Verificação Formal de Blocos Complexos em Arquiteturas

Daniel Vidal
RA 099049
dnlvidal@gmail.com

Enos A. V. F. de Lima
RA 032455
vacilotolima@gmail.com

Resumo
No presente artigo é apresentada uma breve comparação entre verificação funcional e a verificação formal (VF). São mostrados os princípios básicos de funcionamento dos métodos formais e sua importância dentro do fluxo de projeto. É feita uma análise de como a complexidade do design pode influenciar a execução bem-sucedida da etapa de VF, são mostrados alguns equívocos comuns com relação ao papel da verificação formal; em seguida, possíveis soluções aos problemas gerados pela complexidade do design são apresentadas, por fim é feita uma conclusão ponderando sobre a utilidade e viabilidade da verificação formal.

Termos Gerais
Algoritmos, Design, Confiabilidade, Linguagens, Verificação.

Palavras-Chave
Microeletrônica, Verificação Formal, Equivalence-Checking, Model-Checking, Assertions, Lógica Difusa.

1. INTRODUÇÃO
É notório que a envergadura funcional dos chips tem crescido de forma estrondosa nos últimos tempos, cada vez mais um determinado chip engloba mais e mais funções, o que naturalmente eleva a complexidade e o tamanho do design.

Funcionalidade é a palavra-chave no resultado do projeto, o padrão de qualidade do circuito é em grande parte medido pela semelhança ou equivalência entre o que foi especificado e o que foi projetado de fato; reconhecidamente a tarefa de se verificar uma determinada funcionalidade é mais complexa do que a funcionalidade em si, assim sendo, a tarefa de verificação funcional tem se tornado um dos maiores gargalos no desenvolvimento de chips, sendo que em média, 70% dos recursos humanos e de tempo são gastos nesta tarefa.

Mesmo com todo o esforço dispendido ainda assim é praticamente impossível aferir que um determinado chip é 100% livre de bugs, ainda hoje os bugs funcionais são os maiores responsáveis por re-spin de chips já fabricados (algo muito caro em termos monetários).

Métodos de verificação funcional são inerentemente métodos baseados em simulação; simulações são sujeitas a três fatores: exaustividade (habilidade de observar/medir todos os possíveis cenários de estímulos), controlabilidade (controle de estimular um ponto/estado específico do circuito), observabilidade (habilidade de observar pontos do circuito), que são virtualmente impossíveis de serem atingidos em sua completude. Os métodos de verificação funcional são bem comportados e de fácil aplicação em descrições RTL, simulações gate-level geralmente são bem mais complexas e demoradas, o que pesa muito negativamente em termos de exautividade; a verificação funcional também demanda muito planejamento tanto na espeficação de estímulos quanto na implementação de algum modelo de cobertura e na implementação dos test-benches.

A verificação formal é um processo sistemático baseado em provas matemáticas para verificar a correspondência entre a especificação e o que foi projetado, intrinsecamente os métodos de verificação formal são estáticos (não são baseados em simulações) e portanto são menos sujeitos aos três fatores mencionados anteriormente. Comparativamente a execução da verificação formal é mais rápida que a verificação funcional, não há nenhum tipo de restrição quanto à aplicação em descrições gate-level e o tempo gasto em planejamento é muito menor. Contudo, métodos de verificação formal também têm suas limitações, a principal delas é que geralmente os métodos são baseados em alguma espécie de suposição ou modelo, tais modelos nem sempre são gerados de forma 100% correta e as suposições, que são feitas pelo verificador, também são passíveis de erro em suas definições.

Nas últimas duas décadas houve um desenvolvimento muito grande tanto em termos teóricos quanto em termos práticos na área de verificação formal. Métodos como equivalence-checking e model-checking tem se desenvolvimento e se estabelecido na indústria. Nas seções adiante estes métodos são detalhados e algumas considerações são feitas à respeito da aplicabilidade e das difculdades geradas pela complexidade do design.

