki & Jih-Kwon Peir, "Essential Issues in Multiprocessor Systems," *IEEE Computer* (June, 1985): 9-27.

[5] Philip Treleaven, "Fifth Generation Computer Architecture Analysis," em *Fifth Generation Computer Systems* (JIPDEC, North Holland Publishing Company, 1982), 265-275.

[6] Wayne Wolf, "Hardware-Software Codesign," *IEEE Design & Test of Computers* (September, 1993)

[7] Carver Mead & Lynn Conway, *Introduction to VLSI Systems* (Addison Wesley, 1980)

[8] "VLSI Processor Architecture," *IEEE Transactions on Computers* C-33/12 (December, 1984): 1221-1246.

[9] Asaware Kalavade, *System Level Codesign of Mixed Hardware-Software Systems (PhD Thesis)* University of California, (Berkeley, 1996)

[10] Stephen Director et al, "A Design Methodology and Computer Aids for Digital VLSI Systems," *IEEE Transactions on Circuits and Systems* CAS-28/7 (1981): 634-644.

[11] R. Brayton et al, "Multilevel Logic Synthesis, "*Proceedings of the IEEE* (February, 1990): 264-300.

[12] Roger Pressman, *Soflware Engineering: A Practitioner's Approach.* (McGraw-Hill, 1994).

[13] Barry Boehm, "A Spiral Model for Software Development and Enhancement," *IEEE Computer* 21 /5 (May, 1988): 61-72.

[14] Alan Davis, "A Strategy for Comparing Alternative Software Development Life Cycle Models,"*IEEE Transactions on Software Engineering* (October, 1988)

[15] W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of WESCON* (August, 1970)

[16] Barry Boehm, "Improving Software Productivity," *IEEE Computer* (September, 1987)

[17] A. Sage & J. Palmer, *Software Engineering* (John Wiley and Sons, 1990)

[18] W. Riddle & G. Williams, "Software Environments Workshop Report," *ACM SIGSOFT Software Engineering Notes* (1986)

(28) Para uma visão detalhada dos modelos computacionais ao nível arquitetural veja-se Treleaven et al [57].

[19] G. Forte, "Rally round the Repository," *CASE Outlook* (1989)

[20] Stephen Trimberger et al, "A Structured design methodology and associated software tools," *IEEE Transactions on Circuits and Systems* CAS-28/7 (July, 1981): 61 R-633.

[21] Jun Gu & Kent Smith, "A strucutred approach for VLSI circuit design," *IEEE Computer* (November, 1989)

[22] R. Cavin & J. Hilbert, "Design of integrated circuits: directions and challenges.," *Proceedings of the IEEE* (February, 1990)

[23] G. De Micheli, "High-Level Synthesis of digital Circuits," *IEE Design & Test of Computers* (October, 1990)

[24] G. De Micheli, "Hardware-softtware codesign," *IEE Micro* (August, 1994)

[25] K. Mueller-Glaser & J. Bortolazzi, "An approach to computer-aided specification," *IEEE Journal of Solid-State Circuits* (April, 1990)

[26] N. Dutt & D. Gajski, "Design synthesis and silicon compilation," *IEEE Design & Test of Computers* (December, 1990)

[27] M. McFarland et al, "The High Level synthesis of digital Systems," *IEEE Journal of Solid-State circuits* 25/2 (1990)

[28] R. Burger, "The impact of ICs on computer technology," *IEEE Computer* (October, 1984)

[29] Flávio Wagner et al, *Métodos de Validação de Sistemas Digitais* (VI Escola Brasileira de Computação,1988)

[30] D. Gajski & Robert Kuhn, "New VLSI Tools", *IEEE Computer* (June 1985)

[31] Elisabeth Lagnese & Donald Thomas. "Architectural Partitioning for System Level Synthesis of Integrated Circuits", *IEEE Transactions on Computer-Aided Design*, (July, 1991)

[32] Raul Camposano. "From Behaviour to Strucutre: High-Level Synthesis", *IEEE Design & Test of Computers*, (October, 1990)

[33] Cheng-Tsung Hwang et al. "A formal approach to the scheduling problem in high level synthesis", *IEEE Transactions on Computer-Aided Design*, (April, 1991)

[34] Synopsys. *High Level Design Training Workbook*, 1996

[35] Douglas Lewin. *Computer Aided Design for Microcomputer Systems* in: *Microcomputer System Design: an Advanced Course*, (Springer-Verlag, 1982)

[36] Douglas Lewin. *Computer Aided Design of Digital Systems*, (Crane, Russak & company, Inc., 1977)

[37] Peirre Abouzeid et al. "Input-driven partitioning methods and application to synthesis on table-lookup-based FPGAs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (July, 1993)

[38] L. Bouchet & G. Saucier, "Multi-level Synthesis on PALs and on PAGs", in: *IFIP Working Conf. Logic Architecture Synthesis*, (May, 1990)

[39] R. Munoz & C. Stroud, "Automatic partitioning of programmable logic devices", *VLSI System Design*, (October, 1987)

[40] J. Millman & C. Halkias. *Eletrônica*, (McGraw-Hill do Brasil, 1981)

[41] Yen-Chen Wei et al, "Multiple-level partitioning: na application to the very large-scale hardware simulator", *IEEE Journal of solid State Circuits*, (May, 1991)

[42] Mani Srivastava & Robert Brodersen, "SIERRA: A unifiede ftamework for rapid-prototyping of system level hardware and software", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (June, 1995)

[43] D. Thomas et al, "A model and methodology for hardware-software cod esign", *IEEE Design & Test of Computers*, (September, 1993)

[44] R. Gupta & G. De Micheli, "Hardware-Software Cosynthesis for Digital Systems", *IEEE Design & Test of Computers*, (September, 1993)

[45] Frank Vahid et al, "A VHDL ftont-end for embedded systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (June,1995)

[46] G. De Micheli, "Computer-Aided Hardware-Software Codesign", *IEEE Micro*, (August, 1994)

[47] A. Kalavade & E. Lee, "A hardware-software codesign methodology for DSP applications",

IEEE Design & Test of Computers, (September, 1993)

[48] M. Srivastava & R. Brodersen, "Using VHDL for high-level, mixed-mode system simulation. *IEEE Design & Test of Computers*, (September, 1992)

[49] Stan Hurst, "Custom microelectronics technologies and developments: a survey", *Journal of Semicustom ICs*, (March 1990)

[50] Randall Geiger et al, *VLSI Design Techniques for Analog and Digital Circuits*, (McGraw-Hill, 1990).

[51] G. Meyers, *Advances in Computer Architecture*, (J. Wiley and Sons, 1982)

[52] J. Browne, "Understanding execution behaviour of software systems", *IEEE Computer*, (July 1984)

[53] M. McFarland & T. Kowalski, "Incorporating; bottom-up dcsil,m into hardware synthesis," *IEEE Transactions on Computer-Aided Design* 9/9 (September, 1990): 938-950.

[54] V. Madisetti & J. Debardelaben, "A RASSP Approach to Hardware/Software Codesign," *The HASSP Digest* 2/4 (1995)

[55] V. Madisetti, "Vive la Defference," *The RASSP Digest* 2/4 (1995)

[56] D. Gajski et al, *A Design Methodology and Environment for Interactive Behavioural Synthesis* Dept. of Information and Computer Science, University of California, Irvine, (June, 1996).

[57] Philip Treleaven et al, "Data-Driven and Demand-Driven Computer Architecture," *ACM Comp. Surveys* (vtarch, 1982).