

Métodos e Ferramentas para o Projeto de Sistemas Digitais

Ney Laert Vilar Calazans

Instituto de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Fone: (051) 3391511 Ramal 3211 - Fax: (051) 3391564
Porto Alegre - RS - Brazil
e-mail: calazans@music.pucrs.br

1 Introdução

Antes do final do primeiro semestre de 1995, estarão disponíveis para comercialização no mercado mundial amostras de pelo menos quatro microprocessadores de 64 bits. Estes circuitos integrados (CIs) possuem entre 3,8 e 9,3 milhões de transistores. Além disto, trabalham com uma frequência de relógio máxima entre 133 e 300 MHz e executam instruções a uma taxa máxima que varia de 532 a 1200MIPS (Milhões de Instruções Por Segundo) [GEP 95]. Tais dados são impressionantes, especialmente se atentarmos para o fato de que a tecnologia de integração de circuitos surgiu em meados da década de 60, e que os primeiros microprocessadores não continham mais do que algumas centenas de transistores. Eles trabalhavam com frequência de relógio não superior a 1MHz, alcançavam uma taxa máxima de execução de instruções que era uma fração de MIPS e somente estavam disponíveis no mercado a partir de 1974 [DeM 94]. Figuras de mérito similares são encontradas ao estudarmos a evolução do estado da arte em CIs de memória e em CIs específicos para uma dada aplicação, no mesmo período de tempo.

A mera existência dos microprocessadores de 64 bits nos coloca diante da questão de como é possível viabilizar a construção de sistemas digitais de tal porte. A dúvida é a mesma, caso imaginemos o sistema digital distribuído em um conjunto de componentes dispostos sobre uma ou mais placas de circuito impresso. Mesmo se abstrairmos do fato de se tratar de um sistema eletrônico e pensarmos em construir um sistema cuja complexidade exija a interconexão de cerca de dez milhões de componentes, a tarefa se revela difícil de ser apreendida globalmente. A resposta à questão não é nem simples, nem única.

Visando entender o problema, podemos inicialmente particionar o conjunto de tarefas necessárias para a construção de um sistema digital complexo em dois blocos. Em um dos blocos agrupamos todas as tarefas que envolvem manipulação direta de elementos do mundo físico, tais como a difusão de impurezas em regiões de um substrato semicondutor para a criação de transistores, ou a solda de pinos de CIs em determinadas posições de uma placa de circuito impresso. Denominamos este bloco de tarefas de processo de fabricação¹. No segundo bloco, agrupamos todas as tarefas de manipulação de modelos abstratos capazes de representar o sistema que queremos implementar, tais como a simulação de um diagrama de interconexão de portas lógicas, ou a minimização de equações Booleanas que representem a funcionalidade de um módulo do sistema. Chamamos este bloco de processo de projeto.

¹ O termo fabricação compreende aqui não somente a construção de CIs, mas também seu teste, encapsulamento, a montagem de placas, etc.

O processo de projeto de sistemas digitais obviamente precede o processo de fabricação. Este último pode ser visto como um algoritmo¹ cuja entrada é o resultado do processo de projeto, e cuja saída é o sistema digital desejado. Como tal, o processo de fabricação é passível de ser realizado de forma prioritariamente automática. Este processo estabelece regras que definem o que pode ou não ser obtido pela sua aplicação, estas regras constituindo-se então em um conjunto de restrições a serem obedecidas durante a execução do processo de projeto. A execução do processo de projeto, por outro lado, implica a realização de tarefas com um alto teor de criatividade, além de exigir um profundo conhecimento do processo de fabricação subjacente, suas limitações e suas potencialidades. Sua automatização é portanto um desafio.

O objetivo deste trabalho é prover uma visão do processo de projeto de sistemas digitais complexos, capacitando o leitor a compreender os mecanismos através dos quais a complexidade dos sistemas implementados hoje pode ser dominada. Estes mecanismos dependem primordialmente de um alto grau de automatização das tarefas de projeto. A ênfase dada consiste em salientar a diferença entre o papel dos métodos e ferramentas computacionais associados ao projeto de sistemas digitais e o papel do projetista humano que os emprega. Adicionalmente, apresenta-se uma revisão bibliográfica sucinta e atualizada do tema, destinada a iniciar os interessados. A Seção 2 apresenta a proposta de um modelo para representar o processo de projeto de sistemas digitais, usando-o em seguida para discutir a evolução do papel das ferramentas de projeto de sistemas digitais, na Seção 3. Em seguida, a Seção 4 discute brevemente o estado da arte em métodos e ferramentas computacionais associados ao processo de projeto. Na Seção 5, apresentam-se perspectivas de desenvolvimentos futuros para a área e um conjunto de conclusões a respeito do projeto atual de sistemas digitais.

2 O Processo de Projeto de Sistemas Digitais

2.1 Definição

O processo de projeto pode ser definido de forma simples como a transformação de uma descrição inicial do sistema, freqüentemente denominada especificação, em uma descrição final, também chamada de projeto final ou projeto detalhado². A diferença fundamental entre a descrição inicial e a descrição final está no fato desta última conter todas as informações necessárias à fabricação do sistema, ao contrário da primeira. Quando se trata de projetar sistemas complexos, a transformação dificilmente pode ser feita de maneira direta. Nestes casos, a atividade de projeto produz um *continuum* de descrições obtidas pela sucessiva agregação de informações à descrição inicial, culminando com a descrição final.

Cabe salientar que, em geral, a descrição final que atende aos requisitos da descrição inicial não é única. Na maioria esmagadora dos casos é inviável enumerar todos os caminhos a seguir, mesmo que em alguns casos não existam infinitas possibilidades. Como conseqüência, o projeto dificilmente se realiza de forma linear, pois a cada transformação, decisões devem ser tomadas, sem que se saiba se

¹ Um algoritmo aqui é visto de maneira informal, como um conjunto de passos previamente determinados que pode ser aplicado de maneira mecânica.

² A natureza da informação contida numa especificação não difere fundamentalmente da natureza da informação contida no projeto detalhado, visto que ambas descrevem o sistema que se deseja fabricar. Apenas muda a quantidade de informação (grau de abstração) e a precisão destas. Assim, preferimos aqui empregar o termo genérico descrição.

estas são as que conduzem à melhor das descrições finais que satisfazem a descrição inicial. Além do mais, a cada transformação podem ser introduzidos erros, violados requisitos da descrição inicial, ou inviabilizada a obtenção da melhor solução possível para o processo de projeto.

O processo de projeto pode ser então dividido em dois tipos de atividades, as de síntese, onde se agrega informação a uma descrição, produzindo uma descrição mais próxima da descrição final, e as de análise, onde se verifica o acerto das tomadas de decisão durante uma etapa de síntese.

Dois conceitos fundamentais a serem considerados durante o processo de projeto são os de correção e otimização do sistema a construir. Uma descrição final é correta se ela atende a todos os requisitos da descrição inicial e pode ser fabricada. Uma descrição final é ótima se ela é correta e possui custo mais baixo e melhor desempenho que qualquer outra solução correta. O processo de projeto deve ser guiado, obviamente, pela busca da ou de uma das soluções ótimas. Contudo, na maior parte dos casos, a solução ótima não existe, e algum compromisso entre custo e desempenho deve ser obtido.

Os critérios de otimalidade para sistemas digitais podem ser resumidos em:

- espaço - o sistema digital deve ser o menor possível;
- tempo - o sistema digital deve ser o mais rápido possível;
- energia - o sistema digital deve consumir o mínimo de energia por unidade de tempo (potência mínima).