2. MÉTODOS PARA VERIFICAÇÃO FORMAL

2.1 Verificação de equivalência (Equivalence Checking)
Na verificação de equivalencia, o design a ser testado é posto a prova frente a um modelo de referência supostamente correto (*golden model*). Este método é próprio para verificação de equivalência entre descrições em níveis diferentes de abstração.

Esse tipo de verificação faz parte do fluxo de projeto padrão do desenvolvimento de circuitos integrados digitais onde há diversas modificações nos níveis de abstração usados durante a implementação.

Um circuito digital típico é concebido num nível de abstração RTL (Verilog/VHDL). Depois de verificada a funcionalidade frente à especificação utilizando verificação funcional, o modelo RTL é utilizado para sintetizar o circuito digital propriamente dito onde funcionalidade é mapeada para células ou portas lógicas. As células lógicas por sua vez são traduzidas em transistores em silício.

Além de traduções, os circuitos também sofrem modificações como por exemplo a adição de estruturas para teste ou troca de células para performance.

Todas as traduções e modificações, tanto automáticas quando manuais a princípio devem resultar em um circuito equivalente,

porém falhas nas traduções e modificações podem ser introduzidas tanto por bugs de ferramentas automáticas quanto por erro do projetista no caso de modificações manuais.

Uma solução para validar a equivalência é a simulação do circuito depois de cada modificação, porem a simulação a nivel de circuito costuma ser bem mais custosa: é mais difícil de preparar; pode ter tempo de execução muito grande e demandar muitos recursos computacionais no caso de circuitos muito grandes. No caso de descrições de mais baixo nível, como a representação gate-level, a verificação formal pode ser muito mais facilmente implementada do que a verificação funcional podendo ainda ser mais efetiva.

Nota-se que há a necessidade de um modelo de refência, que idealmente é correto. Na prática o modelo é considerado correto depois de verificado por outros métodos.

A Figura 1ilustra o exemplo típico de ulização de equivalence-checking para projetos de circuitos digitais.

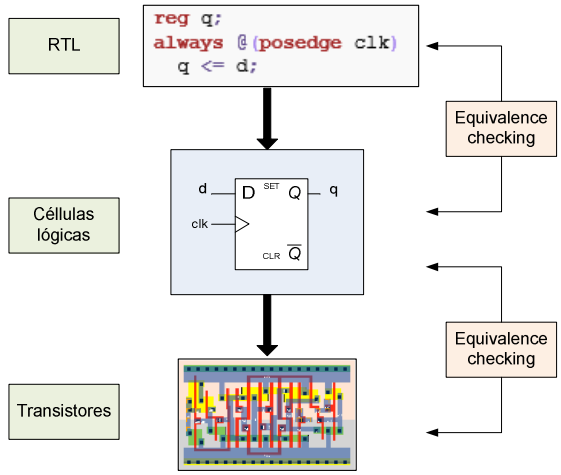

Figura 1: Exemplo de verificação de equivalência.

2.1.1 Equivalencia combinacional.

Somente os elementos combinacionais são verificados. Esse tipo de verificação só pode ser diretamente aplicado quando não há modificação da lógica combinacional entre os elementos sequenciais e não há alteração do número de estados do sistema. Esse tipo de algoritmo é o mais utilizado para verificação de equivalência.

No caso de alterações de modificações na estrutura das células onde a logica combinacional entre registradores é modificada, como no caso de retiming, ou quando não há equivalência aos estados de circuitos combinacionas (ex: máquinas de estado que usam diferentes codificações), a verificação por equivalência combinacional não pode ser diretamente aplicada, ou seja, deve haver uma equivalência estrutural entre o modelo de referência e o design testatdo.

2.1.2 Equivalência sequencial.

Este tipo de algoritmo pode verificar o comportamento de entrada/saida entre dois modelos independente do número de estados possíveis. Sua complexidade é maior que a verificação de equivalência combinacional, o que limita o seu uso em projetos grandes, porém pode ser utilizado em adição ao algoritmo de equivalência combinacional nos casos onde o primeiro não pode ser utilizado como no caso de *retiming*.