Como estes critérios são repetidas vezes conflitantes entre si, a cada projeto deve-se favorecer aqueles que concorram mais facilmente à satisfação dos requisitos particulares da descrição inicial. Por exemplo, durante o projeto de um telefone celular deve-se favorecer os critérios de espaço e energia, ficando o de tempo em segundo plano. Por outro lado, o projeto de um super-computador deve favorecer em primeiro lugar o critério tempo, ficando os outros em um plano secundário.

Projeto é otimização. Este é um fato sabido. Um bom projetista é aquele capaz de elaborar uma descrição detalhada a partir de uma descrição abstrata, de tal forma que a funcionalidade desejada seja aquela do sistema final e que este sistema tenha custo mínimo e desempenho máximo.

2.2 Um Modelo para o Processo de Projeto - o Diagrama Y

Apresentamos nesta Seção um modelo voltado para a representação do processo de projeto de sistemas digitais, proposto em [GAJ 83].

Vimos na Seção 2.1 que o processo de projeto deve ser decomposto em um conjunto de passos, com o objetivo de reduzir a complexidade de um tratamento global. Tal decomposição é feita de maneira hierárquica, baseado na quantidade de informação contida nas descrições associadas a cada passo. Cada nível da chamada hierarquia de projeto é denominado de nível de abstração. Descrições encontram-se em um mesmo nível de abstração se contêm uma mesma quantidade de informação, ainda que representadas sob formas distintas. Um exemplo de descrições no mesmo nível de abstração seria um diagrama de portas lógicas e um conjunto de equações Booleanas. Um contra-exemplo seria uma descrição da arquitetura de um microprocessador como vista pelo programador de linguagem de montagem e o “layout” do mesmo microprocessador entregue para o processo de

fabricação, já que a primeira é muito mais abstrata que a segunda. O grau de abstração varia de maneira inversamente proporcional à quantidade de informação agregada. Assim, descrições com alto nível de abstração possuem pouca informação associada. Em um dado projeto, o mais alto nível de abstração está associado à descrição inicial, e o mais baixo à descrição final. Na prática, descrições de um sistema digital podem ser classificadas em quatro ou cinco níveis de abstração distintos. Neste artigo, empregamos uma hierarquia de quatro níveis que são, em ordem ascendente de abstração:

- nível elétrico - onde os modelos dos componentes provêm da teoria de circuitos elétricos e eletrônicos, e/ou da física de semicondutores, i. e. transistores, resistores, capacitores, equações diferenciais, diagramas elétricos, etc;
- nível lógico - onde o sistema digital é descrito a partir de noções elementares da teoria de circuitos digitais: portas lógicas, flip-flops, equações Booleanas, tabelas e grafos de transição, etc;
- nível arquitetural - onde os modelos de base são os componentes mais complexos da teoria de circuitos digitais e as formas de descrevê-los: registradores, decodificadores, ULAs, multiplexadores, linguagens de transferência entre registradores, grafos de fluxo de dados e de controle, etc;
- nível sistêmico- aqui, o sistema digital é descrito como um conjunto de algoritmos e módulos capazes de executar aqueles, que podem ou não ter um mapeamento direto para um “hardware” existente: processadores, memórias, componentes abstratos implementados no “software” básico do sistema, tais como um pacote de ponto flutuante em um sistema sem coprocessador aritmético, ou monitores de sistemas operacionais;

Além de decompor o processo de projeto segundo a quantidade de abstração das descrições, pode-se classificar as descrições segundo o tipo de informação que elas carregam. Um diagrama de tempos que descreve o comportamento de um dado sistema contém uma informação de tipo distinto do de um diagrama de portas lógicas, ainda que ambos impliquem um mesmo comportamento. Define-se assim o conceito de visões, também denominados de domínios de descrição, onde se entende por domínio o tipo de informação contida em uma dada descrição de sistema digital. Podemos identificar três domínios de descrição para sistemas digitais:

- físico - contém informação geométrica sobre os componentes/módulos e/ou sobre a disposição espacial destes no sistema a ser fabricado;
- estrutural - contém informação sobre como interconectar blocos de base de comportamento conhecido para realizar determinada funcionalidade, sem se ocupar com a disposição física destes no sistema;
- comportamental - carrega informação sobre o comportamento do sistema, sem se envolver em como tal comportamento pode ser obtido, seja do ponto de vista físico, seja do ponto de vista estrutural;

A combinação de níveis de abstração e domínios de descrição provê um modelo ao mesmo tempo simples e poderoso para interpretar o processo de projeto de sistemas digitais. Uma forma de

representação de tal modelo bidimensional foi proposta pela primeira vez por Gajski e Kuhn em [GAJ 83], o chamado Diagrama Y, uma amostra do qual aparece na Figura 1. Neste diagrama, círculos concêntricos correspondem a níveis de abstração, enquanto que segmentos de reta radiais correspondem a domínios de descrição. Cada intersecção de um círculo com um segmento radial representa um tipo de descrição distinta do sistema digital. Por exemplo, um diagrama de esquemáticos de portas lógicas TTL é uma descrição estrutural lógica, estando portanto localizado na intersecção do eixo rotulado Domínio Estrutural com o círculo representando o nível lógico de abstração.

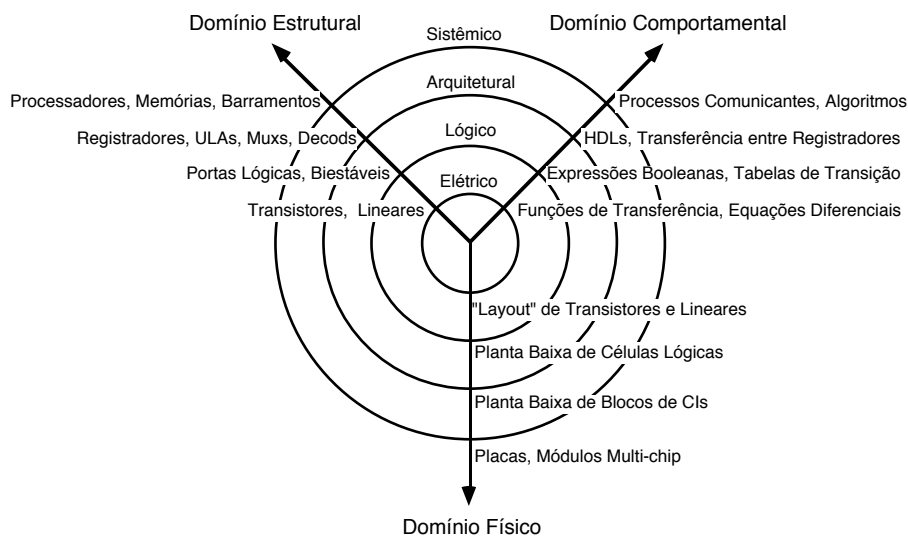


Figura 1 - O Diagrama de Gajski-Kuhn, ou Diagrama Y

Dado o Diagrama Y, como se representa um processo de projeto através dele? Considerando a definição da Seção 2.1, ilustramos o processo como um grafo dirigido sobreposta ao diagrama Y, onde o conjunto de vértices é o conjunto de descrições de projeto (correspondendo ao conjunto de pontos determinado pelas intersecções de círculos com segmentos de reta). O conjunto de arestas é o conjunto de transformações aplicadas sobre estas descrições, ou seja, arestas são associadas a métodos e/ou a ferramentas empregados pelo projetista. Estas arestas conectam descrições de entrada de um método/ferramenta a descrições de saída geradas por este. Exemplos de processo de projeto descrito mediante o diagrama Y serão vistos na Seção 3.

Alguns comentários sobre o diagrama Y ajudam a compreender sua utilidade e emprego. Primeiro, o centro do diagrama corresponde conceitualmente à descrição final, ou seja, àquela que contém toda a informação necessária à fabricação do sistema digital. Ferramentas de projeto executam transformações em descrições, identificando-se então com as arestas do grafo que descreve o processo de projeto. Como ferramentas podem manipular descrições de entrada distintas e gerar descrições de saídas distintas, podemos eventualmente associá-las a um conjunto de arestas. Por outro lado, ferramentas podem gerar uma sucessão de descrições de saída. Assim, no caso mais geral, ferramentas correspondem a conjuntos de caminhos no grafo do processo de projeto. O tipo de arestas presentes no diagrama Y nos permite classificar as ferramentas de acordo com o tipo de transformação que estas realizam sobre suas descrições de entrada. Por exemplo, ferramentas de análise produzem uma descrição de saída que não está mais próxima do centro do diagrama que a descrição de entrada. Ferramentas de síntese no entanto, produzem descrições de saída que não estão

mais afastadas do centro do diagrama que a descrição de entrada. Ferramentas de otimização correspondem a arestas com origem e destino no mesmo vértice (“self-loops”) do grafo do processo de projeto. O processo de projeto ideal seria linear, e lembraria um caminho em espiral, partindo da descrição comportamental de mais alto nível de abstração até o centro do diagrama, visitando todas as descrições de projeto possíveis. Neste caso ideal, a quantidade de informação agregada ao projeto cresceria monotonicamente do início ao fim do processo.

Cabe neste ponto uma derradeira observação sobre o modelo apresentado. Os níveis de abstração e as visões aqui sugeridos não devem ser encarados como rígidos. A complexidade do processo de projeto causa a existência uma multiplicidade de descrições que um esquema rígido de posicionar cada descrição em um nível e em uma visão bem definidos seria alta demais (níveis e visões se multiplicariam excessivamente), toldando a utilidade do modelo. Por exemplo, onde posicionaríamos um esquemático de chaves [HAY 87] [BRY 81] descrevendo um sistema digital? Trata-se de uma descrição claramente estrutural, mas os componentes desta descrição são transistores, encarados contudo como chaves, com um comportamento eminentemente lógico. Em nossa representação, posicionamos tal esquemático sobre o eixo estrutural e entre os níveis lógico e elétrico de abstração. Um outro exemplo seria uma descrição em VHDL [IEE 88] [LIP 89], uma linguagem para descrever arquiteturas de sistemas digitais contendo modelos comportamentais e estruturais, que podem ser misturados em uma mesma descrição. Aqui, o nível de abstração está bem especificado, mas a visão é um misto de estrutura e comportamento. Posicionamos tal descrição sobre o nível de abstração arquitetural mas entre os domínios estrutural e comportamental do diagrama Y. Mais importante ainda, alguns autores e. g. [JAC 93] apontam como uma deficiência do diagrama Y o fato deste não representar explicitamente a visão temporal de sistemas digitais, ao colocar sobre o eixo comportamental descrições de natureza tão dissimilar como a saída de um simulador lógico (um digrama de tempos) e tabelas-verdade de circuitos combinacionais. Este autores freqüentemente sugerem a adição ao diagrama Y de um eixo associado à visão temporal de sistemas digitais, dada a importância deste aspecto do processo de projeto. Várias outras modificações, acréscimos e correções têm sido sugeridas no modelo, mas uma discussão detalhada destas foge ao escopo do presente trabalho.

Independente de suas eventuais deficiências, o diagrama Y é um modelo que permite a visualização do processo de projeto de sistemas digitais, bem como a apreensão de estilos de projeto particulares, dados um conjunto de ferramentas e um fluxo de seu emprego para a execução do projeto.

3 Evolução das Ferramentas de Projeto de Sistemas Digitais

A fabricação de sistemas digitais complexos envolve o emprego de várias tecnologias, por exemplo as tecnologias de fabricação de CIs (CMOS, bipolar, SOI, etc), de encapsulamento de CIs, de fabricação de placas, de teste de CIs, placas e subsistemas, etc. Além disto, a fabricação emprega metodologias de implementação específicas, tais como circuitos “custom”, “semi-custom” (pré-caracterizados ou pré-difundidos) ou dispositivos lógicos programáveis. Estas tecnologias e metodologias afetam sensivelmente as descrições de projeto sobre o eixo físico do diagrama Y, mas têm pouca influência sobre a manipulação de outros tipos de descrição. O leitor interessado pode se referir a [DeM 94] e [CAL 88] para taxonomias e discussões sucintas de tecnologias e metodologias empregadas atualmente.

O projeto e fabricação de sistemas digitais empregados em grande volume e com ciclos de vida longos, tais como memórias e microprocessadores, seguem regras próprias. Entretanto, muitos sistemas digitais eletrônicos modernos exigem componentes dedicados à realização de uma ou de um conjunto limitado de tarefas. Estes componentes são denominados Circuitos Específicos para uma dada Aplicação (ASICs), e ocupam hoje uma larga fatia do mercado de eletrônica [DeM 94].

A automatização do tratamento de descrições de sistemas digitais desempenha papel primordial na redução do tempo de projeto, bem como na qualidade de ASICs, memórias ou microprocessadores. Entretanto, à medida que a escala de integração de CIs aumenta, cresce a dificuldade de se obter projetos sem erros. Assim, as ferramentas para automatizar o processo de projeto devem estar em constante evolução, para serem capazes de lidar com a crescente complexidade dos módulos usados na construção de sistemas digitais. A indústria de ferramentas de Projeto Auxiliado por Computador (“Computer-Aided Design”, ou CAD) tornou-se hoje, ela própria, um setor substancial da indústria eletrônica global [DeM 94].

Descrevemos a seguir a evolução das ferramentas de CAD, traçando um paralelo entre o papel original destas no processo de projeto e o papel atual, sensivelmente diferente.

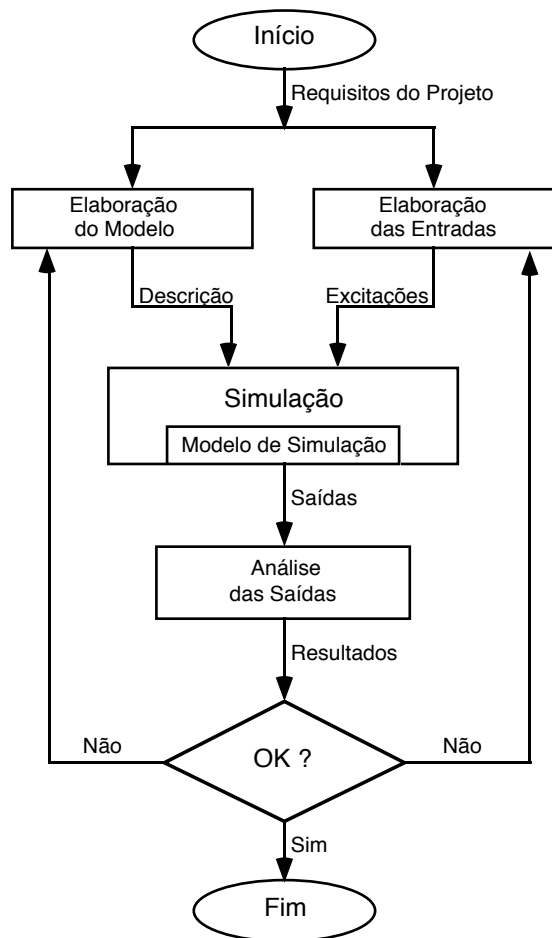
3.1 Ferramentas de Projeto Baseadas em Simulação

Em meados da década de 60, quando o número de componentes de um CI passou da casa das centenas, tornou-se óbvio que o uso de papel milimetrado e régua para desenhar transistores tornar-se-ia rapidamente o gargalo do processo de projeto. Surgiram então ferramentas computacionais para facilitar a captura de descrições físicas de circuitos, os chamados editores de “layout”. Para garantir a correção destas descrições geradas por projetistas humanos (e portanto propensas a erro) com o auxílio de ferramentas de desenho, surgiram ferramentas de verificação de regras de desenho e extratores de circuito (que a partir de uma descrição geométrica geram uma descrição estrutural no nível elétrico de abstração, um diagrama de transistores e elementos lineares).