2.2 Verficação de modelo

Na verificação de modelo, um sistema de estados finitos é verificado quanto às propriedades que deve satisfazer. Tipicamente os sistemas são contituídos por software, hardware ou protocolos e as especificações trazem requisitos de segurança tais como nunca atingir estados de deadlock ou sempre atingir um determinado estado.

A verificação formal de propriedades ainda não é largamente empregada por que as ferramentas comerciais são relativamente novas e os projetistas ainda não estão muito familiarizados com a técnica.

A ferramenta de análise deve expandir todos os possíveis estados e entradas possívies do sistema e as propriedades devem ser verificadas em todos esses estados.

Caso o modelo criado satisfaça todas as propriedades, o sistema é considerado correto e caso não satisfaça é gerado um contraexemplo onde a propriedade falha para que o erro possa ser facilmente reproduzido, localizado e corrigido no modelo.

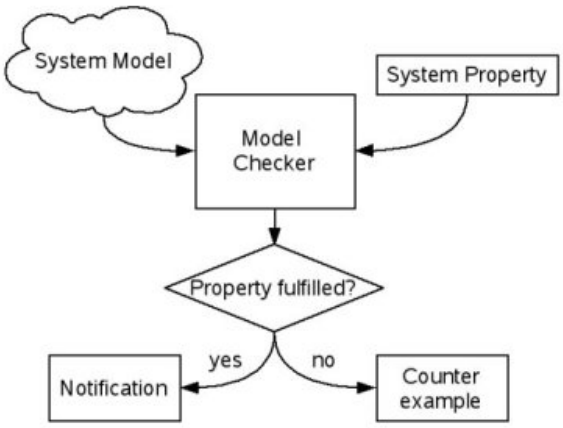

Figura 2: Fluxo de verificação de modelo

Para que um algoritmo possa ser aplicado e a verficação seja feita de modo automático os requerimentos precisam ser escritos em uma linguagem precisa e não ambígua.

No caso de projetos de circuitos digitais, as ferramentas computacionais comerciais normalmente usam assertions para a descrição das propriedades. Na figura abaixo mostra-se um exemplo de *assertions* em duas linguagens largamente utilizadas (*PSL - Property Specification Language; SVA-SystemVerilog Assertion).*

A grande vantagem das assertions é que elas podem servir a um duplo propósito, pois as mesmas podem ser usadas em simulações tradicionais e em análises formais.

PSL:

```
assert always (
  {req && ack} |=>
  {!req within gnt[->1]}
)@(posedge clk);
```

SVA:

```
always @(posedge clk)
assert property (
  req && ack |=>
  (!req within gnt[->1])
);
```

Figura 3: Exemplos de assertions

3. DIFICULDADES ENCONTRADAS NA VERIFICAÇÃO FORMAL DE BLOCOS COMPLEXOS

Nas subseções a seguir são discutidos alguns fatores de complexidade do design e como estes fatores podem refletir na implementação e execução da análise formal.

3.1 Tamanho da Lógica

O primeiro fator de complexidade do design é o tamanho da lógica, quanto maior a lógica maior é o número de estados e maior tende a ser o número de transição entre estes estados, o resultado da verificação depende de se verificar se uma determinada prédica ou regra é valida para o todo o conjunto de estados, assim o número de estados é determinante na complexidade de implementação e execução (pelas ferramentas comercias) da verificação formal.

Conhecidamente, a complexidade de um design cresce de forma exponencial (O(2^n)) com o tamanho do espaço de possíveis entradas e com o tamanho do espaço de possíveis estados, este problema de complexidade de design é conhecido como explosão de estados e se configura como problema principal na verificação formal.