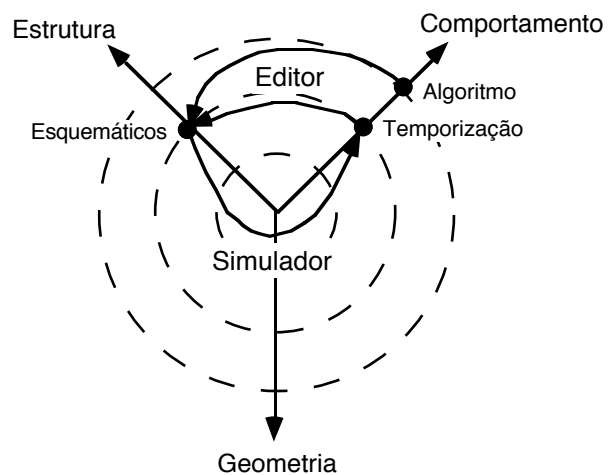
Como garantir que o desenho dos transistores possui a funcionalidade desejada? O dimensionamento de transistores, baseado na física de semicondutores, garante as características de cada componente. Contudo, a complexidade de resolver as equações diferenciais para um número mínimo de componentes, inviabilizava o cálculo para circuitos com cinco ou seis componentes em tempo hábil. Surgiram então os simuladores elétricos como SPICE [BER 81], para automatizar o processo de dimensionamento e de previsão de funcionamento de transistores. Em paralelo, o mesmo ocorreu no nível lógico de abstração, onde surgiram editores de diagramas de portas lógicas e simuladores lógicos.

O estilo de projeto estabelecido por estas primeiras ferramentas é fortemente centrado no talento criativo do ser humano, e aparece ilustrado na Figura 2(a) sob a forma de um fluxograma. Aqui, o projetista deve não somente compreender os modelos abstratos subjacentes ao projeto, mas também deve ser eficiente em descrever um sistema que atenda aos requisitos de projeto (tipicamente uma descrição mais abstrata), e em sugerir entradas relevantes para exercitar a descrição. Uma vez que o projetista humano tenha elaborado uma descrição estrutural e/ou comportamental, ele deve criar um conjunto de entradas para exercitar a descrição, as chamadas excitações. Feito isto, descrição e excitações são entregues ao simulador. Este, com base no modelo de simulação interno computa as

saídas para cada excitação e as fornece como saída. Cabe então ao projetista analisar as saídas e decidir se os resultados satisfazem os requisitos de projeto. Caso negativo, ele deve saber identificar a origem dos problemas. As fontes possíveis são múltiplas: falhas na descrição, falha na geração das entradas, falhas da tecnologia em atender aos requisitos, ou requisitos exigentes demais. Simuladores e ferramentas de captura de descrições possuem papel passivo, sendo empregados pelo projetista para reduzir o tempo de projeto. A Figura 2(b) abaixo descreve um exemplo do processo de projeto no nível lógico, com a descrição simulável assumida estrutural.



(a) Fluxograma de Emprego



(b) Diagrama Y para o Nível Lógico

Figura 2 - Ferramentas de Projeto Baseadas em Simulação

3.2 Ferramentas de Projeto Baseadas em Síntese Automatizada

Métodos de síntese automatizada ótima para equações Booleanas datam das décadas de 50 e 60, mas seu impacto na prática de projeto de sistemas digitais não foi então significativo, tendo permanecido como trabalho de relevância puramente teórica até meados da década de 70. A necessidade de manipular quantidades crescentes de informação durante o projeto de sistemas digitais exigia a consideração constante de novas ferramentas. Estas deveriam transcender as atividades de facilitar a captura e de exercitar descrições, passando a ser capazes de gerar novas descrições de forma automatizada e corretas por construção, baseado nos mesmos requisitos manipulados pelos projetistas.

3.3 Comparação de Ferramentas

No cenário de projeto de sistemas digitais não houve propriamente uma mudança do tipo de ferramentas empregadas, das baseadas em simulação para as baseadas em síntese. O que ocorreu foi uma maior valorização das ferramentas de síntese, que hoje servem de base para o projeto, lugar antes ocupado pelas ferramentas de simulação. A mudança de enfoque iniciou-se em meados da década de 80, quando foi cunhado o termo compilação de silício [GAJ 88]. Este termo designava uma linha de pesquisa que ambicionava possibilitar a geração total ou preponderantemente automática de descrições finais de CIs (e até sistemas digitais completos), a partir de descrições iniciais com alto grau de abstração. A falha desta abordagem foi acreditar que esta tarefa fosse factível para quaisquer aplicações. Nesta Seção comparamos os tipos de ferramentas e vemos a justificativa para a difusão do uso das ferramentas de síntese automatizada.

A diferença mais visível entre simulação e síntese é o fato das descrições simuláveis serem propensas a erros, enquanto que as descrições geradas por ferramentas de síntese são corretas por construção. Esta diferença faz com que o emprego de síntese automatizada forneça uma segurança muito maior com relação à corretude do projeto final, uma característica essencial para minimizar os custos do sistema final. Por outro lado, uma outra consequência do emprego de síntese automatizada reside na rapidez de obtenção de uma descrição sem erros, ao contrário das ferramentas baseadas em simulação, onde a obtenção de uma descrição correta está associada a um processo iterativo de edição de descrições e simulação destas. Sendo rápida a obtenção de soluções corretas, aumenta a possibilidade de explorar o espaço de soluções na busca da solução ótima. Logo, ferramentas de síntese automatizada são mais propensas a aproximar soluções ótimas de projeto, em comparação com ferramentas baseadas em simulação.

Ferramentas de síntese possuem modelos subjacentes que podem ser bastante complexos, o mesmo ocorrendo com ferramentas de simulação. No entanto, o projetista não precisa dominar todos os conceitos associados a estes modelos para explorar os recursos da ferramenta, visto que seu papel é parametrizar a ferramenta para que esta gere descrições corretas. Ele apenas precisa dominar a influência desta parametrização no desempenho dos modelos de síntese. Tomemos como exemplo a ferramenta de minimização lógica ESPRESSO [BRA 84]. Muito poucos projetistas dominam o algoritmo descendente recursivo empregado pelo programa, mas todos seus usuários compreendem os efeitos da ferramenta, a minimização simultânea de um conjunto de equações Booleanas visando uma implementação sob a forma de soma de produtos. Além disto, a parametrização da ferramenta consiste em controlar a velocidade de execução (para possibilitar um compromisso entre qualidade da solução e o tempo necessário para gerá-la) e estabelecer o formato de saída (soma de produtos, produto de somas, fase afirmada ou negada para as saídas, etc), aspectos acessíveis para qualquer usuário, ainda que sua obtenção se faça através de algoritmos complexos.

Os modelos de simulação, por outro lado, precisam ser dominados mais profundamente pelo projetista, pois de sua compreensão depende a qualidade da descrição inicial, bem como a capacidade do projetista de analisar as saídas. Por exemplo, simuladores lógicos empregam álgebras para definir a interação entre os diversos estados de um sinal lógico [HAY 86], uma maneira de garantir a possibilidade de simular efeitos que não possam simplesmente ser obtidos com valores digitais 0 e 1, tais como armazenamento de cargas em capacitâncias, bidirecionalidade de componentes tais como

chaves CMOS, conflito de sinais, etc. Somente o domínio destas álgebras e do seu emprego pode capacitar o projetista a entender se o simulador é capaz de lidar com técnicas de projeto tais como barramentos pré-carregados, lógica dinâmica, etc.

Como conclusão dos dois parágrafos anteriores, o uso de ferramentas de síntese é, em geral, mais simples que o uso de ferramentas de simulação.

Contudo, ferramentas de simulação possuem também um conjunto de vantagens com relação a ferramentas de síntese. Ferramentas baseadas em simulação não pressupõe um estilo de projeto específico, uma vez que cabe ao projetista gerar a descrição simulável. Ferramentas de síntese partem do princípio que as descrições geradas se encaixam em um dos modelos de síntese embutidos. Logo, a flexibilidade para explorar o espaço de soluções é mais baixa que nas ferramentas de simulação. Uma boa ferramenta de síntese deve permitir muitas parametrizações para reduzir sua falta de flexibilidade, enquanto que uma ferramenta de simulação deve apenas ater-se a um modelo de simulação, por mais complexo que este seja. Logo, a implementação de ferramentas de síntese é mais complexa que a implementação de simuladores. Para reforçar ainda mais este último ponto, ferramentas de síntese devem capturar os aspectos mais criativos do processo de projeto (a geração de descrições finais a partir de descrições iniciais), enquanto que os simuladores se limitam a exercitar descrições geradas externamente.

4 Métodos e Ferramentas - Estado da Arte

O processo de projeto de sistemas digitais pode ser dividido em quatro fases, baseado na natureza das técnicas de projeto empregadas para resolvê-los e na dependência destas técnicas da tecnologia de implementação subjacente:

- projeto físico - está relacionado com descrições situadas nas imediações do eixo físico e/ou do nível de abstração elétrico;
- projeto lógico - vinculado à manipulação de descrições posicionadas entre e sobre os eixos comportamental e estrutural e nas imediações do nível lógico de abstração;
- projeto arquitetural - também chamado de projeto de alto nível, manipula descrições entre os eixos comportamental e estrutural e no entorno do nível arquitetural de abstração;
- projeto sistêmico - manipula descrições entre os eixos comportamental e estrutural e no entorno do nível sistêmico de abstração.

A automatização destas fases de projeto vem crescendo muito desde os anos 60, seu desenvolvimento maior sendo sempre atrelado ao emprego de novos métodos no meio industrial. Historicamente a automatização iniciou-se na fase onde um maior volume de informação deve ser manipulado, ou seja, no projeto físico, seguida de perto pelo projeto lógico. A fase de projeto arquitetural recebeu maior atenção a partir dos anos 80, com a difusão do uso de linguagens formais de descrição de sistemas digitais (“Hardware Description Languages”, ou HDLs) na prática de projeto. A fase de projeto sistêmico é uma nova fronteira para ferramentas de automatização, aberta há apenas alguns anos, ao final dos anos 80.

Embora muitos pesquisadores sustentem que as fases de projeto físico e de projeto lógico tenham atingido hoje um alto grau de maturidade, é ainda nesta área que se concentra a maior parte das publicações sobre resultados de pesquisa. Por exemplo, naquela que é considerada uma das conferências de maior importância técnica em ferramentas de projeto para sistemas digitais, a IEEE/ACM International Conference on CAD (ICCAD), em 1994, de 40 sessões técnicas, 11 trataram prioritariamente sobre temas relacionados ao projeto físico, 12 sobre projeto lógico e apenas 3 sobre cada uma das outras fases de projeto, além de 5 sessões sobre o problema do teste de sistemas digitais, 3 sobre projeto de sistemas analógicos, 2 sessões sobre o processo de projeto de sistemas digitais e 1 sessão sobre CAD para tecnologia de CIs.

O desenvolvimento de ferramentas de CAD para sistemas digitais em todas as fases de projeto acima exige conhecimentos profundos de dois campos da matemática de forma prioritária: a teoria de grafos e a álgebra Booleana. Além destes ramos do conhecimento, exige-se de implementadores de ferramentas de projeto conhecimentos não-triviais sobre algoritmos. Referências fundamentais para esta parte de embasamento são: [COR 90], para algoritmos e teoria de grafos e [DAV 78], para a parte de álgebra Booleana para sistemas digitais. A maioria dos subproblemas de otimização encontrados durante o projeto de sistemas digitais pode ser reduzido a um de um pequeno conjunto de problemas fundamentais. Entre estes citamos os problemas de coloração de grafos, cobertura mínima, cliques de um grafo, isomorfismo de grafos e satisfactibilidade Booleana. Todos estes são problemas computacionalmente intratáveis, mas existem técnicas heurísticas eficazes desenvolvidas para atacá-los. A referência fundamental para apresentar problemas intratáveis é [GAR 79]. Soluções exatas e técnicas heurísticas para resolver estes problema são discutidos em [COR 90] em geral, e no escopo de sistemas digitais em [DeM 94].

Apresentamos a seguir um breve apanhado dos problemas mais relevantes em cada fase de projeto, junto com uma bibliografia indicada para aprofundamento posterior.

4.1 Projeto Físico

As grandezas manipuladas durante o projeto físico são em sua maior parte de natureza geométrica, donde o sinônimo de projeto geométrico para esta fase. Uma bibliografia geral para o projeto físico em todos os seus aspectos é [WES 94]. Apesar de limitar-se à tecnologia CMOS de implementação de circuitos e a uma visão de projetista, por oposição a uma visão do construtor de ferramentas de projeto, trata-se de uma excelente obra de referência.

Os principais problemas do projeto físico são o posicionamento de blocos (“placement”) e o roteamento, ou traçado de rotas de interconexão (“routing”). Estes problemas podem receber nomes específicos em níveis de abstração distintos, tais como planejamento de planta baixa (“floor-planning”), mas suas características variam pouco desde o nível elétrico até o nível sistema. O problema de posicionamento consiste em colocar módulos adjacentes uns aos outros, de forma a minimizar a área ocupada e/ou o tempo de propagação dos sinais elétricos dentro do sistema digital. Referências básicas aqui são os algoritmos de posicionamento descritos em [LAU 79] e [SEC 86]. O roteamento toma a saída do posicionamento e uma lista de pontos a unir e cria interconexões entre os blocos. Embora considerada uma tecnologia madura do ponto de vista de circuitos integrados [WES 94], a aplicação destes problemas a dispositivos mais recentes, tais como os FPGAs (“Field-

Programmable Gate Arrays”) [BRO 92], tem gerado renovado interesse por este problema. Referências para o tratamento do problema de roteamento relacionados a CIs são [SEC 86], [RIV 82], [SOU 78] e [SOU 79], enquanto que para FPGAs podemos citar [BRO 93], [CHA 94] e [WU 94].

Além do posicionamento e do roteamento, a geração de macro-células, merece menção. Trata-se de problemas resolvíveis por programas especializados em sintetizar estruturas regulares tais como RAMs, ROMs, PLAs, conjuntos de registradores, multiplicadores e blocos operacionais de processadores. Alguns sistemas criam diretamente o desenho dos transistores, sendo específicos para uma dada tecnologia. Outros criam desenhos simbólicos que podem ser mais tarde desenvolvidos para se adequar a diferentes tecnologias [LAW 88].

4.2 Projeto Lógico

O projeto lógico comporta duas grandes subdivisões, o projeto de circuitos combinacionais e o projeto de circuitos seqüenciais. Todos os sistemas digitais modernos são compostos basicamente de módulos seqüenciais, mas estes são por sua vez compostos de partes combinacionais e partes seqüenciais. Além do mais, as técnicas de implementação de circuitos seqüenciais são geradas em muitos casos pela adaptação ou extensão de técnicas de projeto de circuitos combinacionais.