3.2 Complexidade Inerente da Lógica

Descrevemos acima os fatores de dificuldades gerados pelo tamanho da lógica, um fator mais subjetivo e difícil de medir é a complexidade inerente da lógica, por exemplo, um bloco criptográfico pode não ter um tamanho muito grande em número de gates e estados mas pode envolver uma lógica extremamente complexa, este tipo de complexidade pode tornar extremamente difícil a tarefa de se "fabricar" o modelo de comparação do método formal, seja ele baseado em model ou equivalence checking, outro fator é que geralmente as ferramentas não possuem a capacidade de medir e resolver as dificuldades geradas por essa complexidade.

3.3 Complexidade de Interfaces e Protocolos de Comunicação

Um terceiro fator que vale a pena ser mencionado é a complexidade de interfaces e protocolos de comunicação, e o fluxo de dados que um determinado design pode impor de uma interface a outra. Este tipo de interação pode gerar dificuldades muito grandes no uso de property checking usando assertions, pois de foram geral as possíveis entradas checadas por equivalence checking não costumam ser restritas a um grupo válido de entradas referentes a um determinado protocolo.

3.4 Lógica Difusa

Há um quarto fator que tem se tornado foco de muita discussão ultimamente que é a complexidade inserida pela lógica difusa (*pervasive logic*), este termo é usado para referenciar lógicas não ligadas às funções primarias do design, os exemplos mais comuns desse tipo de lógica são os self-tests de memória, lógicas inseridas por cadeias de scan, lógicas ligadas a power-on-reset, sincronismo interno de reset e inserção de interface de debug como JTAG.

Estes tipos de lógica geralmente são espalhadas por todos os blocos do design e portanto não podem ser verificadas de forma isolada. Como nos casos de scan chain, a lógica difusa utiliza os mesmos fios que a estrutura da lógica convencional, gerando um entrelaçamento das duas lógicas. Outro fato que também merece destaque é que a lógica difusa, por envolver grandes porções do design, acaba se estendendo por uma cadeia muito grande de células, ou seja, resulta numa profundidade muito grande em relação ao tamanho de design o que por sua vez pode comprometer a eficiência da análise formal.

3.5 Limitação de Ferramenta

Vale lembrar por último que quanto maiores as complexidades mencionadas, mais será difícil o computo realizado pelas ferramentas, esta dificuldade é diretamente refletida em tempo de execução e uso de memória.

4. CONSIDERAÇÕES SOBRE VERIFICAÇÃO FORMAL

1. Verificação formal não garante que o circuito é 100% correto

   As ferramentas de verificação são complexas e não são verificadas formalmente

   As ferramentas trabalham com modelos e suposições que podem ser incompletos ou que não representam fielmente a implementação física.

2. Verificação formal não substitui totalmente a simulação

   Para níveis mais baixos de abstração, onde as simulações são lentas e requerem mais recursos computacionais a verificação formal é uma ótima alternativa. Entretanto para níveis mais altos de abstração a simulação ainda é uma boa maneira de exercitar o projeto e neste caso a verificação formal deve ser complementar.

3. Pode ser usado em projetos grandes e complexos

   As ferramentas comerciais para verificação de equivalência combinacional são poderosas e maduras para lidar com projetos relativamente grandes porém a verificação de equivalência sequencial e de modelo podem ser limitadas pela complexidade e tamanho do projeto.

4. A verificação formal pode ser usada por qualquer projetista

   Quanto a verificação de equivalência é verdade. As ferramentas são maduras e poderosas e o processo é quase todo automático. Porém a verificação de modelo requer mais habilidade e treinamento especial para a formulação das propriedades.

5. A verificação formal diminui o tempo de projeto

   Substituir a simulação do circuito pela verificação de equivalência nos níveis mais baixos de abstração reduz

drasticamente o tempo de verificação o que torna possível as modificações seguras de última hora.

6. Verificação formal de modelos não é uma tarefa trivial

   A verificação formal com a utilização de modelos é uma atividade relativamente nova para os projetistas e que ainda precisa ser amadurecida.