A mudança de ênfase no estilo de projeto de sistemas digitais, passando de ferramentas baseadas em simulação para as baseadas em síntese automatizada encontrou aqui um vasto campo de desenvolvimento. O domínio de métodos complexos e eficazes de otimização e síntese no nível lógico de abstração deu lugar a várias ferramentas cujas funcionalidades estão hoje incorporadas a boa parte dos sistemas comerciais de CAD.

Sintetizar e otimizar circuitos combinacionais de grande tamanho é uma tarefa bastante complexa e de amplo uso na indústria de sistemas digitais, em especial na implementação de controladores. Como forma de diminuir a dificuldade da tarefa, o espaço de soluções pode ser reduzido caso o tipo de implementação física seja limitado a estruturas regulares do tipo ROMs ou PLAs. A geometria destas estruturas pode ser implementada através de geradores de módulos, conforme discutido na seção anterior. Mais importante, elas correspondem a descrições lógicas comportamentais bem definidas, as funções Booleanas representadas através de formas normais disjuntivas ou conjuntivas, também conhecidas como soma de produtos ou produtos de somas, respectivamente. Este mapeamento direto para entidades abstratas com uma estrutura algébrica determinada possibilita a aplicação de métodos e técnicas derivados do estudo matemático de funções Booleanas. O principal resultado desta estreita relação entre as fases de projeto físico e lógico é a determinação de funções custo extremamente eficazes para avaliar as figuras de mérito da implementação final. Um exemplo de ferramenta desta natureza é o programa ESPRESSO [BRA 84], um minimizador lógico de funções Booleanas desenvolvido na Universidade de Berkeley.

As formas normais referidas acima também são denominadas genericamente de descrições lógicas de dois níveis, devido ao fato de poderem ser implementadas usando dois níveis de portas lógicas. Descrições envolvendo mais de dois níveis são ditas multinível, e seu tratamento automatizado é bem mais complexo. Um exemplo de sistema voltado para a síntese automatizada de lógica multinível é o sistema MIS [BRA 87], da mesma Universidade de Berkeley.

Uma revisão bastante abrangente dos métodos, algoritmos e ferramentas voltados para a síntese e a otimização de circuitos combinacionais pode ser encontrada em [BRA 84] para lógica em dois níveis e em [BRA 90] para lógica multinível.

O projeto de sistemas sequenciais envolve uma série de tarefas que podem ser classificadas em dois grupos: aquelas que partem de descrições comportamentais e aquelas que partem de descrições estruturais [CAL 93].

Dentre as tarefas dirigidas por comportamento, as mais importantes são a minimização de estados e a atribuição ou codificação de estados. O último destes problemas pode, a exemplo da síntese de circuitos combinacionais ser especificado para implementação através de lógica de dois níveis ou para lógica multinível. O intuito aqui é o mesmo, reduzir o espaço de soluções a pesquisar durante o processo de otimização de projeto. Um bom exemplo de método computacional de resolução do problema de minimização de estados é aquele implementado pelo programa STAMINA [HAC 91], desenvolvido na Universidade do Colorado, em Boulder. Uma referência sobre atribuição de estados para lógica em dois níveis é [VIL 90], onde a implementação do programa NOVA é discutida. A proposta do programa MUSE [DU 91], por outro lado, é de realizar atribuição de estados voltada para uma implementação em lógica multinível. Alguns trabalhos recentes sugerem a solução simultânea dos problemas de minimização e atribuição, como por exemplo através do programa ASSTUCE [CAL 93] ou do programa SMAS [AVE 92], visando superar deficiências da resolução separada.

As tarefas dirigidas pela estrutura de circuitos sequenciais são baseadas principalmente na idéia de retemporização (“retiming”) [LEI 91], um conceito que permite reposicionar os elementos de memória de um circuito seqüencial de forma a possibilitar a otimização estrutural.

Boas revisões bibliográficas para as tarefas baseadas em comportamento podem ser encontradas em [AVE 92], [ASH 92], [CAL 93] e [DeM 94]. Para uma revisão das tarefas baseadas em estrutura, ver [DeM 94].

4.3 Projeto Arquitetural

Esta fase também é freqüentemente denominada de projeto de alto nível (“high-level synthesis”). Ao contrário do que ocorre nas fases de projeto anteriores, os métodos formais de descrição arquitetural não eram dominados pelos projetistas até bem pouco tempo atrás. Por exemplo, a maioria destes sempre possuiu familiaridade com as equações diferenciais usadas para descrever o comportamento de dispositivos ativos e passivos, bem como com técnicas de descrição lógica como diagramas de esquemáticos e técnicas de otimização lógica como mapas de Karnaugh. Contudo, até o final dos anos 80, encontrar projetistas capazes de empregar HDLs não constituía tarefa fácil, ainda que linguagens como ISPS [SIE 74] e CDL [CHU 74] tivessem sido propostas e estivessem em uso em meios acadêmicos há mais de uma década. No momento em que a indústria de sistema digitais necessitou empregar tais métodos como única forma de superar a complexidade de projetos do final dos anos 80, surgiram os primeiros esforços sérios de automatização desta fase de projeto. Novas linguagens foram propostas, baseado nas novas necessidades de projeto. Exemplos destas são VHDL [IEE 88], VERILOG [THO 91] e SILAGE [HIL 85] [GAJ 88]. Estas linguagens servem hoje como descrição inicial para a fase de projeto arquitetural.

Uma descrição arquitetural de um sistema digital consiste de um conjunto de operações a serem realizadas e de um conjunto de dependências relacionando estas operações. O projeto arquitetural compreende então dois problemas fundamentais: a alocação (“allocation” ou “binding”) de recursos para realizar as operações e o escalamento (“scheduling”) no tempo destas operações supondo-as realizadas pelos recursos citados. Os recursos aqui mencionados são tipicamente componentes estruturais de sistemas digitais, tais como ULAs, registradores, multiplicadores, etc.

Em geral, a síntese arquitetural pressupõe que os recursos contituem um bloco operacional (“data-path”) acionado por um bloco de controle (“control-unit”) que comanda a temporização das operações, criando um modelo de implementação do tipo mestre-escravo. A saída da síntese arquitetural são estes dois blocos, onde o bloco operacional pode seguidas vezes ser passado diretamente para a fase de projeto físico e onde o bloco de controle, sob a forma de uma máquina de estados finitos [KOH 78], serve de entrada para o projeto lógico. Embora simples, o modelo mestre-escravo pode apresentar gargalos de processamento devido à centralização do controle. A maneira de superar o problema reside em permitir a existência de paralelismo entre as operações realizadas, multiplicando o número de blocos operacionais e/ou de controle.

Exemplos de sistemas voltados para a síntese arquitetural são o Yorktown Silicon Compiler [GAJ 88], elaborado pela IBM e os compiladores CATHEDRAL [CAT 90] implementados pelo instituto belga IMEC e voltados para aplicações específicas na área de CIs para o processamento digital de sinais. Embora estes sistemas enderecem a síntese nas fases de projeto físico e lógico tanto quanto na fase arquitetural (donde a denominação de compiladores de sílcio), seu interesse maior reside na forma de tratamento da síntese arquitetural. Referências gerais para a síntese arquitetural são [GAJ 88], [DeM 94], [McF 90] e [CAM 91].

A síntese de blocos operacionais é uma tarefa passível de ser automatizada desde o projeto arquitetural até o projeto físico, criando ferramentas denominadas compiladores de blocos operacionais (“data-path compilers”). Exemplos de tais sistemas são encontrados em [GAJ 88].

Métodos e ferramentas para resolver o problema de escalamento foram inicialmente baseados em técnicas usadas em programação e pesquisa operacional. Exemplo de um escalador para sistemas digitais seguindo tal filosofia é o constante do sistema System Architect Workbench [THO 90]. Uma das técnicas mais difundidas hoje para resolver este problema é o escalamento dirigido por força, proposto em [PAU 89]. Do lado de algoritmos de alocação, um trabalho seminal é o de Thomas e Leive [THO 83], que abriu as portas para a pesquisa no assunto. Dentro do problema de alocação é importante considerar o compartilhamento de recursos para evitar a explosão do número destes. Um trabalho importante que primeiro tocou o problema foi [HAF 82].

Finalmente, alguns trabalhos têm sido propostos para superar o problema de arquiteturas mestre-escravo, permitindo a consideração de arquiteturas concorrentes, como por exemplo [HES 95].

4.4 Projeto Sistemico

Poucos pesquisadores fazem uma distinção atualmente entre o projeto arquitetural e o projeto sistemico, classificando as tarefas de ambos como parte da chamada fase de projeto de alto nível. Em nossa opinião, contudo, a mudança radical provocada pela consideração do “software” embutido em

um sistema digital complexo durante o projeto do sistema não pode ser negligenciada. A necessidade de descrições independentes do fato de um módulo do sistema ser implementado como um componente de “hardware” ou como um programa executando sobre uma dada arquitetura é uma destas mudanças. Uma linha de pesquisa muito ativa nos últimos anos é a de coprojetos de “hardware” e “software”, baseado na necessidade de visualizar um sistema digital de forma mais completa, uma composição de componentes eletrônicos e de componentes abstratos (os programas).

A especificação de sistemas digitais sob este ponto de vista gerou o emprego de formalismos de descrição tais como CSP/OCCAM [HOA 78], linguagens voltadas para a descrição de processos sequenciais comunicantes.

A tarefa mais importante durante o projeto sistêmico é justamente a determinação de quais partes do comportamento devem ser implementadas como componentes eletrônicos e quais devem ser implementadas sob a forma de programas. Esta tarefa é denominada particionamento “hardware/software”.

Um trabalho bastante recente propôs algoritmos para realizar o particionamento [BAR 93], através do uso de descrições de sistemas digitais baseado no formalismo UNITY. Na mesma referência, pode ser encontrada uma revisão ampla do estado da arte da fase de projeto sistêmico.

5 Perspectivas Futuras e Conclusões

O processo de projeto de sistemas digitais é uma área de pesquisa extremamente ativa. A tendência de aumento da complexidade dos problemas a resolver continua sendo a tônica desta disciplina, devido aos avanços das tecnologias de implementação de dispositivos digitais. Novos dispositivos criam a necessidade de rever as técnicas de síntese e otimização já desenvolvidas, pois sua serventia para estes é difícil de prever. O exemplo mais recente é o dos FPGAs. Sua aparição exigiu uma revisão dos métodos de projeto físico, pois os algoritmos de posicionamento e traçado de rotas dedicados a CIs personalizáveis via fabricação mostraram-se ineficazes quando aplicados a FPGAs. As consequências a nível de projeto lógico têm também sido marcantes, devido à inadequação das funções custo anteriormente empregadas. Ferramentas de síntese automatizada para FPGAs encontram-se hoje em um ‘front’ de pesquisa de alto valor agregado.

Por outro lado, a automatização da fase de projeto sistêmico ainda é incipiente, sendo um campo também aberto e repleto de problemas a serem solucionados.

Do ponto de vista de novas técnicas de manipulação de descrições, a introdução em meados dos anos 80 de uma nova forma canônica de representação de funções Booleanas foi crucial. Esta forma denomina-se diagrama de decisão binário reduzido e ordenado (ROBDD) [BRY 86]. ROBDDs serviram como base de desenvolvimento de métodos para representar conjuntos implicitamente [BER 90]. Estes métodos deram margem a uma grande quantidade de novos trabalhos de pesquisa na fase de projeto lógico, tendo como consequência principal o aumento da complexidade dos problemas tratáveis de forma automatizada em várias ordens de grandeza. A re-expressão de vários problemas já resolvidos via representações implícitas é uma atividade que necessita hoje de um grande esforço de pesquisa, com ganhos em termos de desempenho no limite do imaginável, apenas. A aplicação destas representações em outras fases tais como o projeto físico e o projeto arquitetural é certamente viável,

mas nada ainda foi feito neste sentido.

Dentre os estilos de projeto, breve surgirá um terceiro (além daqueles baseados em simulação e síntese automatizada), que promete ser bastante empregado no futuro. Trata-se do projeto baseado em verificação formal como método de síntese. A aplicação de métodos de representações implícitas à verificação formal deve em breve viabilizar este estilo, antes considerado de complexidade muito alta para ser aplicado na prática. Uma revisão bibliográfica de métodos de verificação formal encontra-se em [GUP 92].

O emprego corrente de FPGAs no contexto da indústria eletrônica nacional poderá viabilizar frutos da pesquisa em áreas como ferramentas de projeto auxiliado por computador, desde que o esforço de pesquisa esteja vinculado à realidade e às necessidades das empresas do ramo. Contudo, só o tempo dirá se esta possibilidade se concretizará em um curto, médio ou mesmo longo prazo.

Bibliografia

- [ASH 92] ASHAR, P., DEVADAS, S., & NEWTON, A. Sequential logic synthesis. Kluwer Academic Publishers, Boston, MA, 1992.
- [AVE 92] AVEDILLO, M. J. Una aproximación al diseño óptimo de máquinas de estados finitos. PhD thesis, Universidad de Sevilla, Facultad de Física, Sevilla, Spain, 1992. (In Spanish).
- [BAR 93] BARROS, E. N. S. Hardware/software partitioning using UNITY. PhD thesis, Tübingen University, Tübingen, Germany, 1993.
- [BER 81] BERKELEY UNIVERSITY. SPICE2G User's Guide. CAD/CAM Department. Jan, 1981.
- [BER 90] BERTHET, C., COUDERT, O., & MADRE, J. C. *New ideas on symbolic manipulation of finite state machines*. In: Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors - ICCD, Cambridge, MA, Sep. 1990.
- [BRA 84] BRAYTON, R. K., HACHTEL, G. D., McMULLEN, C., & SANGIOVANNI-VINCENTELLI, A. L. Logic Minimization Algorithms for VLSI Synthesis. Higham, MA: Kluwer Academic, 1984.
- [BRA 87] BRAYTON, R. K., RUDELL, R., SANGIOVANNI-VINCENTELLI, A. L., & WANG., A. *MIS: a multiple-level logic optimization system*. IEEE Transactions on Computer-Aided Design, CAD-6(6): 1062-1081, Nov. 1987.
- [BRA 90] BRAYTON, R. K., HACHTEL, G. D., & SANGIOVANNI-VINCENTELLI, A. L. *Multilevel logic synthesis*. Proceedings of the IEEE, 78(2): 264-300, Feb. 1990.
- [BRO 92] BROWN, S. D., FRANCIS, R. J., ROSE, J., & VRANESIC, Z. G. Field-programmable gate arrays. Kluwer Academic Publishers, 1992.
- [BRO 93] BROWN, S. D., ROSE, J., & VRANESIC, Z. G. *A stochastic model to predict the routability of field-programmable gate arrays*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 12(12): 1827-1838, Dec. 1993.
- [BRY 81] BRYANT, R. E. *MOSSIM: A switch-level simulator for MOS LSI*. In: Proceedings of the 18th Design Automation Conference, pp. 786-790, 1981.