   Nem sempre é possível descrever todas as propriedades necessárias para que o modelo seja correto.

   Algumas propriedades podem ser escritas de diversas maneiras, o que pode influênciar diretamente os recursos computacionais necessários para a verificação.

5. CONSIDERAÇÕES SOBRE A MITIGAÇÃO DOS PROBLEMAS DE COMPLEXIDADE

Nas subseções a seguir são discutidas algumas soluções para os problemas decorrentes da complexidade do design que foram descritos na seção 3, em especial, são discutidas soluções referentes ao problema da explosão de estados.

5.1 Drivers para VF

Como na verificação funcional, a verificação formal precisa de considerações sobre o ambiente onde o design em questão será inserido, se nenhuma consideração é feita a verificação formal levaria em conta que o design pode ser exposto a qualquer condição de entrada, o que na verdade é uma situação irreal e que causaria uma dificuldade imensa de, exautivamente levar a verificação formal à total completude. Por outro lado, se as corretas considerações são feitas, se as propriedades do protocolo de comunicação são bem descritas, a verificação formal pode alcançar a exaustão de maneira mais fácil de forma a assegurar o funcionamento do design dentro das condições que foram impostas. O driver da verificação formal seria então responsável por prover estímulos válidos de forma que a checagem de propriedades não reporte uma falha em condições não especificadas para o design.

Esta técnica possui grande impacto em mitigar o problema de explosão de estados pelo tamanho do conjunto de entradas, pois restringe o conjunto de entradas a um número menor, porém significativo do ambiente de funcionamento do design. Esta técnica pode ser difícil de ser implementada devido ao problema mencionado na subseção 3.3, mas se implementada de maneira correta, pode também ajudar a mitigar a interdependência do fluxo de dados entre diferentes interfaces, pois casos inválidos não seriam levados em conta.

5.2 Aplicação Serial de Propriedades

Dividir um conjunto de propriedades em múltiplas análises formais pode ser mais otimizado do que realizar poucas análises formais que contenham um conjunto maior de propriedades, pois geralmente uma propriedade temporal corresponde a uma porção de lógica do design, assim, dividindo as propriedades reduz-se o número de estados envolvidos em uma análise com um conjunto reduzido de propriedades.

Geralmente as ferramentas comerciais possuem mecanismos que configurados de maneira correta, podem realizar esta otimização de forma automática.

5.3 Divisão do Espaço de Entradas

O mesmo conceito usado na divisão de propriedades pode ser usado para se dividir o espaço de entradas em diversas análises formais. Neste caso, como o número de entradas é reduzido, a complexidade da análise também é reduzida.

Como no caso anterior, as ferramentas também têm mecanismos de implementação deste tipo de otimização. Também é importante mencionar que os mecanismos até agora mencionados não prejudicam a cobertura da verificação formal, por que mesmo com uma amarração de uma determinada propriedade ou de um determinado subconjunto de entradas, as outras variações são completamente cobertas.

5.4 Redução Manual de Profundidade de Data Path

Imagine que se quer realizar a verificação formal de um buffer, claramente, quanto maior o tamanho do buffer maior será o seu data path e o seu espaço de estados; se a descrição do buffer for parametrizada, pode-se realizar a análise formal em um buffer definido com um tamanho menor, sem perda de analogia e com um alcance e facilidade muito maior na análise.

5.5 Redução de Lógica Não Influente

Esta técnica também é recorrente em ferramentas comerciais e consiste em escalonar a análise de forma que para cada propriedade sejam analisadas somente as porções lógicas do design que de alguma forma são correlatas com tal propriedade, as lógicas que não influenciam a propriedade em análise são desconsideradas, de forma que o espaço de estados considerados para cada propriedade é drasticamente reduzido sem perdas de cobertura. Sem esta técnica, muitos casos de análise seriam inviáveis.