- [BRY 86] BRYANT, R. E. *Graph-based algorithms for Boolean function manipulation*. IEEE Transactions on Computers, C-35(8): 677-691, Aug. 1986.
- [CAL 88] CALAZANS, N. CIPREDI: Contribuição inicial para um método de concepção de circuitos integrados pré-difundidos. MSc dissertation, Universidade Federal do Rio Grande do Sul, CPGCC-UFRGS, Porto Alegre, Brazil, Jul. 1988. (In Portuguese).
- [CAL 93] CALAZANS, N. State Minimization and State Assignment of Finite State Machines: their relationship and their impact on the implementation. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, Oct. 1993.
- [CAM 91] CAMPOSANO, R, & WOLF, W. (editors). High-level VLSI synthesis. Kluwer Academic Publishers, Boston, MA, 1991.
- [CHA 94] CHANG, Y.-W., THAKUR, S., ZHU, K., & WONG, D. F. *A new global routing algorithm for FPGAs*. In: Proceedings of the International Conference on CAD, San Jose, CA., pp. 356-361, 1994.
- [CHU 74] CHU, Y. *Introducing CDL*. IEEE Computer, New York, 7(12): 31-33, Dec. 1974.
- [COR 90] CORMEN, T. H., LEISERSON, C. E., & RIVEST, R. L. Introduction to Algorithms. McGraw-Hill Book Company, Cambridge, MA, 1990.
- [DAV 78] DAVIO, M., DESCHAMPS, J.-P. , & THAYSE, A. Discrete and Switching Functions. St-Sphorin: Editions Giorgi - McGraw-Hill, 1978.
- [DeM 94] De MICHELI, G. Synthesis and optimization of digital circuits. Mc Graw-Hill, Inc., 1994.
- [DU 91] DU, X., HACHTEL, G., LIN, B. & NEWTON, A. R. *MUSE: a multilevel symbolic encoding algorithm for state assignment*. IEEE Transactions on Computer-Aided Design, 10(1): 28-38, Jan. 1991.
- [GAJ 83] GAJSKI, D. D. & KUHN, R. H. *New VLSI Tools*. IEEE Computer, New York, 16(12): 11-14, Dec. 1983.
- [GAJ 88] GAJSKI, D. Silicon Compilation. Addison-Wesley, Reading (Mass.), 1988.
- [GAR 79] GAREY, M. R., & JOHNSON, D. S. Computers and Intractability: a guide to the theory of NP-completeness. W. H. Freeman and Company, San Francisco, CA, 1979.
- [GEP 95] GEPPERT, L. *Technology 1995: Solid state*. IEEE Spectrum, 32(1): 35-39, Jan. 1995.
- [GUP 92] GUPTA, A. *Formal hardware verification methods: a survey*. Formal Methods in System Design, 1: 151-238, Jan. 1992.
- [HAC 91] HACHTEL, G. D., RHO, J.-K., SOMENZI, F., & JACOBY, R. *Exact and heuristic algorithms for the minimization of incompletely specified state machines*. In: Proceedings of the European Conference on Design Automation - EDAC, pp. 184-191, 1991.
- [HAF 82] HAFER, L., & PARKER, A. *Automated synthesis of digital hardware*. IEEE Transactions on Computers, C-31(2), Feb. 1982.
- [HAY 86] HAYES, J. P. *Digital simulation with multiple logic values*. IEEE Transactions on Computer-Aided Design, CAD-5(2): 274-283, Apr. 1986.

- [HAY 87] HAYES, J. P. *An introduction to switch-level modelling*. IEEE Design & Test, 4(4): 18-25, Aug. 1987.
- [HES 95] HESSEL, F. P. Descrição e síntese de concorrência em VHDL. MSc dissertation, Universidade Federal do Rio Grande do Sul, CPGCC-UFRGS, Porto Alegre, Brazil, 1995. (To be presented in the 2nd quarter, 1995) (In Portuguese).
- [HIL 85] HILFINGER, P. *A high-level language and silicon compiler for digital signal processing*. In: Proceedings of the Custom Integrated Circuits Conference, pp. 213-216, 1985.
- [HOA 78] HOARE, C. A. R. *Communicating sequential processes*. Communications of the ACM, 21(8):666-677, Aug. 1978.
- [IEE 88] THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE Standard VHDL Language Reference Manual. Institute of Electrical and Electronics Engineers, Mar., 1988.
- [JAC 93] JACOBI, R. A study of the application of binary decision diagrams in multilevel logic synthesis. PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, Sep., 1993.
- [KOH 78] KOHAVI, Z. Switching and finite automata theory. McGraw-Hill Book Company, New Delhi, India, second edition, 1978.
- [LAU 79] LAUTHER, U. *A min-cut placement algorithm for general cell assemblies based on a graph*. In: Proceedings of the 16th Design Automation Conference, pp. 1-10, 1979.
- [LAW 88] LAW, H.-F. S., WOOD, G., & LAM, M. *An intelligent composition tools for regular and semiregular VLSI structures*. In: Silicon Compilation. (D. D. Gajski, ed.) Addison-Wesley, Reading (Mass.), 1988.
- [LEI 91] LEISERSON, C., & SAXE, J. *Retiming synchronous circuitry*. Algorithmica, 6:5-35, 1991.
- [LIP 89] LIPSETT, R, SCHAEFER, C. & USSERY, C. VHDL: hardware description and design. Kluwer Academic Publishers, Boston, MA, 1989.
- [McF 90] McFarland, M. J., PARKER, A., & CAMPOSANO, R. *The high-level synthesis of digital systems*. IEEE Proceedings, New York, 78(2): 301-318, Feb. 1990.
- [PAU 89] PAULIN, P., & KNIGHT, J. *Force-directed scheduling for the behavioral synthesis of ASIC's*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, CAD-8(6): 661-679, Jul. 1989.
- [RIV 82] RIVEST, R. L., & FIDUCIA, C. *A greedy channel router*. In: Proceedings of the 19th Design Automation Conference, pp. 418-424, 1982.
- [SOU 78] SOUKUP, J. *Fast maze router*. In: Proceedings of the 15th Design Automation Conference, pp. 100-102, 1978.
- [SOU 79] SOUKUP, J. *Global router*. In: Proceedings of the 16th Design Automation Conference, pp. 481-484, 1979.
- [SEC 86] SECHEN, C., & SANGIOVANNI-VINCENTELLI, A. L. *TimberWolf3.2: a new standard cell placement and global routing package*. In: Proceedings of the 23rd Design Automation Conference, Las Vegas, Nev., pp. 432-439, 1986.

- [SIE 74] SIEWIOREK, D. *Introducing ISPS*. IEEE Computer, New York, 7(12): 39-41, Dec. 1974.
- [THO 83] THOMAS, D., & LEIVE, G. *Automating technology relative logic synthesis and module selection*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, CAD-2(2): 94-105, Apr. 1983.
- [THO 90] THOMAS, D, LAGNESE, E., WALKER, R., NESTOR, J., RAJAN, J., & BLACKBURN, R. Algorithmic and register transfer level synthesis: the System Architect's Workbench. Kluwer Academic Publishers, Boston, MA, 1990.
- [THO 91] THOMAS, D, & MOORBY, P. The VERILOG hardware description language. Kluwer Academic Publishers, Boston, MA, 1991.
- [VIL 90] VILLA, T., & SANGIOVANNI-VINCENTELLI, A. L. *NOVA: state assignment of finite state machines for optimal two-level logic implementation*. IEEE Transactions on Computer-Aided Design, CAD-9(9): 905-924, Sep. 1990.
- [WU 94] WU, Y.-L., & CHANG, D. *On the NP-completeness of regular 2-D FPGA routing architectures and a novel solution*. In: Proceedings of the International Conference on CAD, San Jose, CA., pp. 362-366, 1994.