5.6 Localização

Esta técnica geralmente é realizada pelas ferramentas de forma oculta e transparente. A técnica da localização consiste em desligar parte da lógica do design que é responsável por *drivar* alguma propriedade e substituir tal lógica por um "estímulo" aleatório, se mesmo em presença desta aleatoriedade a propriedade em questão ainda assim for verdadeira, a ferramenta pode extrapolar e garantir que a propriedade foi checada com sucesso, se a contrário, a propriedade falhar, a ferramenta tentará restringir a aleatoriedade do estímulo até que a propriedade seja verdadeira ou que seja realmente checada como falsa.

5.7 Combinação de Algoritmos

A combinação de diferentes algoritmos tem se mostrado como fundamental na resolução de todos os problemas mencionados nesse artigo e muitos que não foram mencionados, existem muitos algoritmos e conceitos que são usados e que não foram mencionados, cada qual podendo ter uma melhor performance para uma gama ou uma classe específica de problema. A idéia é combinar estas capacidades de forma a otimizar o alcance de cobertura da análise formal tendo com variáveis de custo o tempo e memória utilizada. Os problemas gerados pela complexidade da lógica difusa frequentemente requerem este tipo de abordagem.

6. CONCLUSÃO

A verificação formal se apresenta como uma técnica complementar à verificação funcional tradicional, principalmente em projetos de alta complexidade, visto que as duas técnicas têm suas vantagens e limitações, e que se balanceadas de forma

inteligente, podem ser complementares em suas capacidades, de forma a facilitar um maior alcance da verificação.

A verificação formal possue uma boa gama de métodos e algoritmos que permite alcançar bons resultados, mesmo frente às dificuldades impostas pela complexidade do design.

A verificação formal não é um assunto muito maduro na indústria de semicondutores, mas que pode e muito provavelmente irá amadurecer com o surgimento de novas técnicas, com o aperfeiçoamento das ferramentas e com a adoção crescente deste conhecimento no fluxo de projeto de circuitos integrados.

7. BIBLIOGRAFIA

[1] Bruce Wile, John C. Goss, Wolfgang Roesner, "Comprehensive Functional Verification: The Complete Industry Cycle", Morgan Kaufmann

[2] Roope Kaivola and Naren Narasimhan, "Formal Verification on the Pentium 4 Floating-Point Multiplier"

[3] "Enabling Large-Scale Pervasive Logic Verification through Multi-Algorithmic Formal", IBM

[4] Gitanjali Swarny, "Formal Verification of Digital Systems", Mentor Graphic Corp

[5] Viresh Paruthi, "Large Scale Application of Formal Verification: From Fiction to Fact", IBM Systems and Technology Group

[6] Jianhua Wang, Jinbo Shao, Yingmei Li, Jinfeng Din, "Survey on Formal Verification Methods for Digital IC", Harbin Normal University

[7] Cristiano Rodrigues, "A Case Study for formal verification of a Timing Co-Processor", BSTC Freescale Semiconductor Inc.

[8] Ramayya Kummar, "Formal Verification of Hardware: Misconception and Reality", Verysis Design Automation, Inc.

[9] Nathaniel Ayewah, Nikhil Kikkeri and Peter-Michael Seidel "Challenges in the Formal Verification of Complete State-of-the-Art Processors", Southern Methodist University, Computer Science and Engineering, Dallas, TX

[10] Roope Kaivola, Naren Narasimhan, "Formal Verification of Intel Pentium 4 Floating-Point Multiplier", Intel Corporation

[11] Taeshin Park, "Implicit Model Checking: Formal Verification Technique for Large-Scale Discrete Systems", MC Research & Innovation Center, Inc.

[12] Cadence - *Encounter Conformal Equivalence Checker* - *http://www.cadence.com/products/ld/equivalence_checker/pages/default.aspx* - *acessado 21/06/2011*

[13] Synopsys – *Formality* - http://www.synopsys.com/tools/verification/formalequivalence/pages/formality.aspx - acessado 21/06/2011